IOSR Journal of Electronics and Communication Engineering (IOSR-JECE)
e-ISSN: 2278-2834,p- ISSN: 2278-8735.Volume 8, Issue 5 (Nov. - Dec. 2013), PP 01-09
www.iosrjournals.org

Axi To Apb Interface Design Using Verilog

* Kalluri Usha[1], T.Ashok Kumar[2]
[1] *PG Student (M. Tech), Dept. of ECE, Chirala Engineering College, Chirala, A.P, India.*
[2] *Associate Professor, Dept. of ECE, Chirala Engineering College, Chirala, A.P, India.*

Abstract: *Protocols are commonly used today to connect IP blocks on structured SoCs. Generally Protocol is the back-bone of the SoC and its failure usually leads to a non-functional chip. In present market, various types of standard protocols are available and are used in SoC which requires a bridge to pass the information from one type of protocol to other type of protocol safely and without any data loss.*

Advanced eXtensible Interface (AXI 4.0) and Advanced Peripheral Bus (APB4.0) are commonly used in a microprocessor bus-architecture. In this work, the bus bridge was designed to interface these protocols which plays a vital role in SoC application such as it may lead to application failure, if it doesn't work properly. Initially basic AXI 4.0 and APB4.0 protocols are modelled separately using VHDL and are simulated. Basically Bus Bridge should convert command and data of AXI 4.0 formats to acceptable APB4.0 formats. This conversion does not ensure proper communication unless the timings of each protocol were met. Hence the interconnecting Bus Bridge wrapper between Advanced eXtensible Interface (AXI 4.0) and Advanced Peripheral Bus (APB4.0)) was designed with proper timing delay.

In this paper, the simulation result shows that the communication between AXI 4.0 and APB4.0 through the bridge is proper. All of the commands and data are successfully transferred from AXI 4.0 to APB4.0 protocol by the bridge. There is no loss of data or control information. The Bus Bridge supports the simple write and read operation and the burst extension.

I. Introduction

Advanced eXtensible Interface (AXI) , the third generation of AMBA interface defined in the AMBA 4 specification, is targeted at high performance, high clock frequency system designs and includes features which make it very suitable for high speed sub-micron interconnect:
• 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
APB is designed for low bandwidth control accesses, for example register interfaces on system peripherals. This bus has an address and data phase similar to AHB, but a much reduced low complexity signal list, for example no bursts.

Features

AMBA 4.0 supports the following features.
➢ High performance
➢ Burst transfers
➢ Split transactions
➢ Single edge clock operation
➢ SEQ, NONSEQ, BUSY, and IDLE Transfer Types
➢ Programmable number of idle cycles

Specification

The following points should be considered when reading the AMBA 4.0 specification
• Technology independence
• Electrical characteristics
• Timing specification

Technology Independence

AMBA 4.0 is a technology-independent on-chip protocol. The specification only details the bus protocol at the clock cycle level

Electrical Characteristics

No information regarding the electrical characteristics is supplied within the AMBA 4.0 specification as this will be entirely dependent on the manufacturing process technology that is selected for the design.

Timing Specification

The AMBA 4.0 protocol defines the behavior of various signals at the cycle level. The exact timing requirements will depend on the process technology used and the frequency of operation. Because the exact timing requirements are not defined by the AMBA 4.0 protocol, the system integrator is given maximum flexibility in allocating the signal timing budget amongst the various modules on the bus.

AXI Data Bus Width

One way to improve bus bandwidth without increasing the frequency of operation is to make the data path of the on-chip bus wider. Both the increased layers of metal and the use of large on-chip memory blocks (such as Embedded DRAM) are driving factors which encourage the use of wider on-chip buses.

Specifying a fixed width of bus will mean that in many cases the width of the bus is not optimal for the application. Therefore an approach has been adopted which allows flexibility of the width of bus, but still ensures that modules are highly portable between designs.

The protocol allows for the AXI data bus to be 8, 16, 32, 64, 128, 256, 512 or 1024-bits wide. However, it is recommended that a minimum bus width of 32 bits is used and it is expected that a maximum of 256 bits will be adequate for almost all applications.

For both read and write transfers the receiving module must select the data from the correct byte lane on the bus. Replication of data across all byte lanes is not required.

