Design and Verification of AMBA APB Protocol

Shankar
School of Engineering and Technology,
ITM University, Gurgaon, India

Dipti Girdhar
Department of EECE
ITM University, Gurgaon,India

Neeraj Kr. Shukla, Ph.D
Department of EECE
ITM University, Gurgaon, India

ABSTRACT
The SoC (System on Chip) uses AMBA (Advanced Microcontroller Bus Architecture) as an on chip bus. APB (Advanced Peripheral Bus) is one of the components of the AMBA bus architecture. APB is low bandwidth and low performance bus used to connect the peripherals like UART, Keypad, Timer and other peripheral devices to the bus architecture. This paper introduces the AMBA APB bus architecture design. The design is created using the verilog HDL and is tested by a verilog testbench. This design is verified using UVM (Universal Verification Methodology).

Keywords
AMBA, APB, SoC, UVM, Design, Verification.

1. INTRODUCTION
In system on a chip (SoC) design, Advanced Microcontroller Bus Architecture (AMBA) is used as on chip bus. Earlier it was used in the microcontroller devices but now it is widely used in a large range of ASIC and SoC parts including the application processors used in modern portable mobile devices like smartphones. AMBA is an open standard, on-chip interconnect specification for the purpose of connecting and managing functional blocks in a System-on-Chip (SoC). It helps in right first time development of the multiprocessor designs with large number of controllers and peripherals.

As seen in the Figure.1, AMBA bus architecture consists of three components, namely Advanced High Performance Bus (AHB), Advanced System Bus (ASB), Advanced Peripheral Bus (APB)[3]. AMBA AHB or ASB is high performance bus and has higher bandwidth. So the components requiring higher bandwidth like High Bandwidth on chip RAM, High-performance ARM processor, High Bandwidth Memory Interface and DMA bus master are connected to the AHB or ASB. AMBA APB is low bandwidth and low performance bus. So, the components requiring lower bandwidth like the peripheral devices such as UART, Keypad, Timer and PIO (Peripheral Input Output) devices are connected to the APB.
The bridge connects the high performance AHB or ASB bus to the APB bus[4]. So, for APB the bridge acts as the master and all the devices connected on the APB bus acts as the slave. The component on the high performance bus initiates the transactions and transfer them to the peripherals connected on the APB. So, at a time the bridge is used for communication between the high performance bus and the peripheral devices.

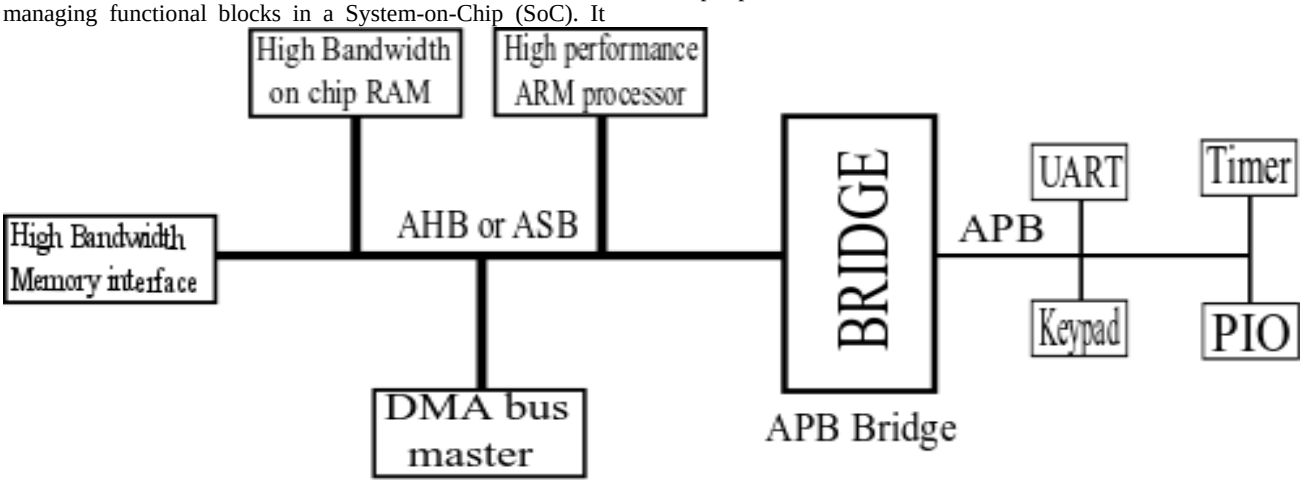

Fig 1: AMBA Bus Architecture[1]

2. APB DESIGN
The APB is the member of the AMBA 3 protocol family which implements a low cost interface which minimizes the power consumption and reduces the interface complexity. Since APB has unpipelined protocol. Therefore, it interfaces to the low bandwidth peripherals that do not demand the high performance of the pipelined bus interface. All the signal transitions are associated with the rising edge of the clock which makes it simple to integrate APB peripherals into any design flow. APB can interface with the AMBA AHB-Lite and AMBA Advanced Extensible Interface(AXI). APB can also be used to access the programmable control registers of the peripheral devices.

