Verification of ARM AMBA3 APB Protocol Using SystemVerilog

Siemens EDA's (Mentor Graphics) Required Task as a Selection Phase

Submitted By : Mohamed Ahmed Abdelaziz

Submitted To: Eng. Mohamed Mourad

Contents

Overview

1.1. APB (Advanced Peripheral Bus)

APB 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.

APB the bridge acts as the master and all the devices connected on the APB bus acts as the slave. So, the bridge can connect the high performance AHB and ASB busses to APB bus.

In this task I followed the UVM steps to able to implement the APB design for both the slave and the bridge (master) and verify it using the UVM procedures. I referred to many tutorials to understand what is the APB protocol and how it works, also I referred to tutorials to know how to verify the DUT I did passing through the UVM steps, starting from sequence items ended by the top modules and how to write an effective readable SyetemVerilog code as this is my first verification project to implement using SystemVerilog and UVM. I use ModelSim as a simulation tool in this project.

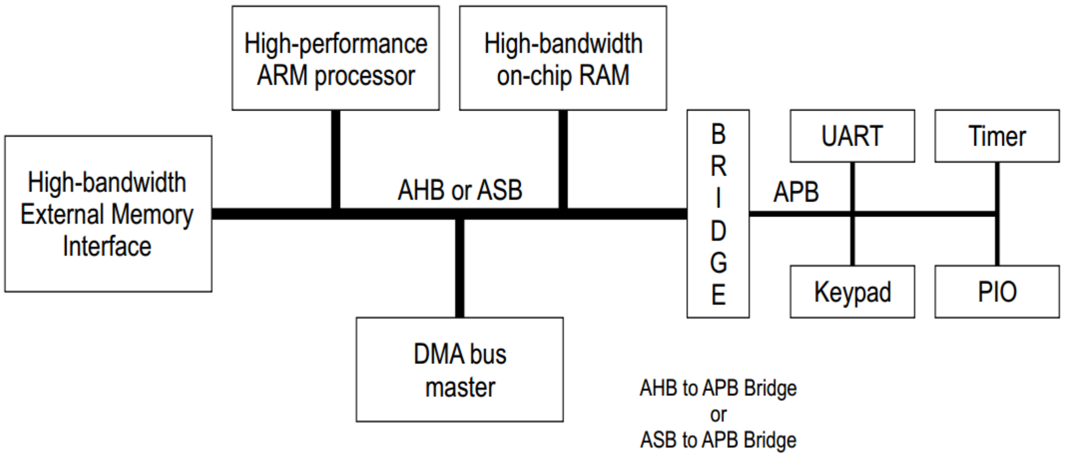

High-performance ARM processor
High-bandwidth on-chip RAM
High-bandwidth External Memory Interface
AHB or ASB
DMA bus master
B R I D G E
APB
UART
Timer
Keypad
PIO
AHB to APB Bridge or ASB to APB Bridge

Figure 1 AMBA Bus including AHB/ASB and APB ©allaboutcircuits.com

1.2. APB Block Diagram

AMBA-APB consists of Bridge and Slave. APB bridge is a bus master on AMBA-APB, it also considered as a slave on the high-level system bus.
Here are the basic block diagrams for both APB bridge and APB slave:

