Available online www.ejaet.com

European Journal of Advances in Engineering and Technology, 2016, 3(8): 60-66

Research Article ISSN: 2394 - 658X

SPI Slave Controller for AMBA Based SOC

Mukthi SL and AR Aswatha

Department of Electrical and Electronics Engineering
Jain University, Bangalore, Karnataka, India
chakshu.son@gmail.com

ABSTRACT

SPI Controller is one of the high performance system bus slaves and provides a simple software interface to the SPI peripherals and the CPU. The AHB slave main function is an interface unit that allows AHB logic to initiate a data transfer on the AHB. The AHB specifies the type transaction to be executed on the slave through a user friendly interface. The SPI Controller provides access to devices with SPI Interface and performs Read Operation, Write Operation or Read/Write Operation. In this paper we are designing a SPI Memory Controller which is compatible with the Advanced Microcontroller Bus Architecture (AMBA) developed by ARM. We are developing a SPI Module which can operate based on the Advanced High Performance Bus (AHB) signals. Aim lies in the Interfacing of the SPI controller to the ARM Processor using the AMBA AHB Bus. Synthesis and simulation is carried out to verify the functionality of the design.

Keywords: Advanced Microcontroller Bus Architecture (AMBA), Direct Memory Access (DMA), Advanced High Performance Bus (AHB), Serial Peripheral Interface (SPI), ARM

INTRODUCTION

An AMBA based microcontroller typically consists of a high performance system backbone bus, able to sustain the external memory bandwidth, on which the CPU on-chip memory and other direct memory access [DMA] devices reside. This bus provides a high bandwidth interface between the elements that are involved in the majority of transfers also located on the high performance bus is a bridge to the lower bandwidth APB, where most of the peripheral devices in the system are located. SPI Controller is one of the high performance system bus slaves. The AHB slave main function is an interface unit that allows AHB logic to initiate a data transfer on the AHB. The AHB specifies the type transaction to be executed on the slave through a user friendly interface. Serial Peripheral Interface (SPI) is one of the widely accepted communication interfaces. It was developed by Motorola. SPI protocol has become a standard but does not have officially released specification or agreed by any International committee. This gives some flexibility in Implementing SPI Protocol in the Electronic devices but also requires programmable flexibility of the host micro controller. The SPI Controller provides access to devices with SPI Interface. It can perform Read Operation, Write Operation or Read/Write Operation.

We are designing a SPI Memory Controller which is compatible with the Advanced Microcontroller Bus Architecture (AMBA) developed by ARM. We are developing a SPI Module which can operate based on the Advanced High Performance Bus (AHB) signals.

PROPOSED DESIGN

SPI slave memory controller is AMBA AHB compliant [2]; the memory controller is interfaced to advanced high performance bus in the AMBA SoC.SPI Serial Controller is a SPI Slave which transfers data between the ARM Processor and Serial Memory Devices. The communication between AHB BUS and the slave module is through AHB protocol and the communication between the slave module and serial flash device is through SPI protocol. Hence the module translates the AMBA signals from AHB bus and communicates to the serial flash memory using SPI protocol facilitating efficient data transfer [1]. The internal architecture is shown in Fig. 1.
The architecture is divided mainly into 4 blocks namely the –