2.1 APB Block Diagram
The Advanced peripheral bus (APB) is designed as per the design specification.[2]. The basic block diagram of the AMBA APB in figure.2 shows the basic interface signals.

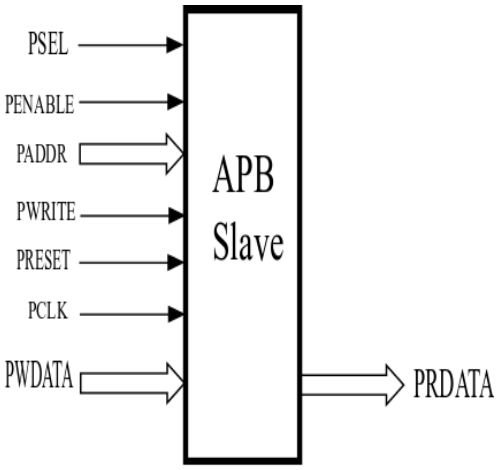

Fig 2: Basic block diagram of APB[2]

The APB slave takes PCLK, PRESET, PSEL, PENABLE, PWRITE as input control signals and PADDR, PWDATA as 32 bits inputs from the bridge and provides 32 bits PRDATA as output.

Table 1. List of APB signals

Signal	Signal Name
PCLK	Clock signal
PRESETn	Reset signal
PADDR	32 bit address bus
PSELx	Select signal
PENABLE	Enable signal
PWRITE	Direction signal
PWDATA	32 bit Write Data bus
PREADY	Ready signal
PRDATA	32 bits read data bus

2.2 APB Operating States

Figure.3 shows the basic state machine that represents operation of the peripheral bus. There are three states namely, IDLE, SETUP and ACCESS state

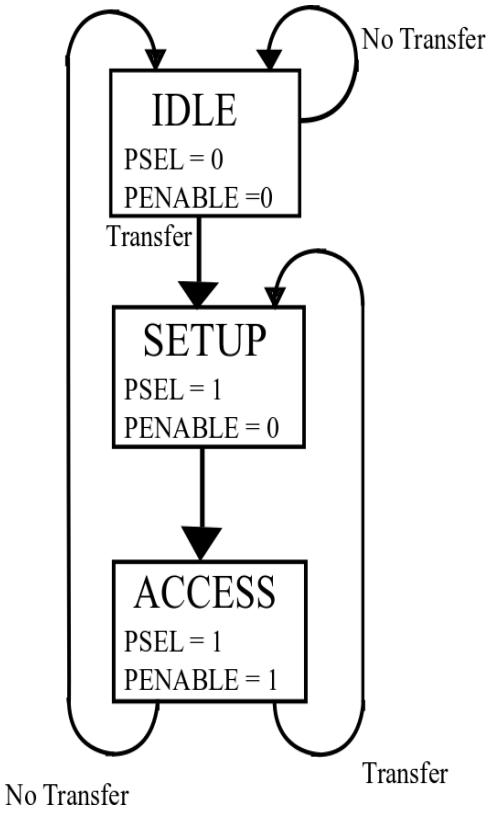

Fig 3: State Diagram[1]

IDLE state is the default state in which no operation is being performed. The assertion of the PSEL signal indicates the beginning of the SETUP phase. The bus enters into the SETUP phase when the data transfer is required. The PWRITE, PADDR and PWDATA are also provided during this phase. The bus remains in the SETUP phase for one clock cycle and on the next rising edge of the clock, the bus will move to the ACCESS state.

The assertion of the PENABLE signal indicates the start of the ACCESS phase. All the control signals, address, and the data signals remains stable during the transition from the SETUP phase to the ACCESS phase. In case of read operation the PRDATA is present on the bus during this phase. PENABLE signal also remain high for one clock cycle. If no further data transfer is required, the bus will move the IDLE state. But, if further data transfer is required then the bus will move to the SETUP phase.