II. AXI Bus Master

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

Here the Arbiter signals and the transfer signals were mentioned in the below data flow AXI bus master interface diagram.

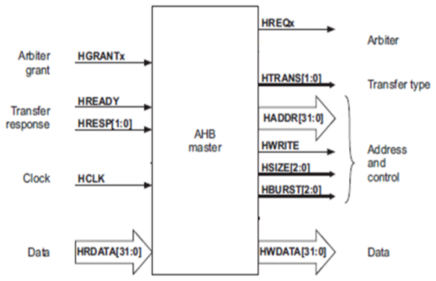

Figure 1 AXI Bus Master

AXI Bus Slave

An AXI bus slave responds to transfers initiated by bus masters within the system. The slave uses a HSELx select signal from the decoder to determine when it should respond to a bus transfer. All other signals required for the transfer, such as the address and control information, will be generated by the bus master.

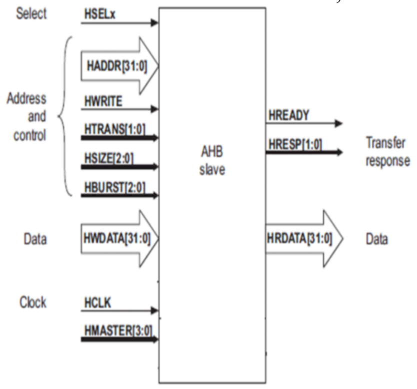

Figure 2 AXI Bus Slave

Hence all the signals involved in the slave and decoder were mentioned in the above AXI bus slave interface diagram.

In the simplest implementation of a multi-layer system, each master has its own AXI Layer and is connected to the slave devices by an interconnect matrix, as shown in Figure 3

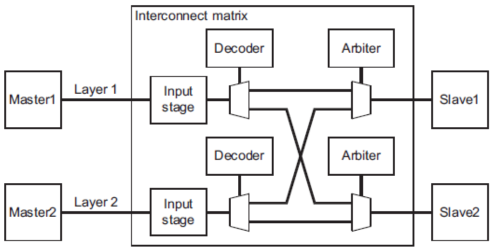

Figure 3 Master Slave interconnection

Within the interconnect matrix:
1. Every layer has a Decode stage that determines which slave is required for a transfer.
2. A mux routes the transfer from the appropriate layer to the required slave. If two layers require access to the same slave at the same time, the arbitration within the interconnect matrix must determine which layer has highest priority. The layer that is not given access is waited using HREADY until it is given access to the required slave. When a layer is waited an Input Stage is used to store a copy of the pipelined address and control information until the access to the shared slave is given. Each slave port has its own arbitration and a number of different schemes can be used.
For example:
• Input layers can be serviced in round-robin manner, changing every transfer or every burst
• The arbitration can use a fixed priority scheme where certain high priority layers are always given access in preference to lower priority layers.
The number of input/output ports on the interconnect matrix is completely flexible and can be adapted to suit the system requirements.

Bus configurations

As the number of masters and slaves in a system increases, the complexity of the interconnect matrix can become significant. You can use the following techniques to optimize the system architecture:
• Figure 4 shows how you can make slaves local, or private, to a particular layer. This reduces the complexity of the interconnect matrix. By using this when it is acceptable that a slave can only be accessed by masters on the same layer.

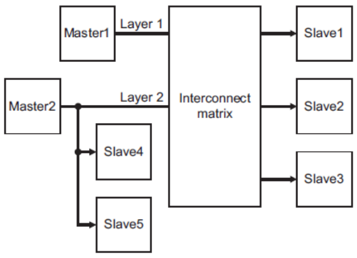

Figure 4 Grouping Masters and Slaves

• Figure 5 shows how you can make multiple slaves appear as a single slave to the interconnect matrix. This is useful to combine a number of low-bandwidth slaves. You can also use it where a set of slaves are normally accessed by just one master, such as a DMA controller, and the interconnect matrix is used only to give access to other masters under special circumstances, such as debugging the system.

