VIT-PROJECT SUBMISSION

PROJECT-REPORT

PROJECT TITLE:

AHB2APB Bridge Design

SUBMITTED FROM:

BANDI PRANEESH REDDY

praneesh.20bec7038@vitap.ac.in

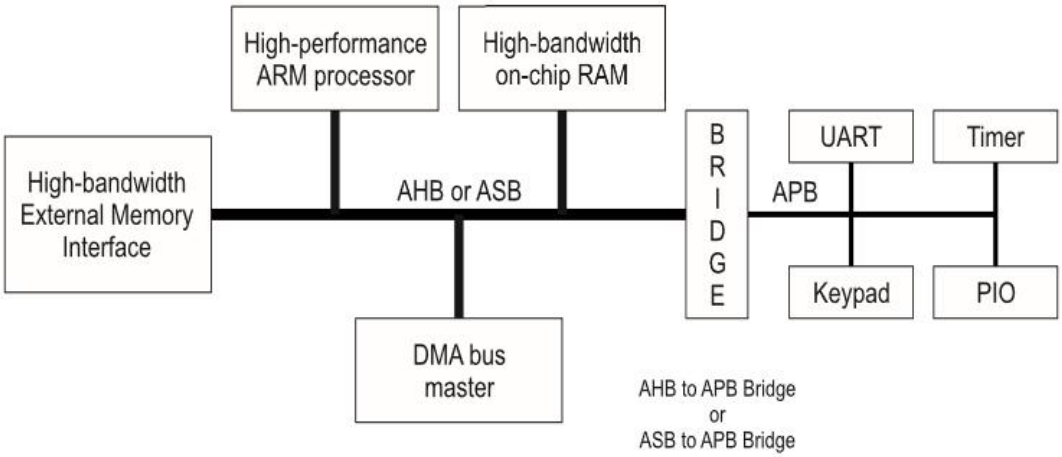
SUBMITTED TO:

Manjunath N L(SIR)

-MAVEN SILICON

VIT-AP UNIVERSITY

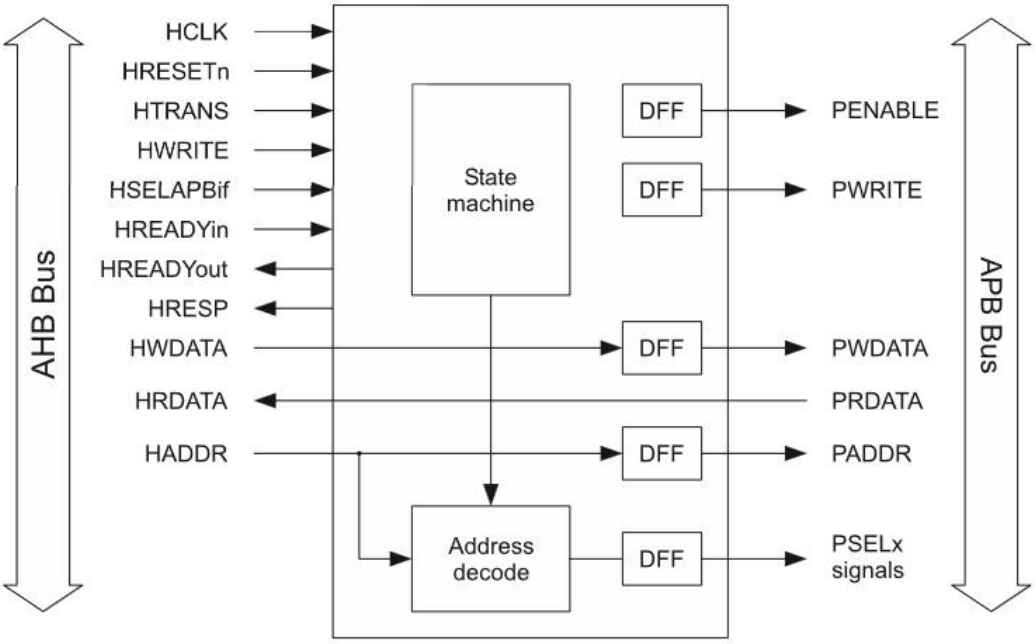
TOP MODULE BLOCK DIAGRAM (overall functionality):



AHB to APB Bridge:

- The AHB to APB bridge is an AHB slave, providing an interface between the high speed AHB and the low-power APB.
- Read and write transfers on the AHB are converted into equivalent transfers on the APB.
- As the APB is not pipelined, then wait states are added during transfers to and from the APB where the AHB is required to wait for the APB.