2.3 Write Cycle

During the write transfer operation, the PSEL, PWRITE, PADDR and PWDATA signals are asserted at the T1 clock edge which is called the SETUP cycle. At the next rising edge of the clock T2, the PENABLE signal and PREADY signal are asserted. This is called the ACCESS cycle. At the clock edge T3, PENABLE signal is disabled and if further data transfer is required, a high to low transition occurs on the PREADY signal.

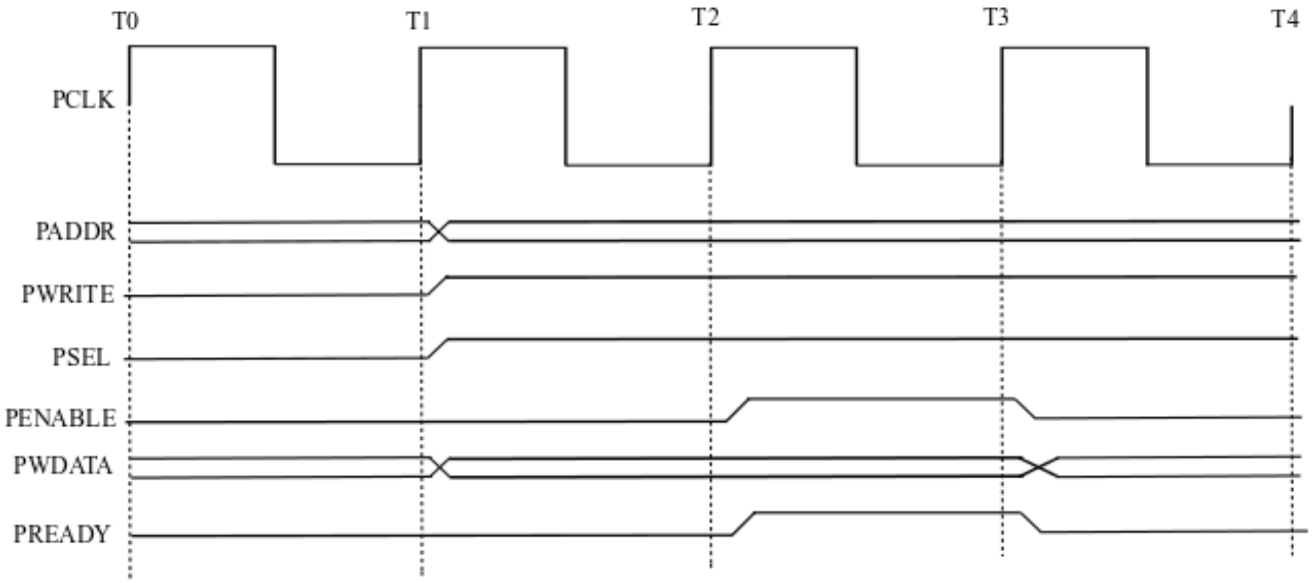

Fig 4: APB Write cycle[2]

2.4 APB Read Cycle

During the read operation, the PSEL, PENABLE, PWRITE, PADDR signals are asserted at the clock edge T1 (SETUP cycle). At the clock edge T2, (ACCESS cycle), the PENABLE, PREADY are asserted and PRDATA is also read during this phase.