1. AHB Slave Interface 2. Register Bank 3. Control Logic and 4. SPI Slave Interface

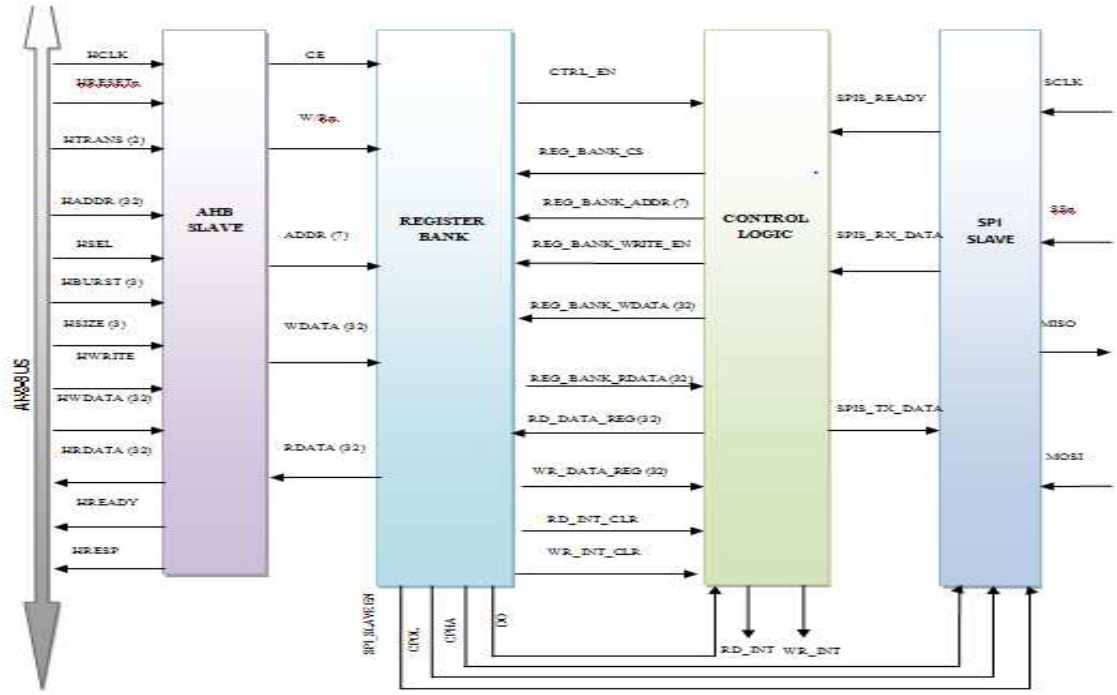

Fig. 1 Internal architecture

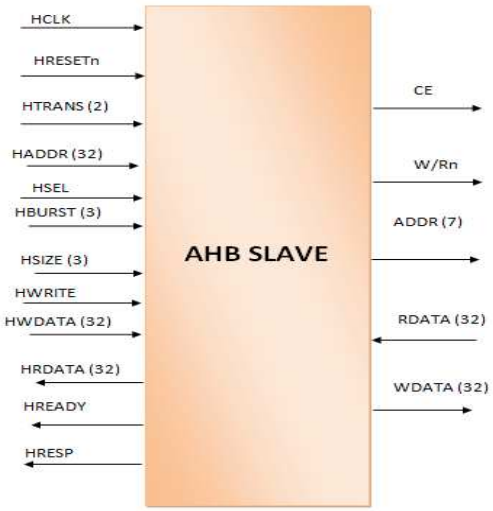

Fig. 2 AHB SLAVE I/O diagram

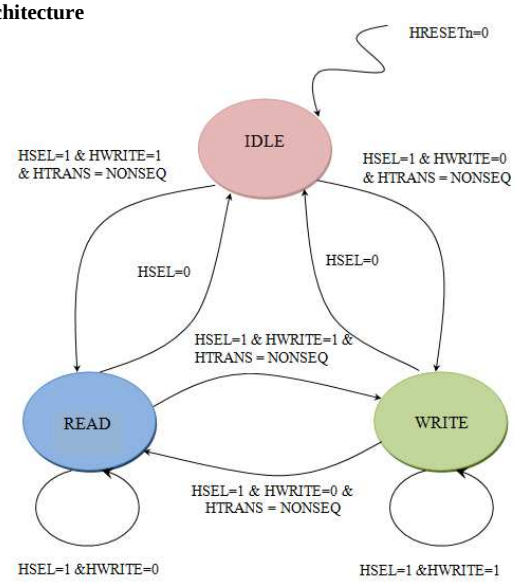

Fig. 3 AHB Slave State Diagram

AHB Slave Design

Fig. 2 shows the top level I/O diagram of AHB slave interface with register bank. AHB SLAVE features are as follows:

- It provides the interface between the AHB bus and rest of the module.
- It responds to the global, control, address and data signals from AHB MASTER and takes necessary actions to be done by the module.
- AHB requires 2 cycles for address and data separately in address and data phase but register bank requires only one clock cycle for both read and write operations.
- An AHB bus slave responds to transfers initiated by bus masters within the system.
- The slave uses a HSEL 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.

The state diagram for AHB slave is shown in Fig. 3. It has 3 states namely Idle State, Read State and Write State. In idle state AHB slave will be in reset condition. Whenever the system is reset irrespective of current state AHB slave

will make transition to idle state. If hsel is removed again AHB slave will go to idle state. In read state AHB slave will perform read operation. The module will read the data from the register bank and will be transfer it to AHB master. If hsel is available and htrans is non-sequential depending on hwrite AHB slave will make transition to read or write state. In write state AHB slave will perform write operation. The module will capture the data provided by the AHB master and writes into the register bank at the provided address.

Register Bank

The Fig. 4 shows the top level I/O diagram of register bank. The register consists of mainly two blocks:

1. The direct addressing block
2. The indirect addressing block

The direct addressing block consists of five registers. They are: Configuration register, read data register, write data register, interrupt clear register and the address register. These registers are used to store the data and the control information transmitted or received between the two terminals. The indirect addressing block stores the 32 bit address of external memory locations. The AHB slave has only a read access to these registers, while the control logic has only writes to these registers. The Memory Organization in Register Bank is shown in the Fig. 5.