Block Diagram of AHB to APB Bridge:

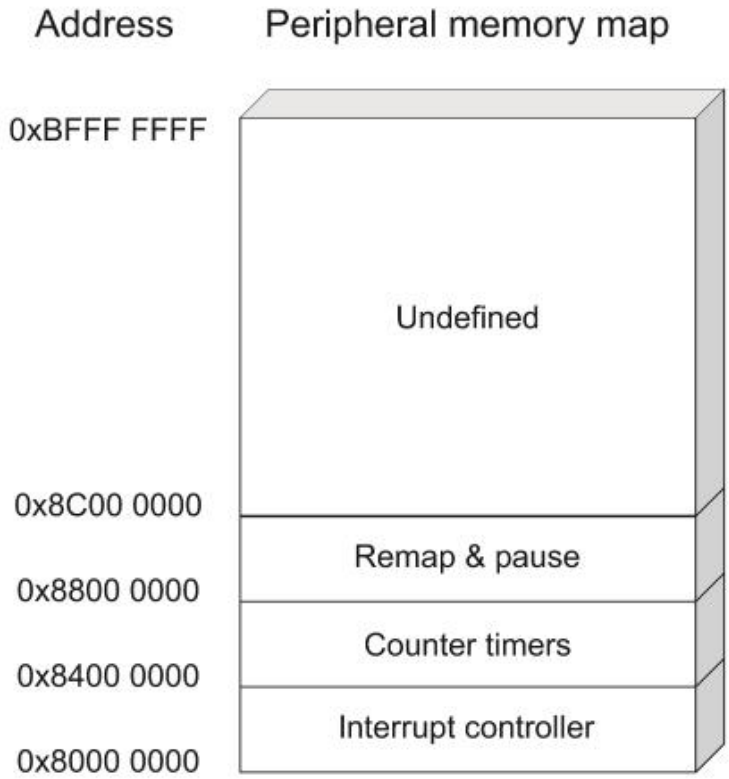


Bridge Description (Functionality):

The bridge unit converts system bus transfers into APB transfers and performs the following functions:

- Latches the address and holds it valid throughout the transfer.
- Decodes the address and generates a peripheral select, **PSELX.** Only one select signal can be active during a transfer.
- Drives the data onto the APB for a write transfer.
- Drives the APB data onto the system bus for a read transfer.
- Generates a timing strobe, **PENABLE,** for the transfer.

Peripheral memory map:



The AHB to APB bridge is formed by interconnecting the following six blocks:

1. `top_tb`: This is the top-level testbench module that instantiates the other modules and defines the signals and connections between them.

2. `ahb_master`: This module represents the AHB master interface. It contains tasks and logic to perform various AHB transactions like single write, single read, burst write, and burst read. It controls the signals related to AHB transactions, such as `hclk`, `hresetn`, `hr_readyout`, `hr_data`, `haddr`, `hwdata`, `hwrite`, `hready_in`, and `htrans`.

3. `ahb_slave_interface`: This module represents the AHB slave interface. It contains logic to handle AHB address decoding and data transfer between the AHB and APB interfaces. It controls signals such as `hclk`, `hresetn`, `hwrite`, `hready_in`, `htrans`, `hwdata`, `haddr`, `pr_data`, `hr_data`, `temp_sel`, and `valid`.

4. `apb_controller`: This module acts as a controller between the AHB and APB interfaces. It determines the next state based on the current state and input signals, and generates temporary output signals accordingly. It controls signals such as `hclk`, `hresetn`, `hwrite_reg`, `hwrite_reg1`, `hwrite`, `valid`, `haddr`, `hwdata`, `hwdata1`, `hwdata2`, `haddr1`, `haddr2`, `pr_data`, `temp_selx`, `penable_temp`, `pwrite_temp`, `hr_readyout_temp`, `psel_temp`, `paddr_temp`, and `pwdata_temp`.

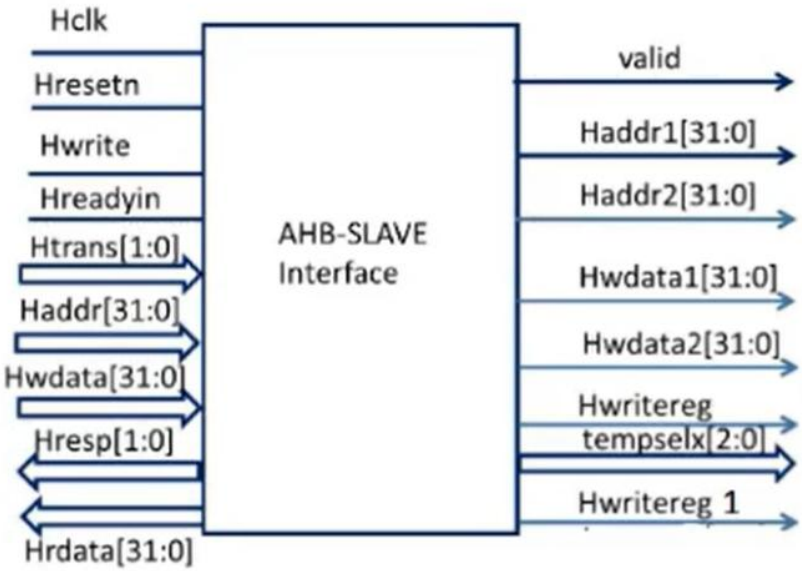
5. `apb_interface`: This module represents the APB interface. It assigns the input signals `pwrite`, `penable`, `psel`, `paddr`, and `pwdata` to the output signals `pwrite_out`, `penable_out`, `psel_out`, `paddr_out`, and `pwdata_out` respectively. It also generates the output signal `pr_data` based on the input signals `pwrite` and `penable`.

6. `bridge_top`: This module connects the signals and interfaces between the AHB and APB interfaces. It connects the signals and interfaces of the AHB master, AHB slave interface, and APB interface. It controls signals such as `hclk`, `hresetn`, `hwrite`, `hready_in`, `htrans`, `hwdata`, `haddr`, `pr_data`, `penable`, `pwrite`, `hr_readyout`, `psel`, `hres`, `paddr`, and `pwdata`.