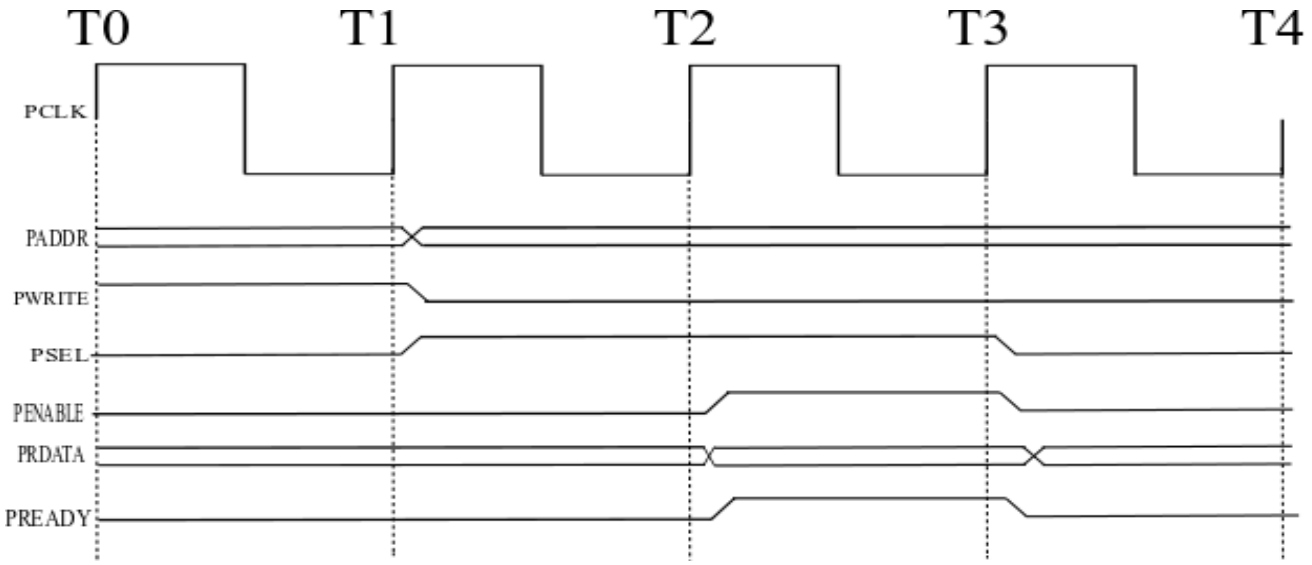

Fig 5: APB Read Cycle[2]

3. SIMULATION RESULTS FOR THE DESIGN

The design and the testbench written in verilog[6] has been compiled using ICARUS verilog. The results show the write and read operation. It is evident from the figure that when PWRITE=1, the address and data are written at the same clock edge. When PWRITE =0, the address is sent on a given clock edge and the data is read on the following clock edge.

```
0 PSEL=0, PENABLE=0, PWRITE=0, PADDR=00000000, PWDATA=00000000, PRDATA=xxxxxxxx
5 PSEL=0, PENABLE=0, PWRITE=0, PADDR=00000000, PWDATA=00000000, PRDATA=00000000
11 PSEL=1, PENABLE=1, PWRITE=1, PADDR=00000001, PWDATA=cccccccc, PRDATA=00000000
31 PSEL=1, PENABLE=1, PWRITE=0, PADDR=00000001, PWDATA=cccccccc, PRDATA=00000000
33 PSEL=1, PENABLE=1, PWRITE=0, PADDR=00000001, PWDATA=cccccccc, PRDATA=cccccccc
51 PSEL=1, PENABLE=1, PWRITE=1, PADDR=00000010, PWDATA=00001111, PRDATA=xxxxxxxx
71 PSEL=1, PENABLE=1, PWRITE=0, PADDR=00000010, PWDATA=00001111, PRDATA=xxxxxxxx
73 PSEL=1, PENABLE=1, PWRITE=0, PADDR=00000010, PWDATA=00001111, PRDATA=00001111
91 PSEL=1, PENABLE=1, PWRITE=1, PADDR=00000011, PWDATA=10101010, PRDATA=xxxxxxxx
111 PSEL=1, PENABLE=1, PWRITE=0, PADDR=00000011, PWDATA=10101010, PRDATA=xxxxxxxx
113 PSEL=1, PENABLE=1, PWRITE=0, PADDR=00000011, PWDATA=10101010, PRDATA=10101010
131 PSEL=1, PENABLE=1, PWRITE=1, PADDR=00000002, PWDATA=11110000, PRDATA=xxxxxxxx
151 PSEL=1, PENABLE=1, PWRITE=0, PADDR=00000002, PWDATA=11110000, PRDATA=xxxxxxxx
153 PSEL=1, PENABLE=1, PWRITE=0, PADDR=00000002, PWDATA=11110000, PRDATA=11110000
```

Fig 6: Simultation results for the design

4. VERIFICATION

Verification is the most important part of the VLSI design flow. It aims to find out the bugs in the RTL (Register Transfer Level) design at an early stage so that it does not prove out destructive at the later stage in the design process. Around 70% of the time is consumed in the verification process. So, it is the most time consuming process. Due to the increase in number of transistors in the integrated circuit (IC), reducing feature size and improved design tools, the complexity of the IC has increased. This raises the probability of occurrence of bugs in the design. Hence, the need for the verification of the IC became necessary [7][9].