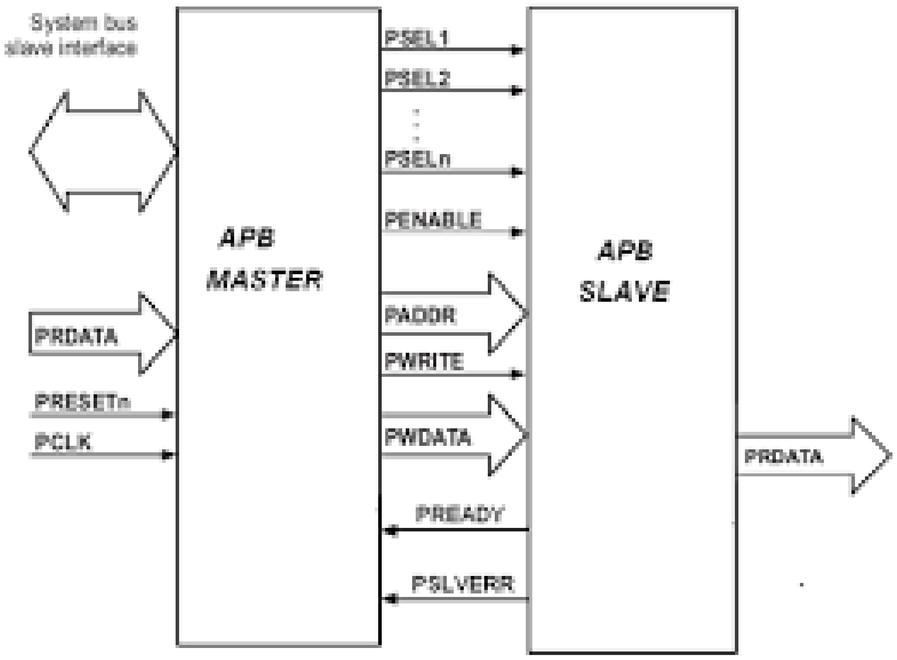


Figure 2 Basic block diagram of APB Bridge - Slave Interface

1.3. APB Operating States

Referring to the AMBA3_APB standard and studying its FSM diagram shown below, there are three main states:
IDLE, SETUP and ACESS.

- **IDLE State**: IDLE state is the default state in which no operation is being performed.
- **SETUP State**: The assertion of the PSEL signal indicates the beginning of the SETUP state. When the data transfer is required then the bus enters the SETUP state.
  Also, the PWRITE, PADDR and PWDATA are provided during this state.
- **ACCESS State**: After Remaining in the SETUP state for one CLK cycle, the bus will move to the ACCESS state on the next rising edge of the CLK after the SETUP state CLK cycle. When PENABLE signal is high this indicates the start of the ACCESS state. All the control signals, address, and the data signals remain stable during the transition from the SETUP state to the ACCESS state.
  In case of read operation the PRDATA is present on the bus during this state.
  PENABLE signal also remain high for one clock cycle.
  If there is no data transfer is required, the bus will move the IDLE state.
  If there is further data transfer is required, then the bus will move to the SETUP state.