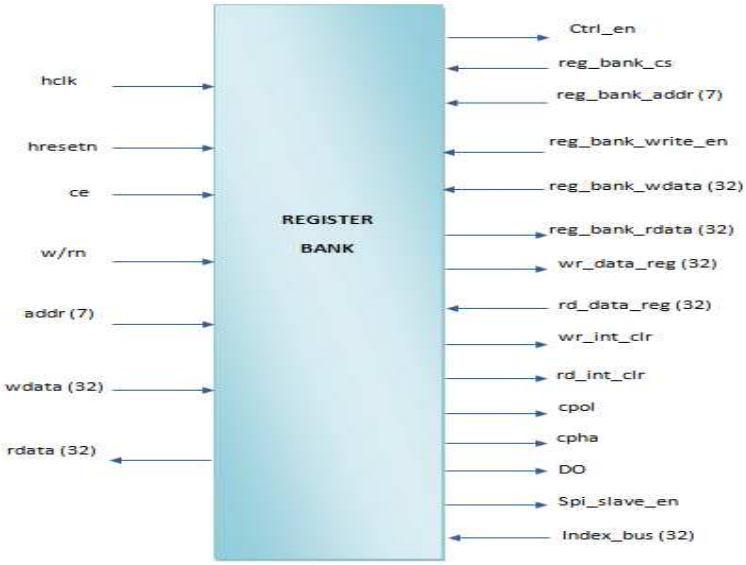

Fig. 4 Register bank I/O diagram

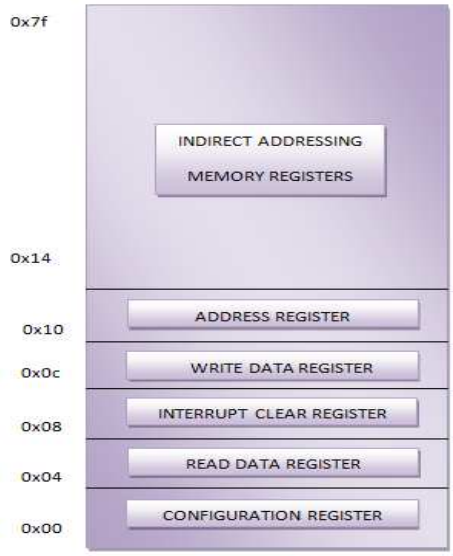

Fig. 5: Memory Organisation in Register Bank

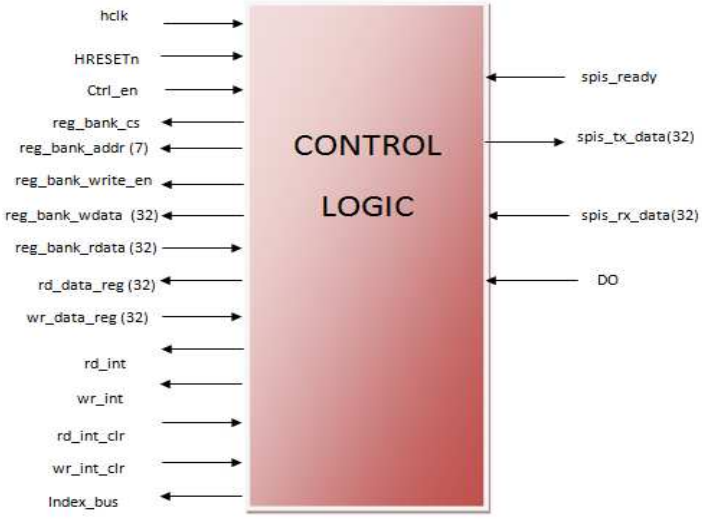

Fig. 6: Control Logic I/O diagram

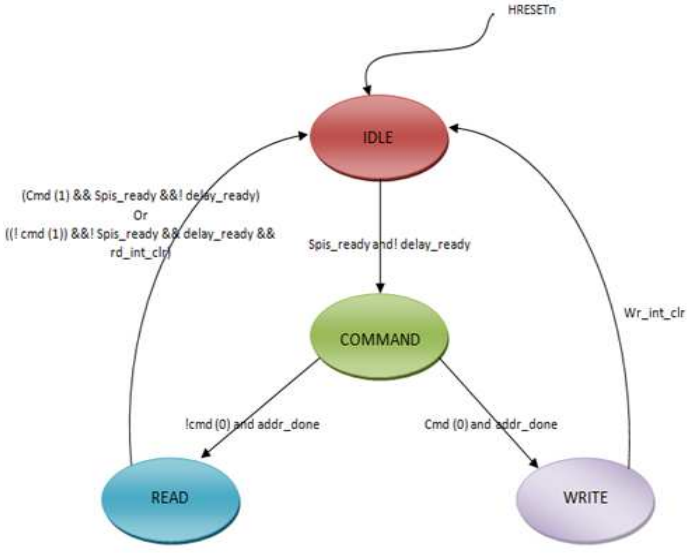

Fig. 7 Control Logic State diagram

Control Logic

The control logic controls and coordinates the operations of the SPI Slave Controller module. The control logic decodes the command from the SPI slave block and directs the module to operate in the following three modes of operations –

1. Direct Read 2. Indirect Read 3. Indirect Write

The Control logic is modelled as a Finite State Machine, with the IDLE state as the default state. The first 32 bit block of data that is received from the SPI slave block when the control logic is in its IDLE state is assumed to be the Command block, with the two LS bits representing the mode as operation and the seven MS bits representing the Register address in the Register bank.

In the DIRECT READ mode:

- The Control Logic receives the Command block – 0000000xxxxxxxxxxxxxxxxxxxxxxx00 or 0000001xxxxxxxxxxxxxxxxxxxxxxx00, from the SPI Slave block, representing the configuration register or the read data register respectively.
- The Control Logic points either to the configuration or the read data register depending on the register address field from the command block.
- The Control Logic receives the contents of the corresponding register and it transmits it to the SPI slave block.

DIRECT WRITE mode is a invalid mode of operation because the Configuration and the Read Data Register (Direct Registers) are having READ only access from the SPI Slave Block.