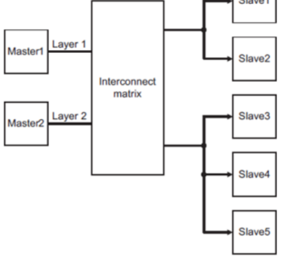

Figure 5 Multiple slaves shown as a single slave

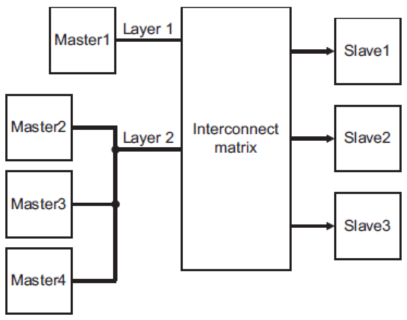

Figure 6 Multiple Masters Sharing a Layer

• In the simplest implementation of a multi-layer system, each master has its own layer. Figure 6 shows how you can also build a system where multiple masters share a layer. This is well suited to combine masters that have low-bandwidth requirements or masters, such as the Test Interface Controller (TIC), that have particular characteristics.

• Each layer can be a complete AXI subsystem, with the interconnect matrix being used to enable communication between the two subsystems. The example in Figure 7 shows only a single slave is shared. Typically this is an on-chip memory slave that is used as a buffer area between the two subsystems.

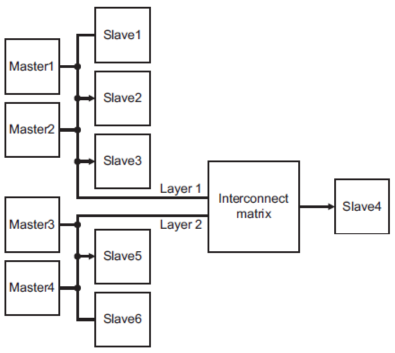

Figure 7 Single Slave Shared

Multi-port slaves

In a multi-layer AXI system certain slaves, such as an SDRAM controller, are able to operate more efficiently by processing transfers from different layers in parallel. Figure 8 shows how you can do this by designing the slave with multiple AXI slave ports.

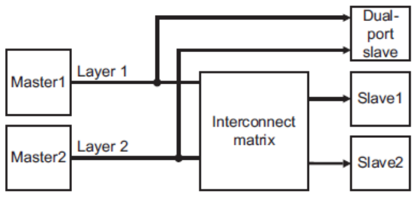

Figure 8 Multiple AXI Slave ports

The Advanced Microcontroller Bus Architecture (AMBA) was introduced by ARM Ltd in 1996 and is widely used as the on-chip bus in System-on-a-chip (SoC) designs. AMTA is a registered trademark of ARM Ltd. The first AMBA buses were Advanced System Bus (ASB) and Advanced Peripheral Bus (APB). In its 2nd version, AMBA 2, ARM added AMBA High-performance Bus (AHB) that is a single clock-edge protocol. In 2003, ARM introduced the 3rd generation, AMBA 3, including AXI to reach even higher performance interconnect and the Advanced Trace Bus (ATB) as part of the CoreSight on-chip debug and trace solution. These protocols are today the de-facto standard for 32-bit embedded processors because they are well documented and can be used without royalties. Some manufacturers utilize AMBA buses for non-ARM designs. As an example Infineon uses an AMBA bus for the ADM5120 SoC based on the MIPS architecture.

The important aspect of a SoC is not only which components or blocks it houses, but also how they are interconnected. AMBA is a solution for the blocks to interface with each other.

The objective of the AMBA 4.0 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
• minimize silicon infrastructure while supporting high performance and low power on-chip communication
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 smart phones.

III. AMBA Products

A family of synthesizable intellectual property (IP) cores AMBA Products licensable from ARM Limited that implement a digital highway in an SoC for the efficient moving and storing of data using the AMBA protocol specifications. The AMBA family includes AMBA Network Interconnect (NIC-301), SDRAM and FLASH memory controllers (DMC-34x, SMC-35x), DMA controllers (DMA-230, DMA-330), level 2 cache controllers (L2C-310), etc.

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.
The AMBA 3.0 specification defines five buses/interfaces:
• Advanced extensible Interface (AXI)
• Advanced High-performance Bus (AHB)
• Advanced System Bus (ASB)
• Advanced Peripheral Bus (APB)
• Advanced Trace Bus (ATB)

Block Diagram
In this study, we focused mainly on the implementation aspect of an AXI4-Lite to APB bridge. The APB bridge provides an interface between the high-performance AXI domain and the low-power APB domain. It appears as a slave on AXI bus but as a master on APB that can acces up to sixteen slave peripherals. Read and write transfers on the AXI bus are converted into corresponding transfers on the APB. The AXI4- Lite to APB bridge clock diagram is shown in Figure. 9.