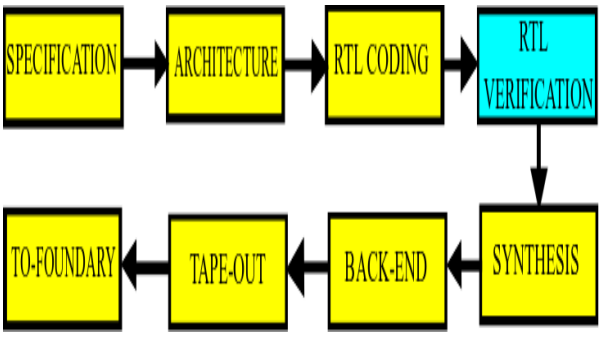

Fig 7: Position of RTL Verification in the VLSI Design Flow

Universal Verification Methodology (UVM) is a standard verification methodology used to verify the RTL (Register Transfer Level) design. It consists of base class library coded in SystemVerilog[8]. The verification engineer can create different verification components by extending these classes. Moreover, UVM provides many other useful verification features such as use of macros for implementing complex function, factory for object creation [8].

Figure 8 shows the various UVM verification components created to verify APB design.

4.1 Sequence item

The transactions are extended from the uvm_sequence_item. This component randomizes the address and data. The field automation macros are applied to the data members of this class.

4.2 Sequences

A sequence is a series of transaction. In the sequence class, the users can create complex stimulus. These sequences can be randomized, extended to create another sequence and can be combined.

4.3 Sequencer

UVM sequencer coordinates between the driver and sequence. It passes the transaction to the driver for execution and obtains the response from the driver. It also acts as an arbitrator for multiple sequences running in parallel.

4.4 Driver

Driver initiates the request for the next transaction and drives it to the lower level components. It is created by extending the uvm_driver.

4.5 Collector and Monitor

The collector extracts the signal information from the bus and converts it into the transactions and passes it through the analysis port to the monitor for further comparing.

4.6 Agent

The agent instantiates the verification components driver, monitor, collector and sequencer. It also connects these components using TLM connections. The agent can have one of the operating modes active or passive. In the active mode of operation, the agent instantiates driver, sequencer collector and monitor whereas in the passive mode of operation only monitor and collector are instantiated and configured.