In the INDIRECT READ mode:

- The Control Logic receives the Command block – AAAAAAAxxxxxxxxxxxxxxxxxxxxxxx10 from the SPI Slave block.
- The Control Logic receives the indirect address in the next data block from the SPI slave block which is stored in the register in the register bank having the address AAAAAAA.
- The Control Logic receives the data contents pointed by the corresponding register from the read data register which is transmitted to the SPI slave block.

In the INDIRECT WRITE mode:

- The Control Logic receives the Command block – AAAAAAAxxxxxxxxxxxxxxxxxxxxxxx10 from the SPI Slave block.
- The Control Logic receives the indirect address in the next data block from the SPI slave block which is stored in the register in the register bank having the address AAAAAAA.
- The Control Logic receives the data from the SPI slave block which is written into the write data register of the register bank.

The Fig. 6 shows the Control Logic I/O diagram. The control logic is modeled as a Finite State Machine [4] and it is bound to have the following four states:

1. IDLE state 2. COMMAND state 3. READ state 4. WRITE state

The IDLE state is the default state of the control logic block. The control logic powers up in the IDLE state after returning from reset as well as after the termination of the READ and WRITE operation. The register bank chip is not enabled for access from the control logic in the IDLE state. A short pulse from the SPI slave block called spis_ready is required for the transition from the IDLE state to the COMMAND state. The command block is received in the COMMAND state which is decoded to identify the mode of operation and the register bank address index. The control logic receives another 32 bit block which represents the indirect address in the case of Indirect mode of operation. Depending on the CMD field of the command block the transition occurs to the READ or the WRITE state. In the Read state, the control logic receives a block of data from the Register bank which is then transmitted to the SPI Slave block. A spis_ready pulse is required for transition from the READ State back to the IDLE state in the case of direct read operation. In the Indirect read operation the control logic interrupts (rd_int) the host and hence a spis_ready pulse along with read interrupt clear signal is required for transition back to the IDLE state. In the Write state, the control logic sends a block of data from the SPI Slave block which is then transmitted to the Register bank. The control logic interrupts (wr_int) the host in the write state and hence a spis_ready pulse along with write interrupt clear signal is required for transition back to the IDLE state.

SPI Slave

SPI slave is used to transform the data serially and communicate with the master when required. It forms the interface between control logic and SPI master. The data is transferred to SPI master in accordance with SPI protocol. The I/O Diagram of SPI Slave is shown in the Fig. 8. SPI slave is enabled by spis_slave_en signal which is routed from register bank and is in accordance with configuration register provided by AHB master. The data is loaded into transmission buffer when it has to be transferred to SPI master through spis_tx_data which is of 32 bits. This data is shifted out to SPI master through miso. All this process occurs when SPI slave is selected by master by pulling down the SSn signal which is active low. Similarly data from SPI master is loaded into reception register through mosi and is transferred to register bank using Spis_rx_data which is of 32 bits. The Spis_ready is asserted once all 32 bits are transmitted to spi master or 32 bits are received from SPI master. Cpol and Cpha signals decide

the mode of communication between spi slave and SPI master. The data transmission and reception process is always with respect to sclk provided by SPI master and this is done by configuring the spi slave initially with respect to frequency supported by SPI slave.

Transmissions normally involve two shift registers of some given word size, such as eight bits, one in the master and one in the slave; they are connected in a ring. Data are usually shifted out with the most significant bit first, while shifting a new least significant bit into the same register. After that register has been shifted out, the master and slave have exchanged register values. Then each device takes that value and does something with it, such as writing it to memory. If there are more data to exchange, the shift registers are loaded with new data and the process repeats. Transmissions may involve any number of clock cycles. When there are no more data to be transmitted, the master stops toggling its clock. Normally, it then deselects the slave. Transmissions often consist of 32-bits of data, and a master can initiate multiple such transmissions if it wishes/needs.