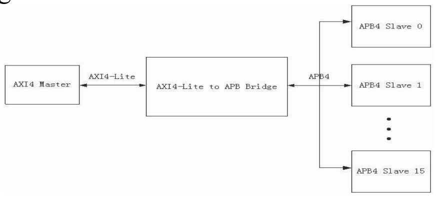

Figure 9 AXI4- Lite to APB bridge clock diagram

Signal Connections
Figure. 10 shows the component signal connections. The bridge uses:

• AMBA AXI-Lite signals as described in the AMBA AXI-Lite 4.0 protocol specification.
• AMBA APB signals as described in the AMBA APB 4.0 protocol specification

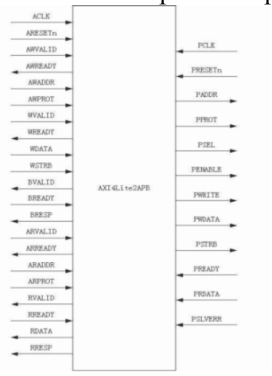

Figure 10 Signal specifications & Connections

AXI Handshake Mechanism

In AXI 4.0 specification, each channel has a VALID and READY signals for handshaking. The source asserts VALID when the control information or data is available. The destination asserts READY when it can accept the control information or data. Transfer occurs only when both the VALID and READY are asserted.

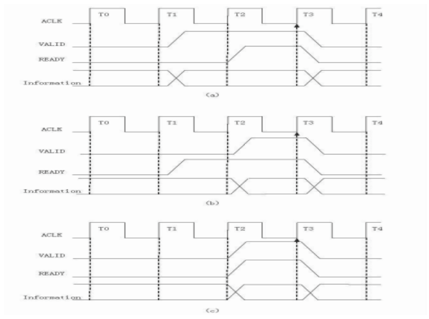

Figure 11 VALID & READY Cycle Timing Diagrams

The AHB2APB interfaces the AHB to the APB. It buffers address, control and data from the AHB, drives the APB peripherals and returns data and response signals to the AHB. It decodes the address using an internal address map to select the peripheral. The AHB2APB is designed to operate when the APB and AHB clocks have the same frequency and phase.In addition to supporting all basic requirements for AMBA compliance, the AHB2APB provides enhancements to the APB protocol. With AHB2APB, an APB peripheral can insert wait states by holding its pready signal low for the required number of cycles.In the AMBA specification, APB accesses are word-wide (32 bits). The AHB2APB provides the signal pbyte_enable[3:0] to allow byte and half-word accesses. These can be used by the APB peripheral as necessary. The AHB2APB does not perform any alignment of the data, but transfers data from the AHB to the APB for write cycles and from the APB to the AHB for read cycles.The AHB2APB does not support burst transfers and converts any burst transfers into a series of APB accesses.

IV. Results & Conclusions

Presently there is much other architecture along with AMBA AXI- APB, but with its high performance, speed and reliability its usage is widely increasing. The AMBA-AXI architecture is now only confined to ARM Company, but due to AMBA this proposed design AXI-APB advantages, many other companies are moving towards implementing this architecture.

Presently the AMBA APB is using the 8 bit transfer per clock period but this rate can further increased with the improving technology. The configurations of the SM arbitration scheme with the maximum performance need to be found automatically during run time. In this paper the Xilinx ISE EDA Tool is used for synthesis and Modelsim is used for simulation. In future The AMBA Interface may applicable to AMBA next versions. The following Figures 12,13 & 14 are the simulation results of the proposed system.