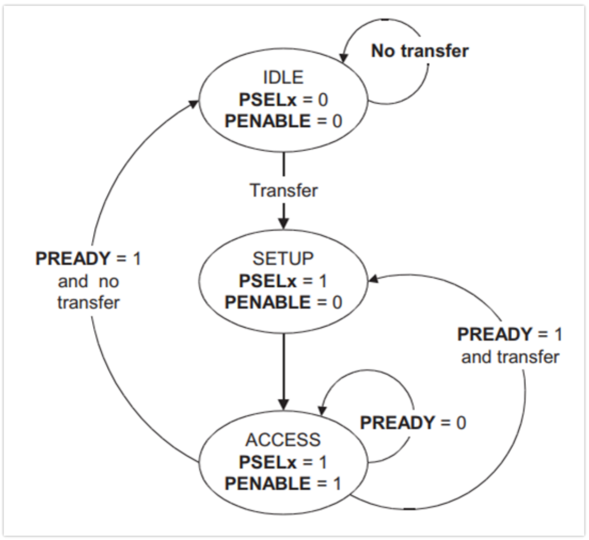


Figure 3 APB State Diagram

1.4. Write Transfers (with no wait states)

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

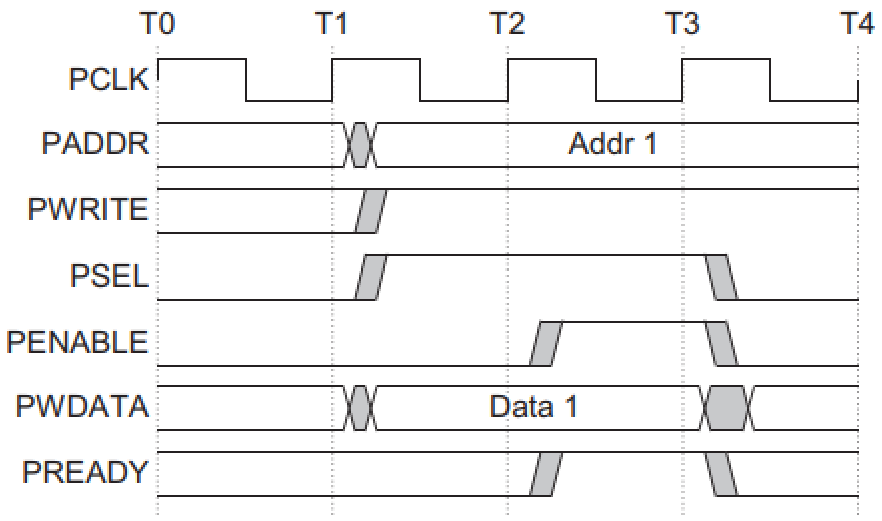


Figure 4 Write Transfers with no wait states

1.5. Read Transfers (with no wait states)

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

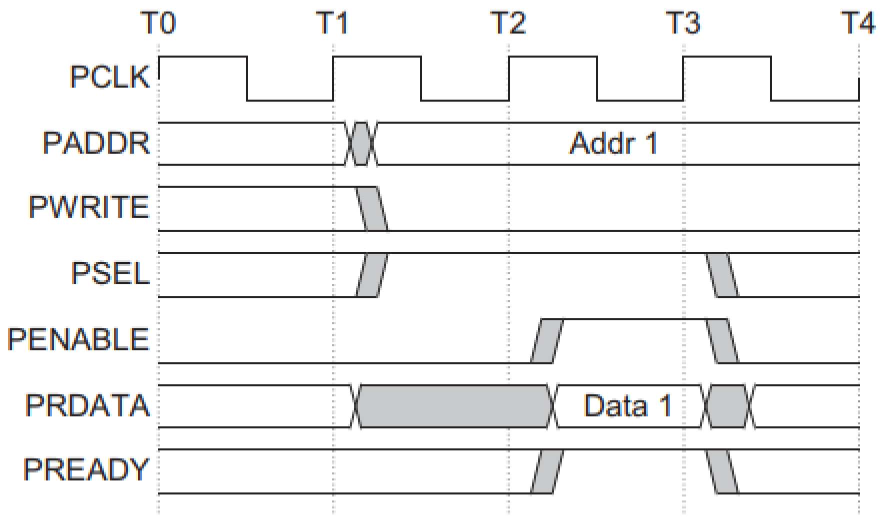


Figure 5 Write Transfers with no wait states

APB Protocol Verification

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.

2.1. Sequence Item

- The fields required to generate the stimulus are declared in the sequence item, it can be used as a placeholder for the activity monitored by the monitor on the DUT (design under test) signals.
- Procedure I followed to write the sequence Items:
  - sequence_item is written by extending uvm_seq_item
  - Declaring the fields in apb_seq_item

```
bit [7:0] PRDATA; // data sent by slave
rand bit [20:0] sysbus; // rand is used
to generate a random stimulus
```

  - In order to use the uvm_object methods, all the fields are registered to uvm_field_* macros

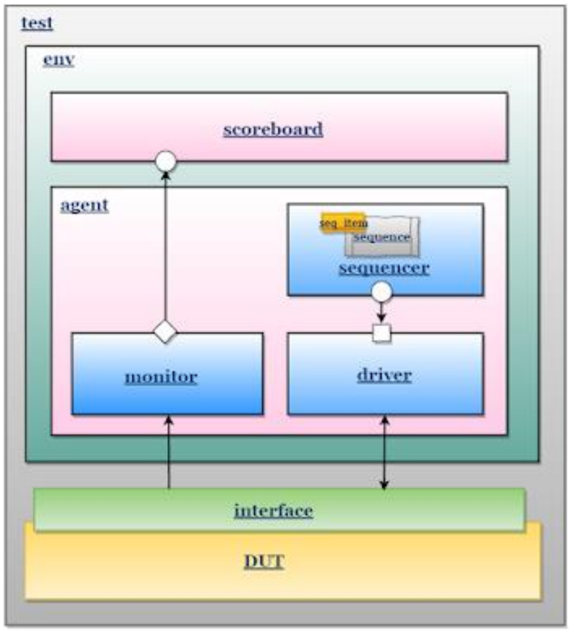


Figure 6 UVM TestBench architecture ©Verification Guide

2.2. Sequence

- Sequence generates the stimulus and sends to the driver via sequencer

2.3. Sequencer

- Sequencer is written by extending uvm_sequencer, there is no extra logic required to be added in the sequencer