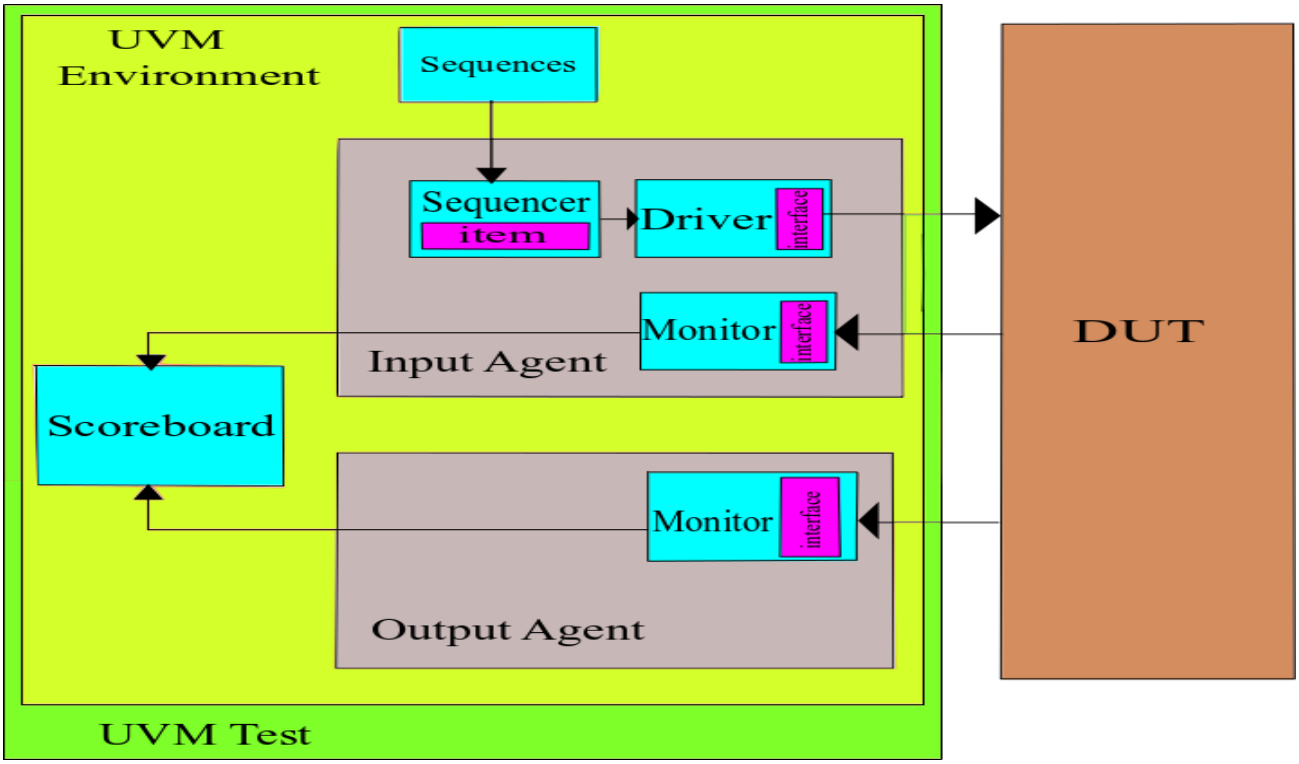

Fig 8: UVM Verification components[10]

4.7 Environment

The Environment class instantiates all the sub components such as agents, driver, monitor etc. and configures them.

4.8 Test

The uvm_test is extended from the uvm_component. Different testcases can be generated for the given verification environment

5. SIMULATION RESULTS OF VERIFICATION

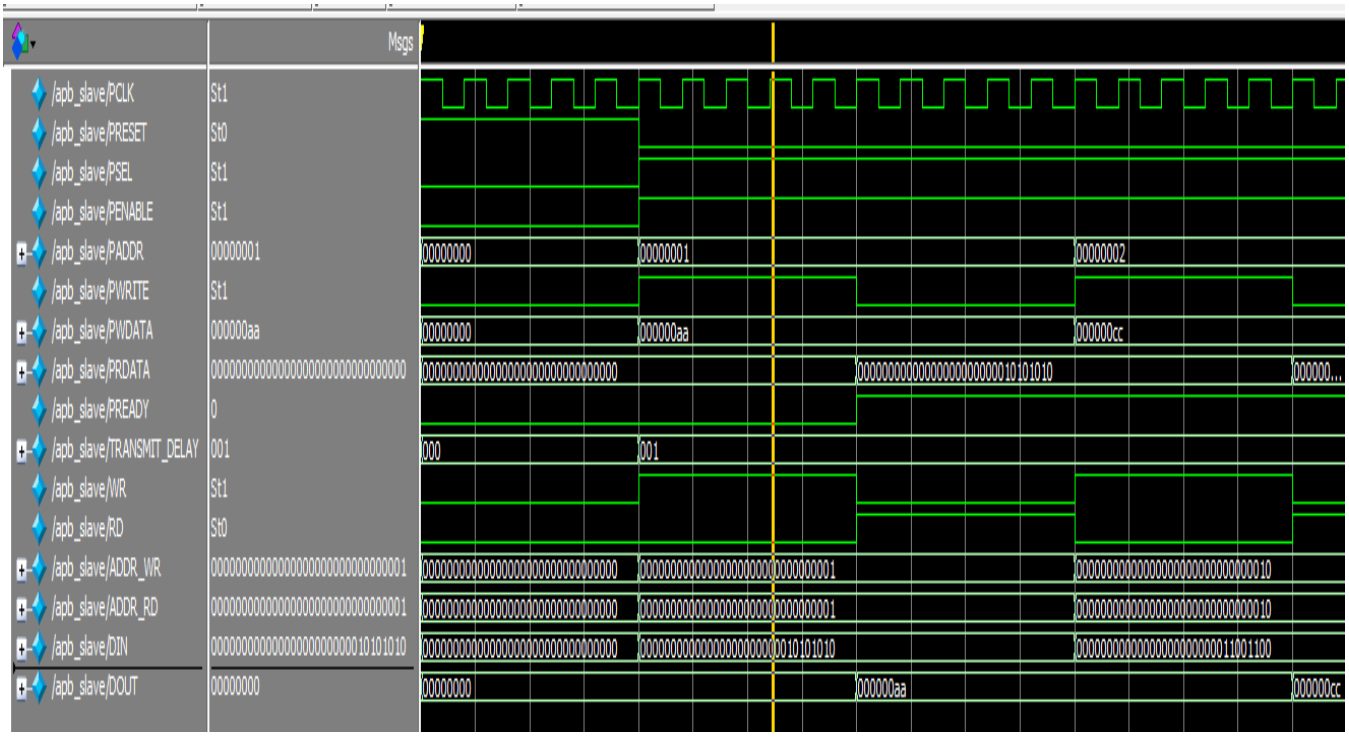

Fig 9: Simulation results from write operation obtained after verification of APB

Figure 9 shows the simulation results obtained by creating verification environment. It is seen that in the simulation results there are some additional signals like DIN, DOUT, ADDR_WR, ADDR_RD. These signals are of the memory which is connected to the APB bus.