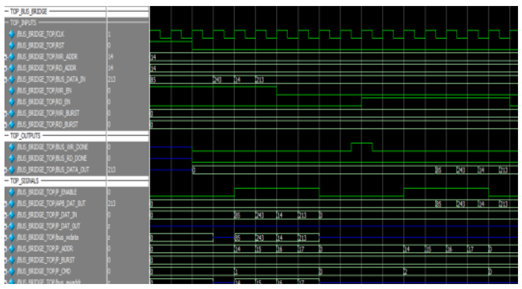

Figure 12 a. Simulation Result for Top Module

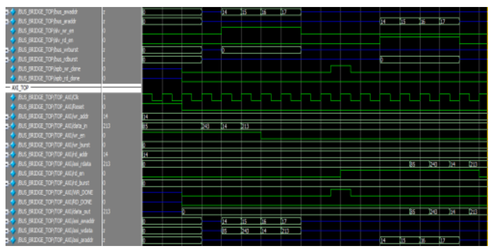

Figure 13 b. Simulation Result for Top Module

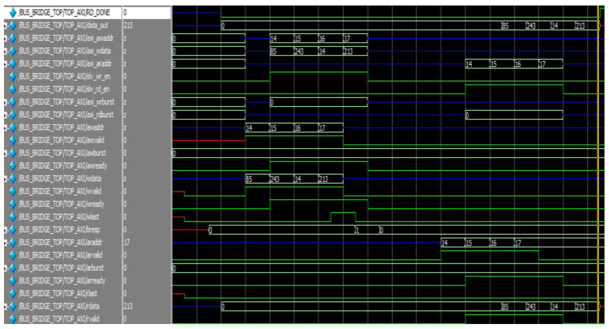
Figure 14 c. Simulation Result for Top AXI MOdule

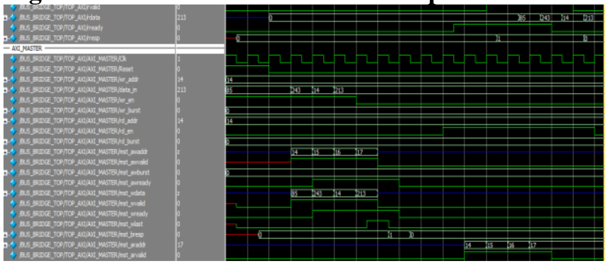
Figure 15 d. Simulation Result for Top AXI Master

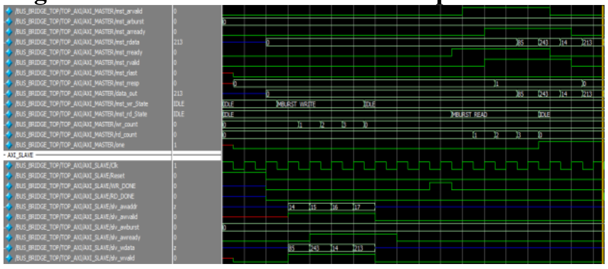
Figure 16 e. Simulation Result for Top AXI Slave MOdule

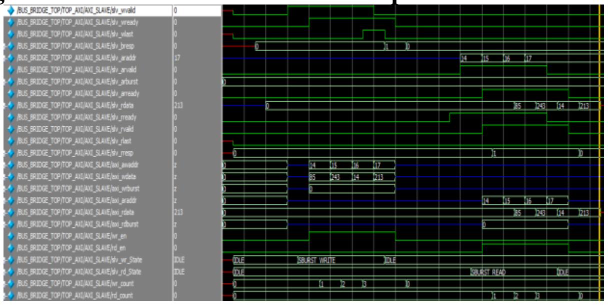
Figure 13 a. Simulation Result for APB Top MOdule

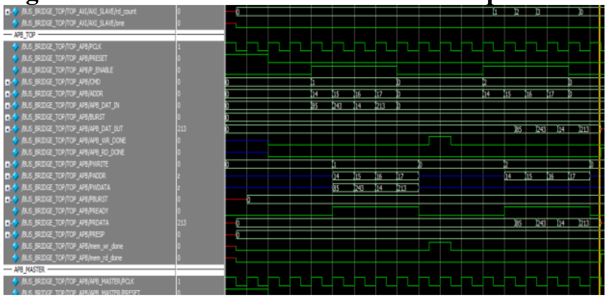
Figure 17 b. Simulation Result for APB Master MOdule

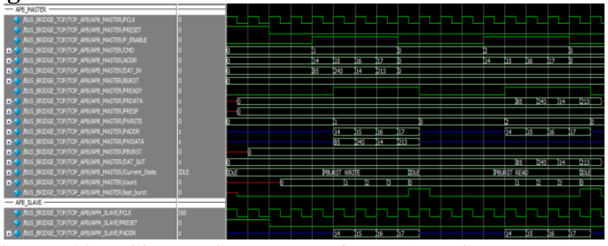
Figure 18 c. Simulation Result for p APB Slave MOdule