2.4. Driver

- Driver receives the stimulus from sequence via sequencer and drives on interface signals
  - apb_driver is written by extending the uvm_driver
  - Declare the virtual interface : virtual apb_intf intf;
  - Get the interface handle using get config_db & adding the get config_db in the build_phase

```
     function void build_phase(uvm_phase phase);
        super.build_phase(phase);
      if(!uvm_config_db#(virtual
apb_intf)::get(this,"","intf",intf))`uvm_fatal("NO_VIF",{"virtual
interface must be set for: ",get_full_name(),".vif"});
     endfunction : build_phase
```

- Add driving logic. get the seq_item and drive to DUT signals

```
// drive phase
virtual task drive();
req.print();

@(posedge intf.DRIVER.PCLK);
`driv_if.sysbus <= req.sysbus;
@(posedge intf.DRIVER.PCLK);
req.PRDATA= `driv_if.PRDATA;
endtask : drive
```

2.5. Monitor

Monitor samples the DUT signals through the virtual interface and converts the signal level activity to the transaction level. Here we can Add Sampling logic in run_phase

- Sample the interface signal and assign to trans_collected handle
- Sampling logic is placed in the forever loop

```
virtual task run_phase(uvm_phase phase);
forever begin
 @(posedge intf.MONITOR.PCLK);
 trans_collected.sysbus=intf.monitor_cb.sysbus;
 @(posedge intf.MONITOR.PCLK);
 trans_collected.PRDATA=intf.monitor_cb.PRDATA;
 item_collected_port.write(trans_collected);
 end
endtask: run_phase
```

- After sampling, by using the write method send the sampled transaction packet to the scoreboard

```
item_collected_port.write(trans_collected);
```

2.6. Agent

An agent is a container class contains a driver, a sequencer, and a monitor. Depending on Agent type, create agent components in the build phase, take in considerations that the driver and the sequencer will be created only for the active agent.

2.7. Scoreboard

scoreboard receives the transaction from the monitor and compares it with the reference values.

2.8. Environment

The environment is the container class, It contains one or more agents, as well as other components such as the scoreboard, top-level monitor, and checker.

2.9. Test

The test defines the test scenario for the testbench.

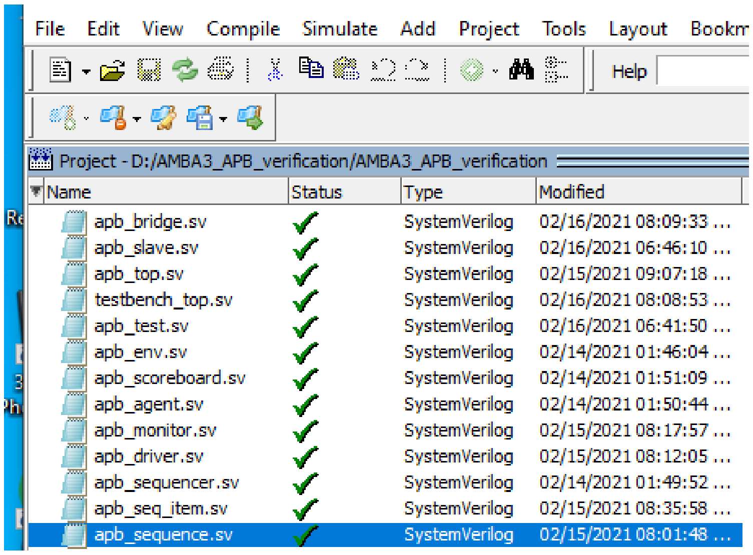
2.10. TestBench_Top

TestBench top is the module, it connects the DUT and Verification environment components.