In figure 9, the data 000000aa is written to the memory address 00000001 and in figure 10, the same data is read from the same memory location. During write operation in figure.9, the high signal on the PWRITE causes a WR signal of the memory to go high and the data which is applied to PWDATA port is sent to the memory through the DIN port of the memory. The address is applied to the PADDR port which is sent to the ADDR_WR port of the memory.

During the read operation as shown in figure.10, when the PWRITE signal goes low, it causes RD signal of the memory to go high and the address is applied to the PADDR port which is sent to the ADDR_RD port of the memory. The data obtained at the DOUT port of the memory is read at the PRDATA port.

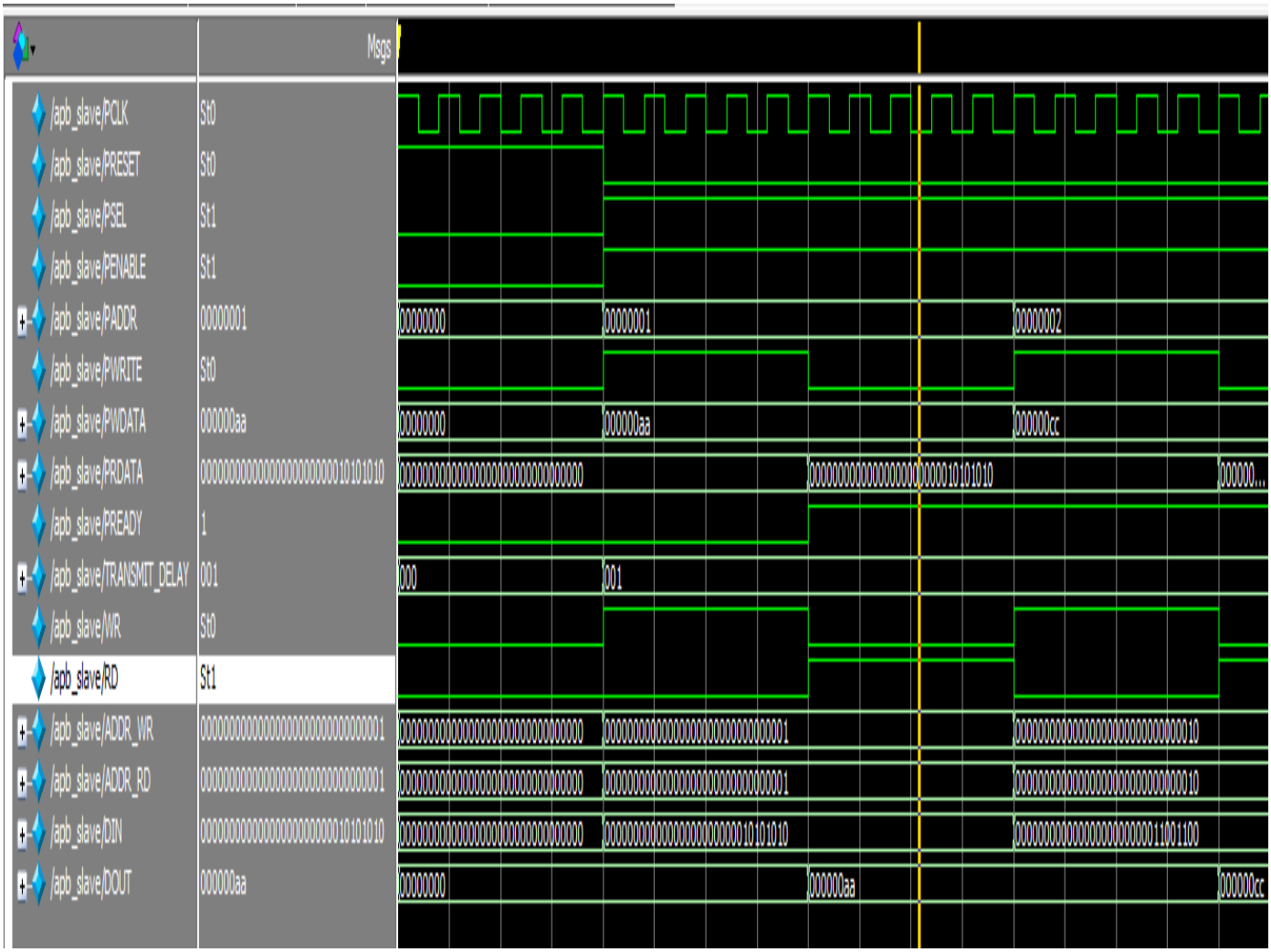

Figure 10: Simulation results for read operation obtained after verification of APB