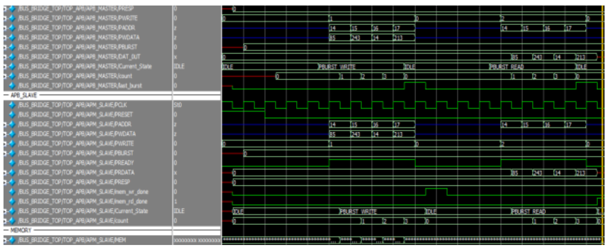

Figure 19 d. Simulation Result for p APB Slave MOdule

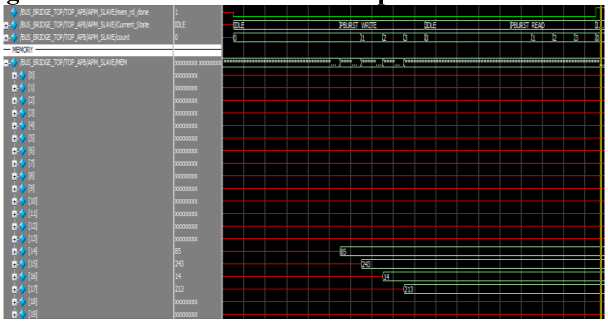

Figure 20 e. Simulation Result for Memory Module

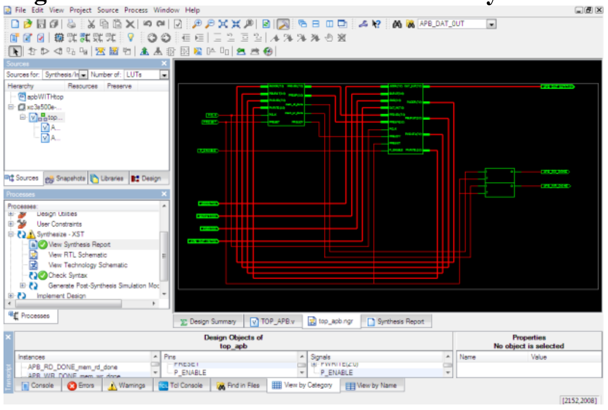

Figure 214. Simulation Result for Memory Module

SYNTHESIS REPORT
Number of Slices: 0 out of 4656 0%
Number of bonded IOBs: 11 out of 232 4%
Total latency: 5.020ns (4.275ns logic, 0.745ns route) (85.2% logic, 14.8% route)
Total memory usage is 185052 kilobytes

Acknowledgements

The authors would like to thank the anonymous reviewers for their comments which were very helpful in improving the quality and presentation of this paper.

References:

[1] Clifford E. Cummings, "Synthesis and Scripting Techniques for Designing Multi-Asynchronous Clock Designs," SNUG 2001
[2] Chris Spear, "SystemVerilog for Verification, 2nd Edition", Springer, www.springeronline.com, 2008.
[3] Lahir, K., Raghunathan A., Lakshminarayana G., "LOTTERYBUS: a new high-performance communication architecture for system-on-chip deisgns," in Proceedings of Design Automation Conference, 2001.
[4] Sanghun Lee, Chanho Lee, Hyuk-Jae Lee, "A new multi-channel onchip- bus architecture for system-on-chips," in Proceedings of IEEE international SOC Conference, September 2004.
[5] Martino Ruggiero, Rederico Angiolini, Francesco Poletti, Davide Bertozzi, Luca 86
[6] Benini, Roberto Zafalon, "Scalability Analysis of Evolving SoC Interconnect Protocols," Int. Symposium on System-on-Chip, 2004.
[7] Lukai Cai, Daniel Gajski, "Transaction level modeling: an overview," in Proceedings of the 1st IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis, October 2003.
[8] Min-Chi Tsai, "Smart Memory Controller Design for Video Applications," Master thesis: National Chiao Tung University, July 2006.

Authors Profile:

Kalluri Usha is Pursuing her M. Tech from Chirala Engineering College, Chirala in the department of Electronics & Communications Engineering (ECE) with specialization in VLSI & Embedded Systems

T. Ashok Kumar is working as an Associate Professor in the department of Electronics & Communication Engineering in Chirala Engineering College,Chirala. He has six years of teaching experience.