These six modules work together to establish the AHB to APB bridge functionality, allowing communication between the AHB master and APB slave devices.

Sub- Blocks Block Diagram (Functionality):

BLOCK-1: AHB_SLAVE Interface:

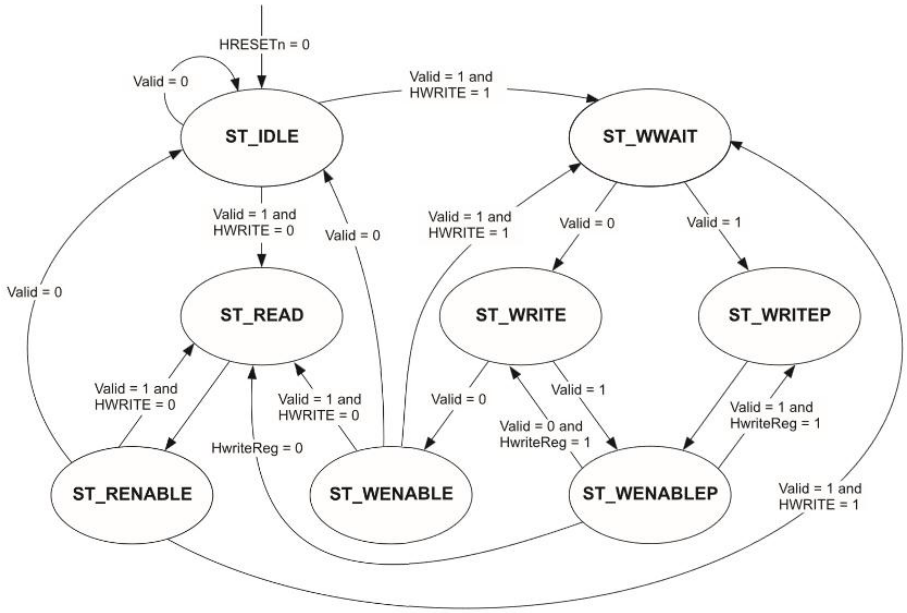


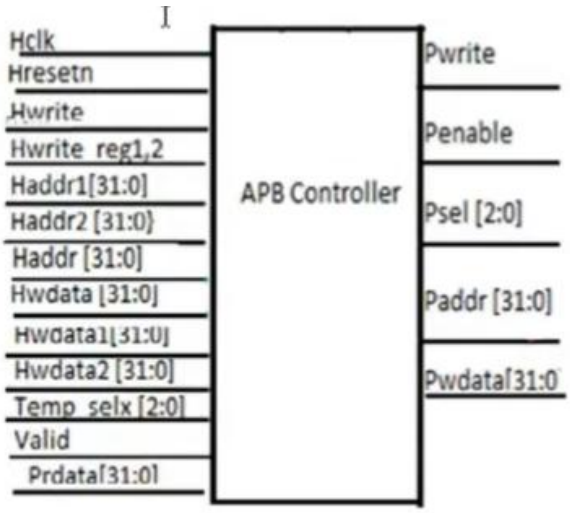
Functionality of the block:

We have to write the pipeline logic of haddr, hwdata, and hwrite and generate the valid and temp_selx.

The AHB slave interface module (**ahb_slave_interface**) represents the behavior of the target device connected to the AHB bus. It receives the AHB bus signals (**hclk**, **hresetn**, **hwrite**, **hready_in**, **htrans**, **hwdata**, **haddr**, **pr_data**) and generates the necessary response signals (**hr_data**, **hresp**, etc.) based on the transactions requested by the AHB master.

BLOCK-2: APB_CONTROLLER:





Functionality of the block:

The APB controller module (**apb_controller**) controls the APB interface module based on the transactions initiated by the AHB master. It includes a state machine that transitions between different states (**ST_IDLE**, **ST_WWAIT**, **ST_READ**, etc.) based on the current transaction type and the validity of the transactions. The APB controller generates temporary output signals (**paddr_temp**, **pwdata_temp**, **pwrite_temp**, **psel_temp**, **penable_temp**, **hr_readyout_temp**) based on the current state, which are then synchronized to the clock (**hclk**) to generate the actual output signals (**paddr**, **pwdata**, **pwrite**, **psel**, **penable**, **hr_readyout**).

Output logic discussion:

1. As per the protocol we have to register the output. Registering output will create 1 clock delay.

2. Even after delaying the output waveforms must match protocol waveforms (AHB2APB output waveforms).

3. Temporary variables for all the output of the apb controller

Paddr_temp. pwdata_temp. psel_temp, penable_temp, hr_readyout_temp

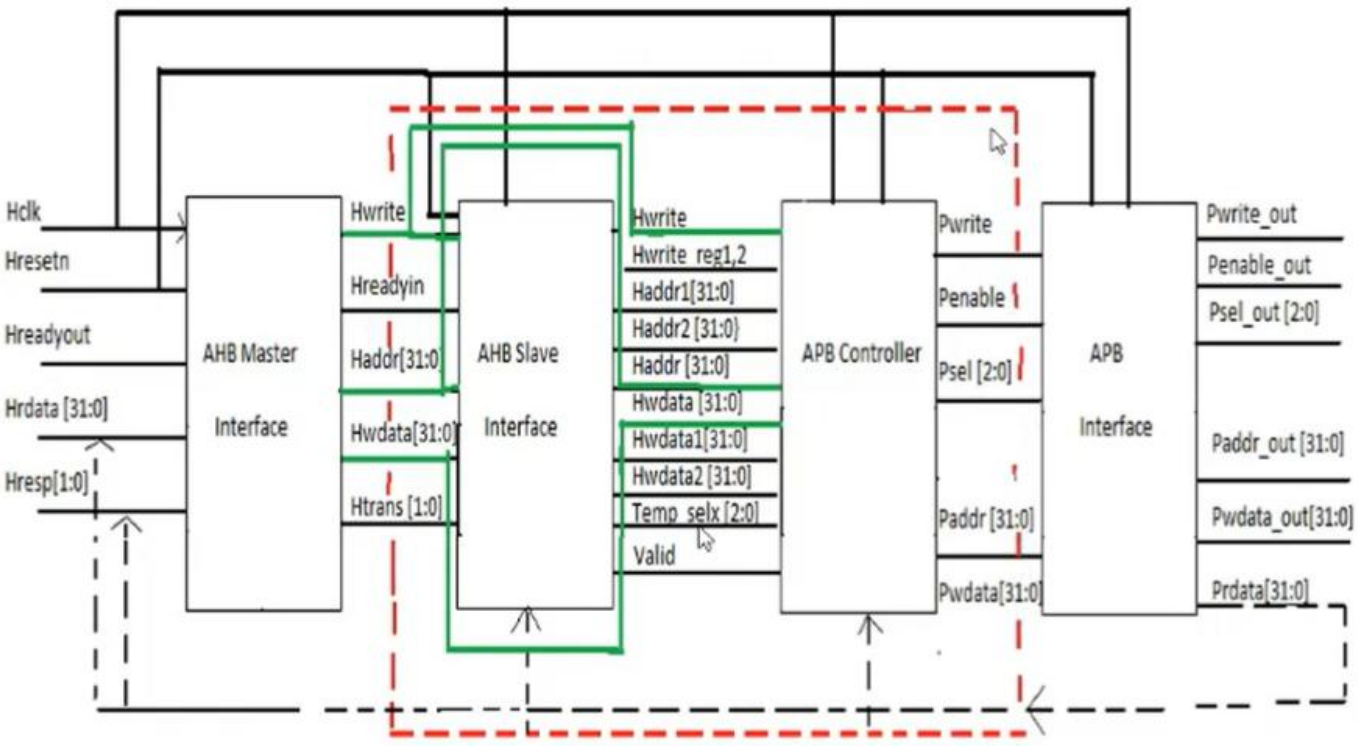
4. Always block for 1. Temporary output logic (comb) 2. output logic (sequential)

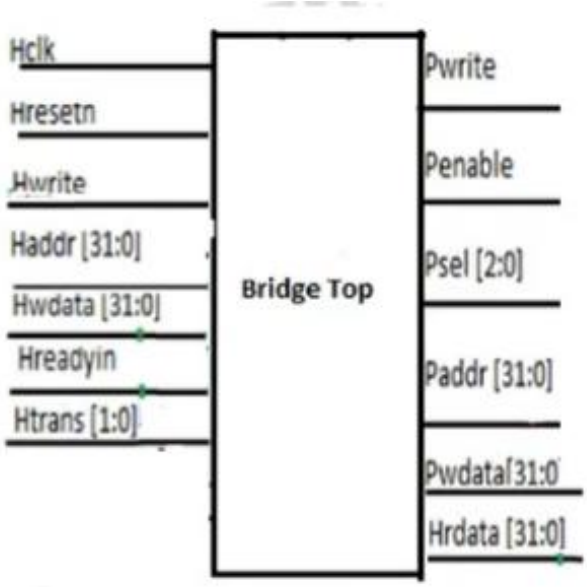
5. Temporary output logic written w.rt waveform in such a way that even after registering output, it has to come has waveform protocol.

6 Output logic block temporary variables are assigned to output variables on posedge of hclk.

==========

BLOCK-3: BRIDGE TOP:





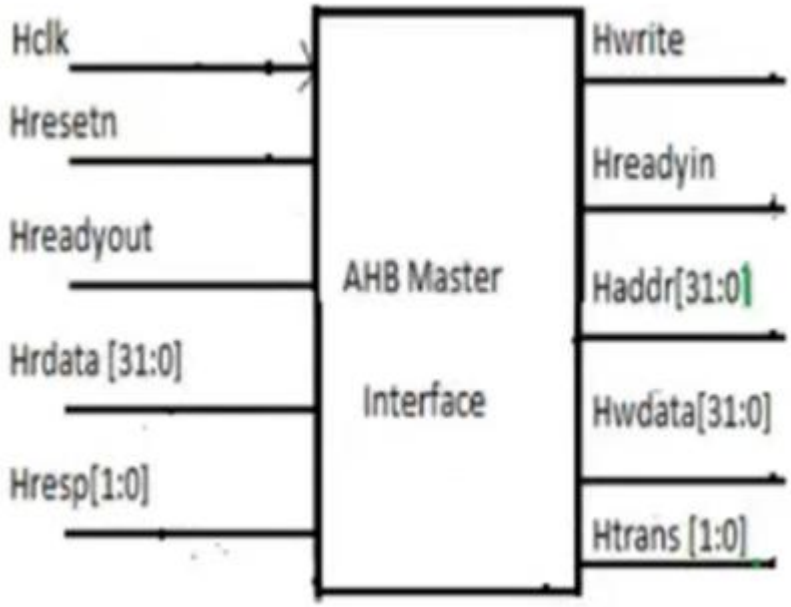
Bridge Top is a module in the given code that serves as an interface between the AHB (Advanced High-Performance Bus) master module and the APB (Advanced Peripheral Bus) interface module. It facilitates communication between the AHB and APB protocols by translating the signals and data between the two bus protocols.