UVM report provides the results obtained after the simulation of UVM testbench. Figure 11 shows the UVM report summary generated after running all the UVM phases. The UVM_INFO in the UVM report summary in figure 11 shows that there are thirty six information messages. The data provided by the UVM report summary ensures that design is error free and does not produce any warnings or fatal error since the UVM_ERROR, UVM_WARNING and UVM_FATAL is equal to zero.

```
50: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
# UVM_INFO ../../Desktop/apb_vip/apb_vip_20_3_2014/uvm-1.1d/src/base/uvm_phase.svh(1199) @ 180450:
reporter [PH/TRC/EXE/ALLDROP] Phase 'common.run' (id=93) PHASE EXIT ALL_DROPPED
# UVM_INFO ../../Desktop/apb_vip/apb_vip_20_3_2014/uvm-1.1d/src/base/uvm_phase.svh(1215) @ 180450:
reporter [PH_READY_TO_END] Phase 'common.run' (id=93) PHASE READY TO END
# UVM_INFO ../../Desktop/apb_vip/apb_vip_20_3_2014/uvm-1.1d/src/base/uvm_phase.svh(1215) @ 180450:
reporter [PH_READY_TO_END] Phase 'uvm.uvm_sched.post_shutdown' (id=304) PHASE READY TO END
# UVM_INFO ../../Desktop/apb_vip/apb_vip_20_3_2014/testlib.svh(42) @ 180450: uvm_test_top [INFO] Ca
lled my_test::report_phase
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :   36
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [INFO]     6
# [PH/TRC/EXE/ALLDROP]     1
# [PHASESEQ]    13
# [PH_READY_TO_END]    13
# [RNTST]     1
# [TEST_DONE]     2
# ** Note: $finish    : ../../Desktop/apb_vip/apb_vip_20_3_2014/uvm-1.1d/src/base/uvm_root.svh(430)
#    Time: 180450 ns  Iteration: 62  Instance: /tb_top
shankar@nandi:~/Desktop/apb_vip$
```

Fig 11: UVM Report Summary

6. CONCLUSION

This paper gives an overview of the AMBA bus architecture and discusses the APB bus in detail. The APB bus is designed using the verilog HDL according to the specification and is verified using Universal Verification Methodology. The simulation results show that the data read from a particular memory location is same as the data written to the given memory location. Hence, the design is functionally correct. The UVM report summary also ensures the functional correctness of the design.

The electronic system level model of the same design will be created in the future since ESL is the requirement of the future because of increasing design complexity. The ESL model of the APB design will be created using SystemC. Then the design will be verified using UVM testbench. The results obtained after the simulation will be compared with the results obtained in this paper.

7. REFERENCES

[1] ARM, "AMBA Specification Overview", http://www.arm.com/. .

[2] ARM, "AMBA APB3 Specification Overview", http://www.arm.com/

[3] Akhilesh Kumar, Richa Sinha, "Design and Verification analysis of APB3 Protocol with Coverage," IJAET, Nov 2011.

[4] Santhi Priya Sarekokku, K. Rajasekhar, "*Design and Implementation of APB Bridge based on AMBA AXI 4.0*," IJERT, Vol.1, Issue 9, Nov 2012.

[5] *UVM Reference Manual*, http://www.accellera.com

[6] Samir Palnitkar, "*Verilog HDL: A guide to Digital Design and Synthesis (2nd Edition)*, Pearson, 2008.

[7] Chris Spear, "SystemVerilog for verification (2nd Edition): A guide to learning the testbench features, Springer, 2008.

[8] URL:http://www.testbench.com.

[9] Bergeron, "Writing testbenches using SystemVerilog," Springer, 2009.

[10] Vanessa R. Cooper, "*Getting Started with UVM: A Beginner's Guide*," Verilab, 2013.