Simulation Results

3.1. Compilation Process Results

As Shown in the opposite figure (7) that compilation Process is done successfully.


File Edit View Compile Simulate Add Project Tools Layout Bookm
Help

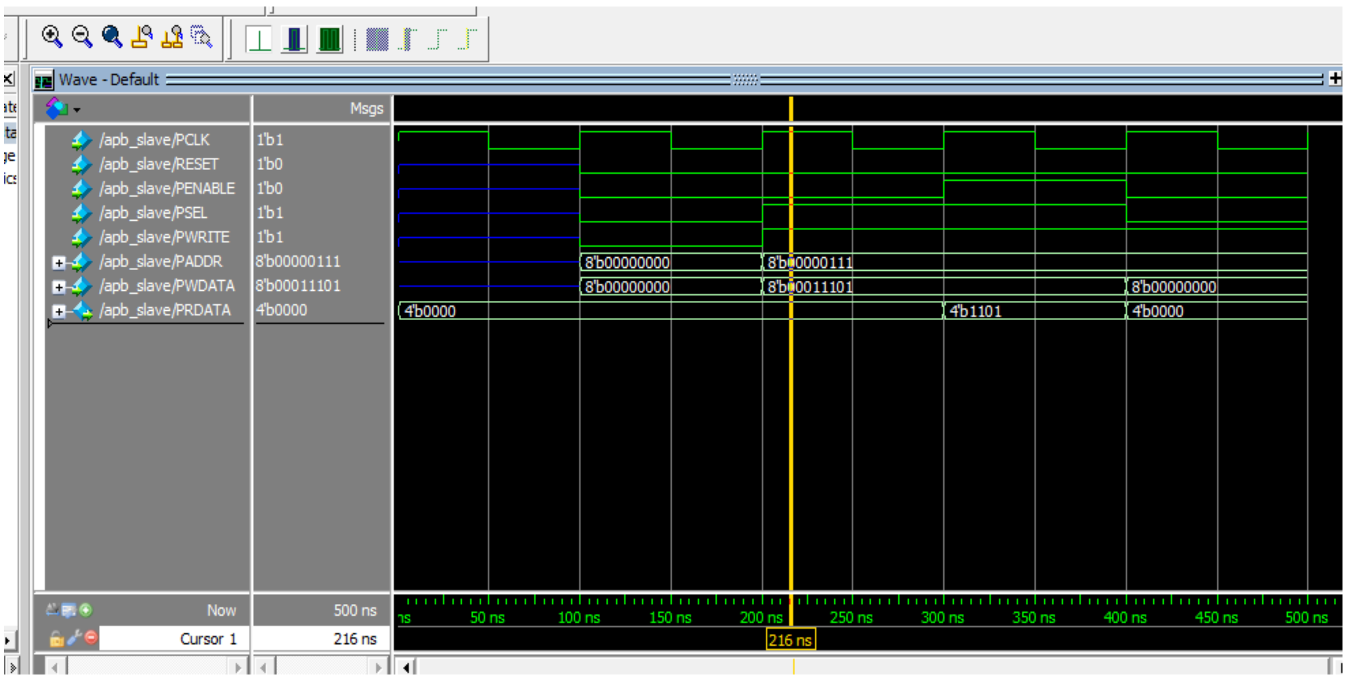
Project - D:/AMBA3_APB_verification/AMBA3_APB_verification