Functionality of the block:

- The Bridge Top module receives signals from the AHB master and translates them into corresponding signals for the APB interface.
- It connects the AHB signals (**hwrite, hready_in, htrans, hwdata, haddr**) to the corresponding APB signals (**pwrite, penable, psel, paddr, pwdata**).
- It also connects the APB read data (**pr_data**) to the corresponding AHB signal (**hr_data**).
- The module handles the translation of bus protocols, ensuring compatibility between AHB and APB interfaces.
- It manages the flow of data and control signals between the AHB master and APB interface, enabling communication between the two modules.

Overall, the Bridge Top module acts as a bridge or intermediary between the AHB master and APB interface, facilitating seamless data transfer and communication between the two bus protocols.

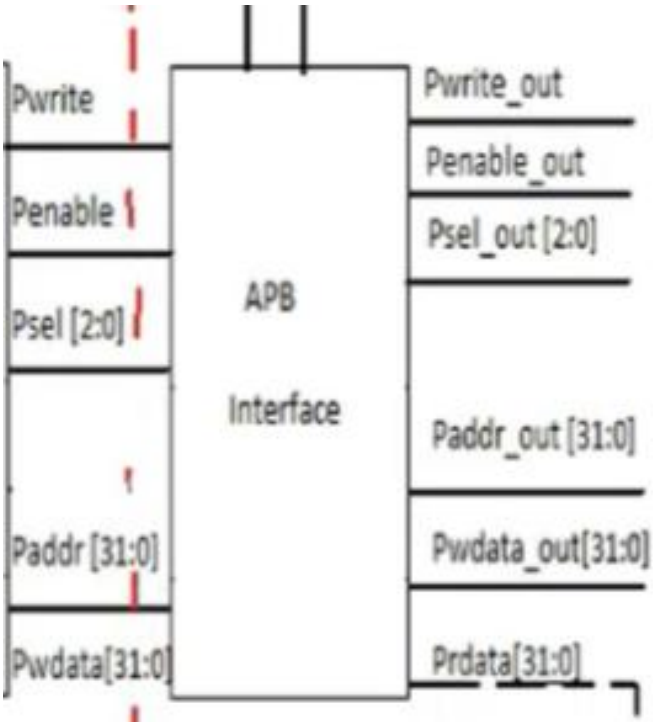
BLOCK-4: AHB_MASTER:



Functionality of the block:

The AHB master module (**ahb_master**) contains tasks for performing various operations such as single write, single read, burst write, and burst read. The operations are initiated based on the values of the input signals (**hwrite**, **valid**, etc.). The module generates the appropriate control signals (**htrans**, **hsize**, **hburst**, etc.) and the address and data signals (**haddr**, **hwdata**) for communicating with the AHB slave interface.

BLOCK-5: APB_INTERFACE:



Functionality of the block:

The APB interface module (**apb_interface**) is responsible for converting the AHB signals (**pwrite**, **penable**, **psel**, **paddr**, **pwdata**) into APB signals (**pwrite_out**, **penable_out**, **psel_out**, **paddr_out**, **pwdata_out**). It also generates the read data (**pr_data**) by randomly assigning a value when a read request is detected (**pwrite == 0** and **penable == 1**).

==========

Finally, Testbench for top_tb():