Every slave the bus that hasn't been activated using its chip select line must disregard the input clock and MOSI signals, and must not drive MISO. The master must select only one slave at a time.

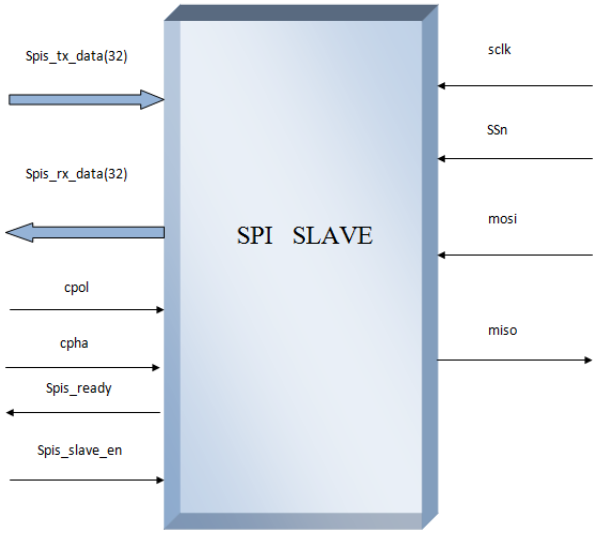

Fig. 8 SPI SLAVE I/O diagram

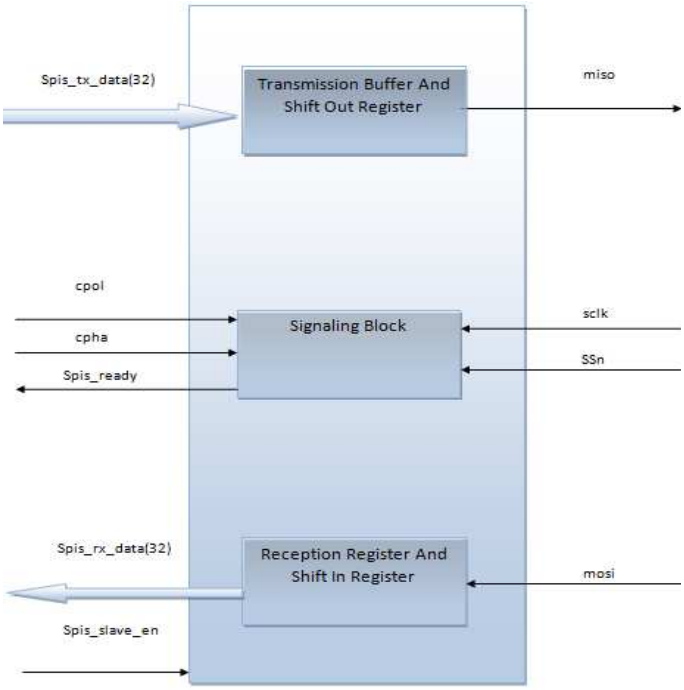

Fig. 9: SPI SLAVE internal architecture

RESULTS

The following snapshots shows the simulation results obtained from ModelSim and the RTL synthesis report is also illustrated in Table 1.

Table -1 Synthesis Report of the Proposed Design

Parameters	AHB interface	Register bank	Control logic	SPI slave	Integrated model
Number of Slices	8 out of 3584 - 0%	4371 out of 3584 - 121%	100 out of 3584-2%	145 out of 3584-4%	4450 out of 3584-126%
Number of Slice Flip Flops	12 out of 7168 - 0%	1180 out of 7168 - 16%	145 out of 7168 - 2%	160 out of 7168-2%	1480 out of 7168-20%
Number of 4 input LUTs	12 out of 7168 - 0%	8598 out of 7168 - 119%	155 out of 7168-2%	219 out of 7168 - 3%	8927 out of 7168 -124%
Number of bonded IOBs	151 out of 141-107%	248 out of 141-175%	240 out of 141-170%	72 out of 141-51%	84 out of 141-59%
Number of GLKs	1 out of 8-12%	1 out of 8-12%	1 out of 8-12%	1 out of 8-12%	1 out of 8-12%