Name	Status	Type	Modified
apb_bridge.sv	✓	SystemVerilog	02/16/2021 08:09:33 ...
apb_slave.sv	✓	SystemVerilog	02/16/2021 06:46:10 ...
apb_top.sv	✓	SystemVerilog	02/15/2021 09:07:18 ...
testbench_top.sv	✓	SystemVerilog	02/16/2021 08:08:53 ...
apb_test.sv	✓	SystemVerilog	02/16/2021 06:41:50 ...
apb_env.sv	✓	SystemVerilog	02/14/2021 01:46:04 ...
apb_scoreboard.sv	✓	SystemVerilog	02/14/2021 01:51:09 ...
apb_agent.sv	✓	SystemVerilog	02/14/2021 01:50:44 ...
apb_monitor.sv	✓	SystemVerilog	02/15/2021 08:17:57 ...
apb_driver.sv	✓	SystemVerilog	02/15/2021 08:12:05 ...
apb_sequencer.sv	✓	SystemVerilog	02/14/2021 01:49:52 ...
apb_seq_item.sv	✓	SystemVerilog	02/15/2021 08:35:58 ...
apb_sequence.sv	✓	SystemVerilog	02/15/2021 08:01:48 ...

Figure 7 Compilation Results # 13 compiles, 0 failed with no errors

3.2. Slave Simulation Result

The wave shows the write operator from the APB Slave, at First all the control signals are in the idle state, As shown in figure(8), PWDATA = 8’b00011101 is written to the memory address PADDR = 8’b00000111.

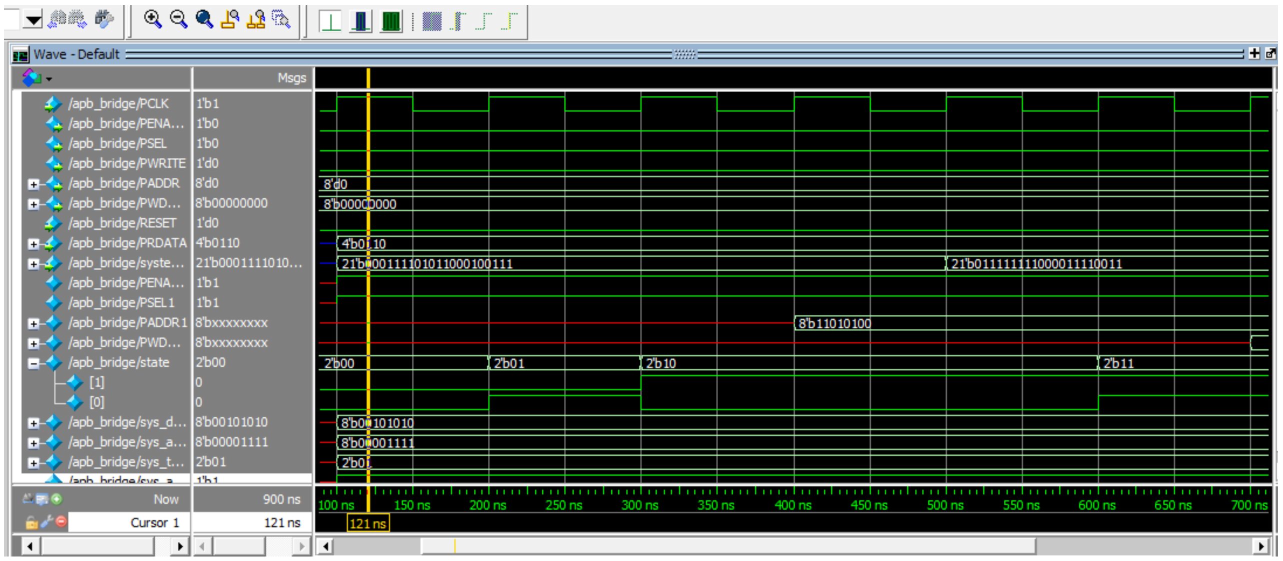


	Msgs
/apb_slave/PCLK	1'b1
/apb_slave/RESET	1'b0
/apb_slave/PENABLE	1'b0
/apb_slave/PSEL	1'b1
/apb_slave/PWRITE	1'b1
/apb_slave/PADDR	8'b00000111
/apb_slave/PWDATA	8'b00011101
/apb_slave/PRDATA	4'b0000
Now	500 ns
Cursor 1	216 ns

Figure 8 generated output result from APB_Slave Write operator

3.3. Master Simulation Result

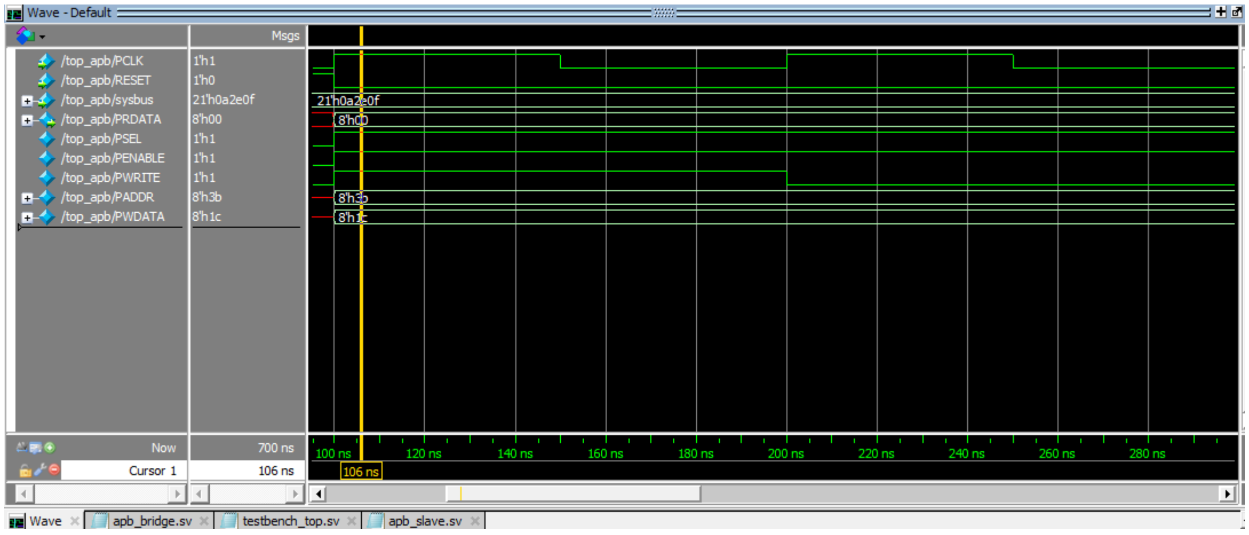
The Implemented code has some issues in the write operation that it consumes too much time to debug it.



```
Wave - Default
                          Msgs
/apb_bridge/PCLK          1'b1
/apb_bridge/PENA...       1'b0
/apb_bridge/PSEL          1'b0
/apb_bridge/PWRITE        1'd0
/apb_bridge/PADDR         8'd0
/apb_bridge/PWD...        8'b00000000
/apb_bridge/RESET         1'd0
/apb_bridge/PRDATA        4'b0110
/apb_bridge/syste...      21'b0001111010...
/apb_bridge/PENA...       1'b1
/apb_bridge/PSEL1         1'b1
/apb_bridge/PADDR1        8'bxxxxxxxx
/apb_bridge/PWD...        8'bxxxxxxxx
/apb_bridge/state         2'b00
  [1]                     0
  [0]                     0
/apb_bridge/sys_d...      8'b00101010
/apb_bridge/sys_a...      8'b00001111
/apb_bridge/sys_t...      2'b01
Now                       900 ns
Cursor 1                  121 ns
```

Figure 9 Simulation Result for APB_Bridge Read Operation

3.4. Testbench Simulation Result



```
Wave - Default
                          Msgs
/top_apb/PCLK             1'h1
/top_apb/RESET            1'h0
/top_apb/sysbus           21'h0a2e0f
/top_apb/PRDATA           8'h00
/top_apb/PSEL             1'h1
/top_apb/PENABLE          1'h1
/top_apb/PWRITE           1'h1
/top_apb/PADDR            8'h3b
/top_apb/PWDATA           8'h1c
Now                       700 ns
Cursor 1                  106 ns
```

Figure 10 Tb simulation result

- I try to extract the Test Summary but It returns an error and I can't even trace or debug it. The same issue occurs also when I try to extract a test summary for each module.

References

- AMBA3 APB Protocol – ARM Standard
- https://verificationguide.com/uvm-topics/
- VLSI Kingdom – youtube Channel