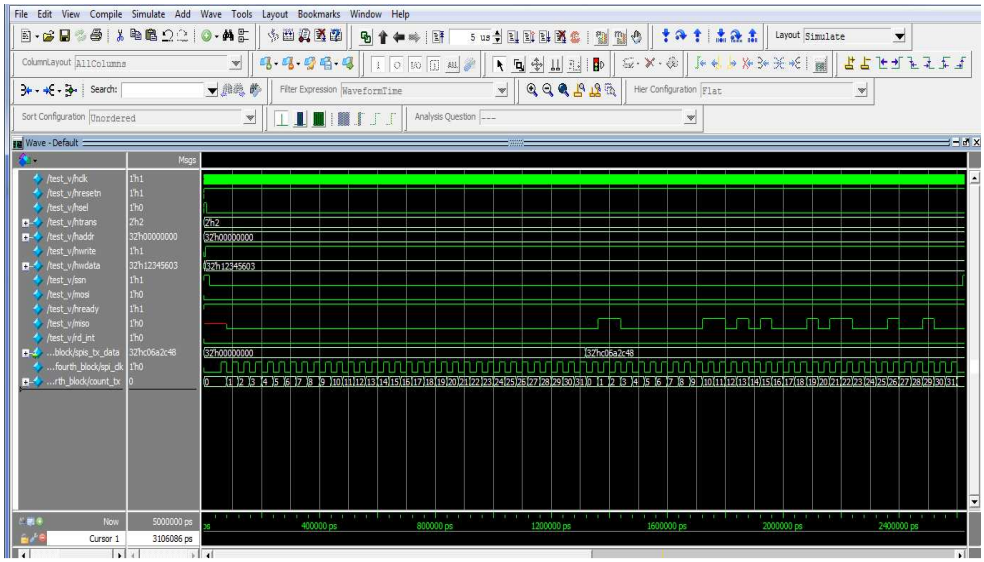

Fig. 10 Simulation waveforms for Direct Read

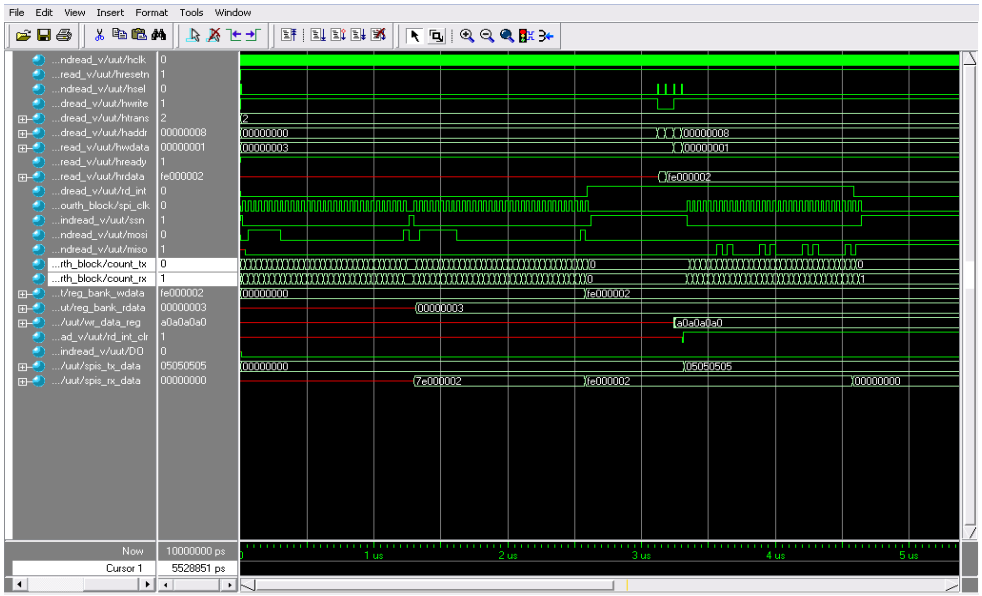

Fig. 11 Simulations waveforms for Indirect Read

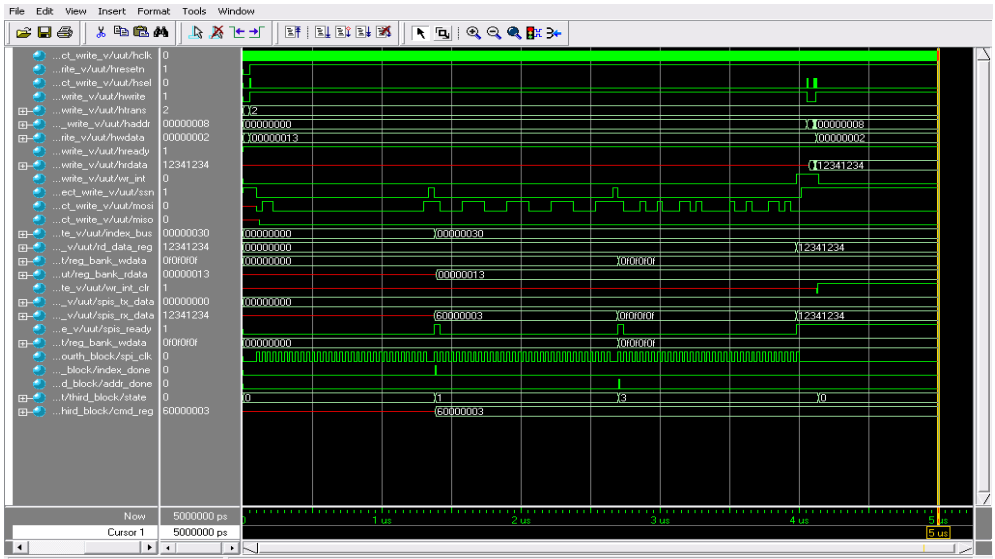

Fig. 12 Simulation waveforms for Indirect Write

CONCLUSIONS

A serial memory controller using Serial Peripheral Interface has been designed. AHB Bus interface is used for communication with processor and the Serial Peripheral Interface. CPU can conFig. the device by providing the configuration and will set the mode of operation and data order. Control Logic will receive the command from the SPI Master and will decode the received command. Direct read, indirect read or indirect write operation can be performed depending on the command. During the indirect read and indirect write operations the processor can be interrupted and processor will access the data or provide the data. The future enhancements are data size which can be made compatible for variable sizes, the data block can be extended to support block data transfer in proposed design it supports only one data transfer, the AHB driver can be used and the design can be implemented in the FPGA kit and the AHB slave can be designed to support the burst type of data and hence do burst of data transfers.

REFERENCES

[1] ARM Corporation, *AMBA specification* 2.0, Reference Manual, **2006.**
[2] *AMBA Open Specifications*- ARM - *http://www.arm.com*
[3] SPI- *www.en.wikipedia.org/wiki/Seria_Peripheral_Interface_Bus*
[4] John F Wakerly, *Digital Design Principles and Practises*, Third Edition, Prentice Hall Publication, **2000.**
[5] J Basker and A Verilog, *HDL Primer*, 3rd Edition, **2003.**
[6] J Noseworthy, Efficient Communications between an Embedded Processor and Reconfigurable Logic on FPGA, *IEEE Transaction on Very Large Scale Integration (VLSI) Systems*, **2008,** 16 (8), 1083-1090.
[7] J Basker and A Verilog, *HDL Synthesis* 1st Edition, **1998.**
[8] Nazeih M Botros, *HDL Programming* (VHDL and VERILOG), Dreamtech Press, **2009.**