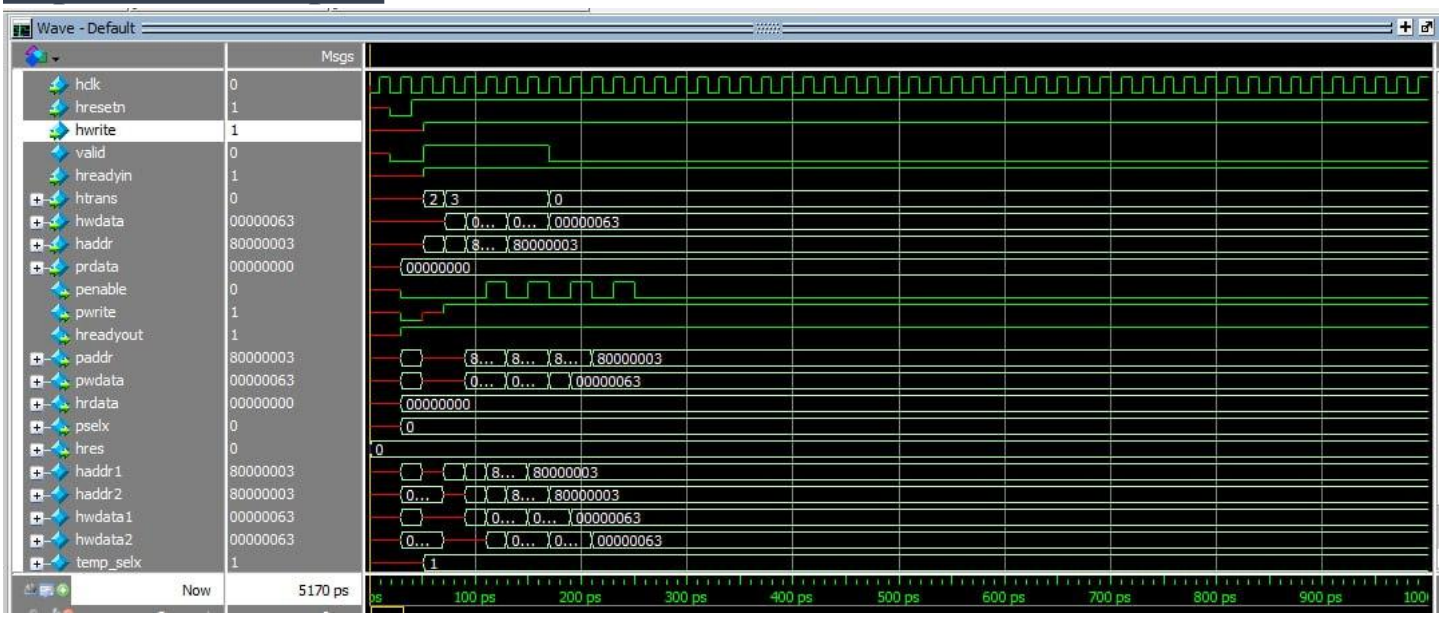
The top-level testbench (**top_tb**) instantiates the AHB master, AHB slave interface, APB interface, and APB controller modules. It includes an initial block that initializes the testbench and calls the desired tasks of the AHB master module (**single_write()**, **single_read()**, **burst_4_incr_write()**) after a reset. The testbench is then simulated for a certain duration before finishing (**$finish**).

The output waveforms are:

1) A1.single_write();
2) A1.single_read();
3) A1.burst_write_incr();
4) A1.burst_write_wrap();
5) A1.burst_read_incr();
6) A1.burst_read_wrap();

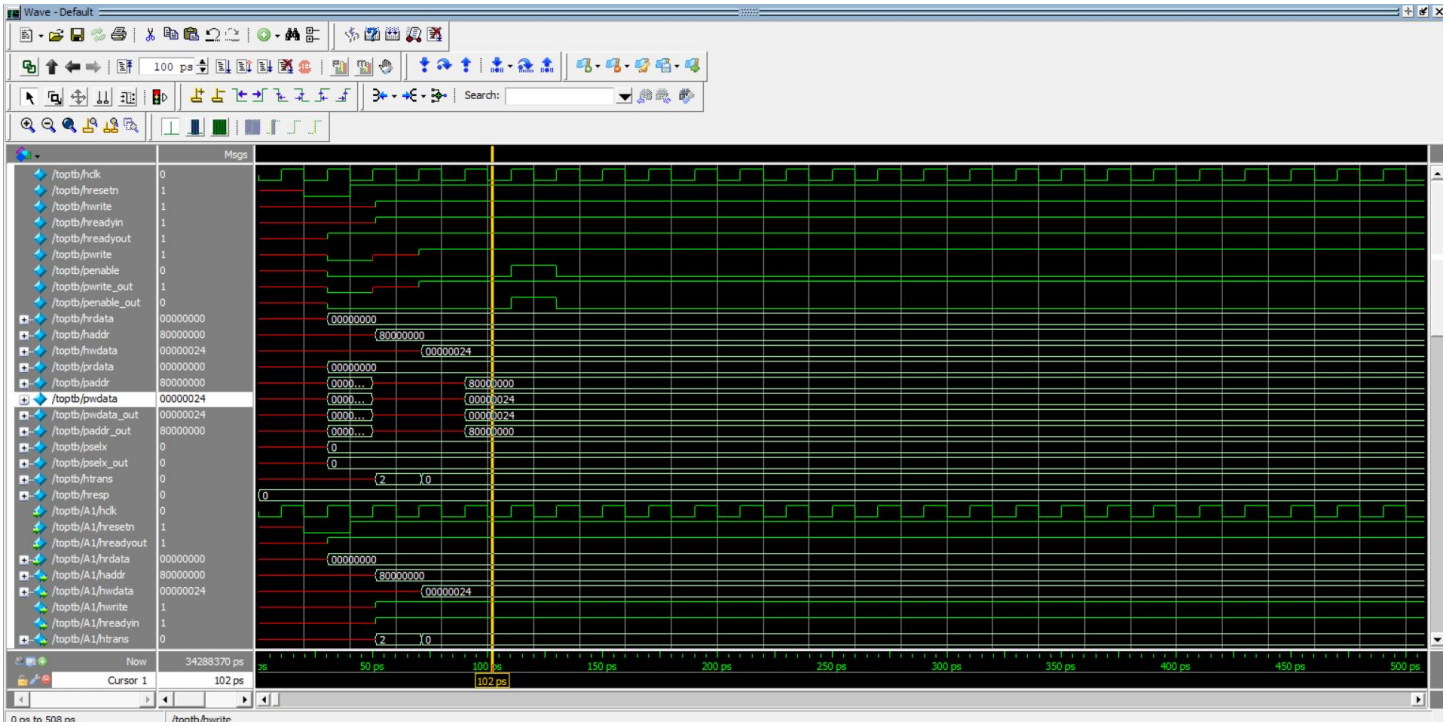
WAVE FORMS:

Top Module Output:

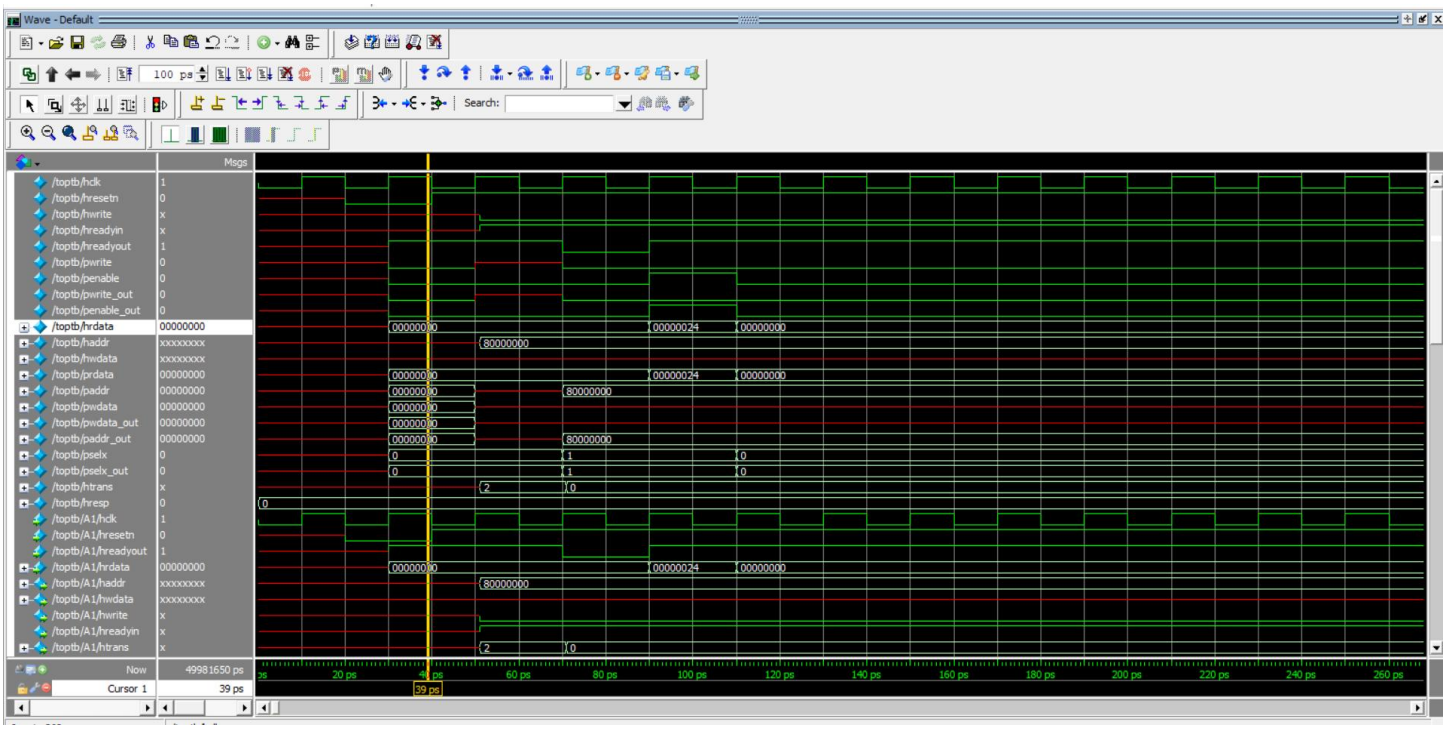


Sub-Blocks output waveforms:

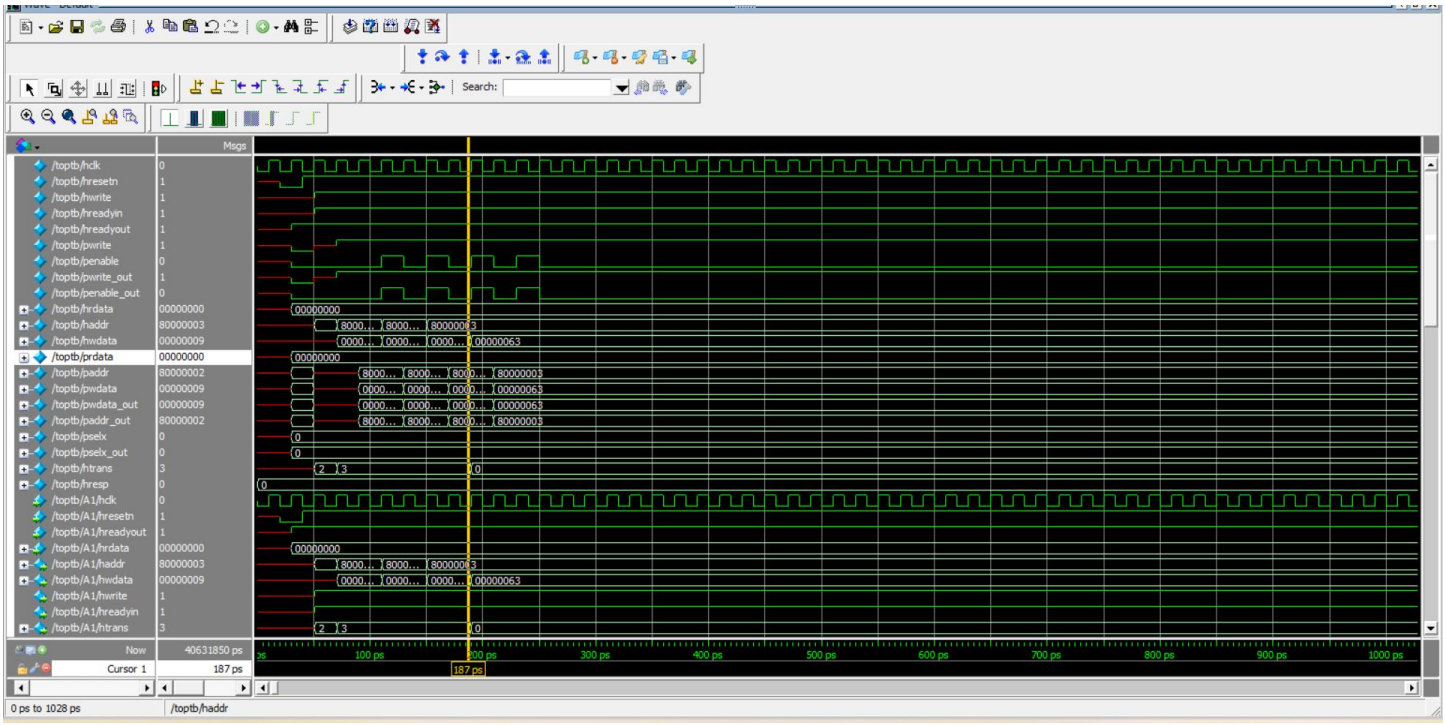
1) A1.single_write():



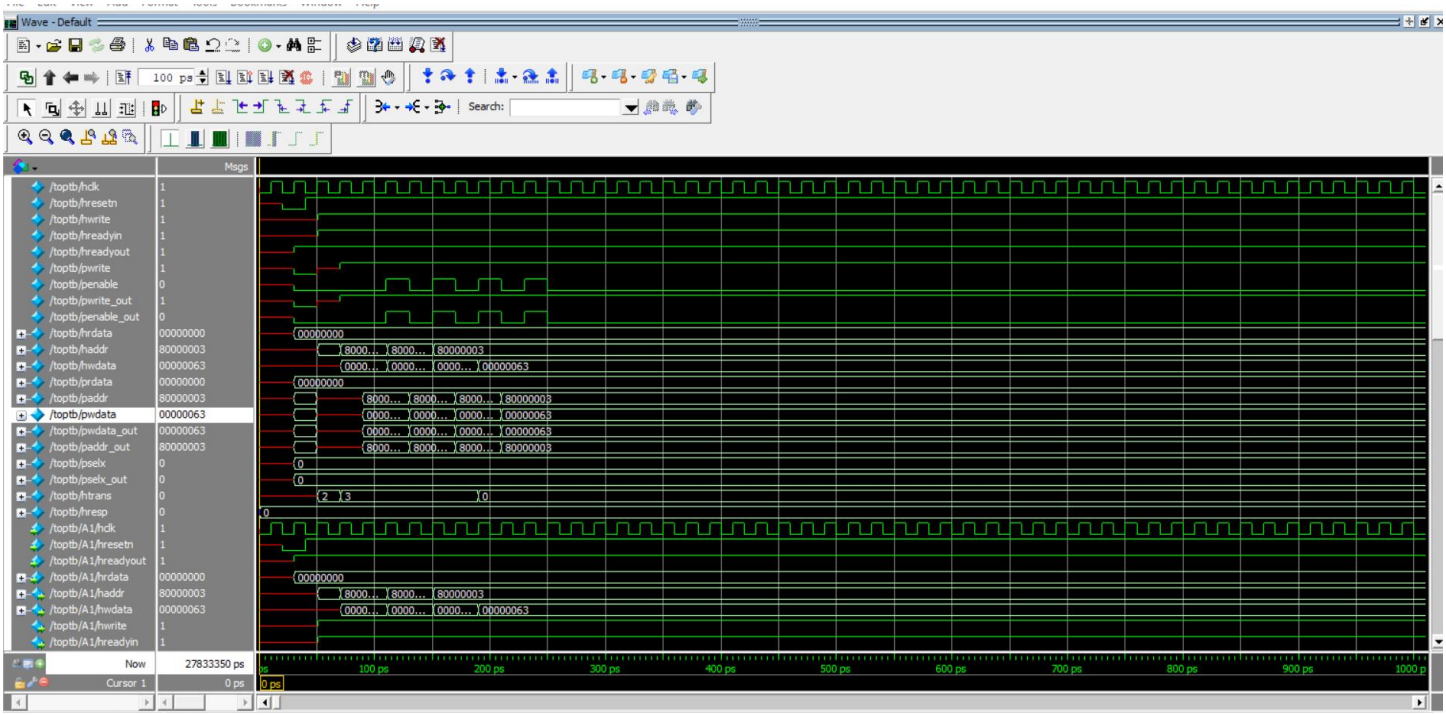
2) A1.single_read():



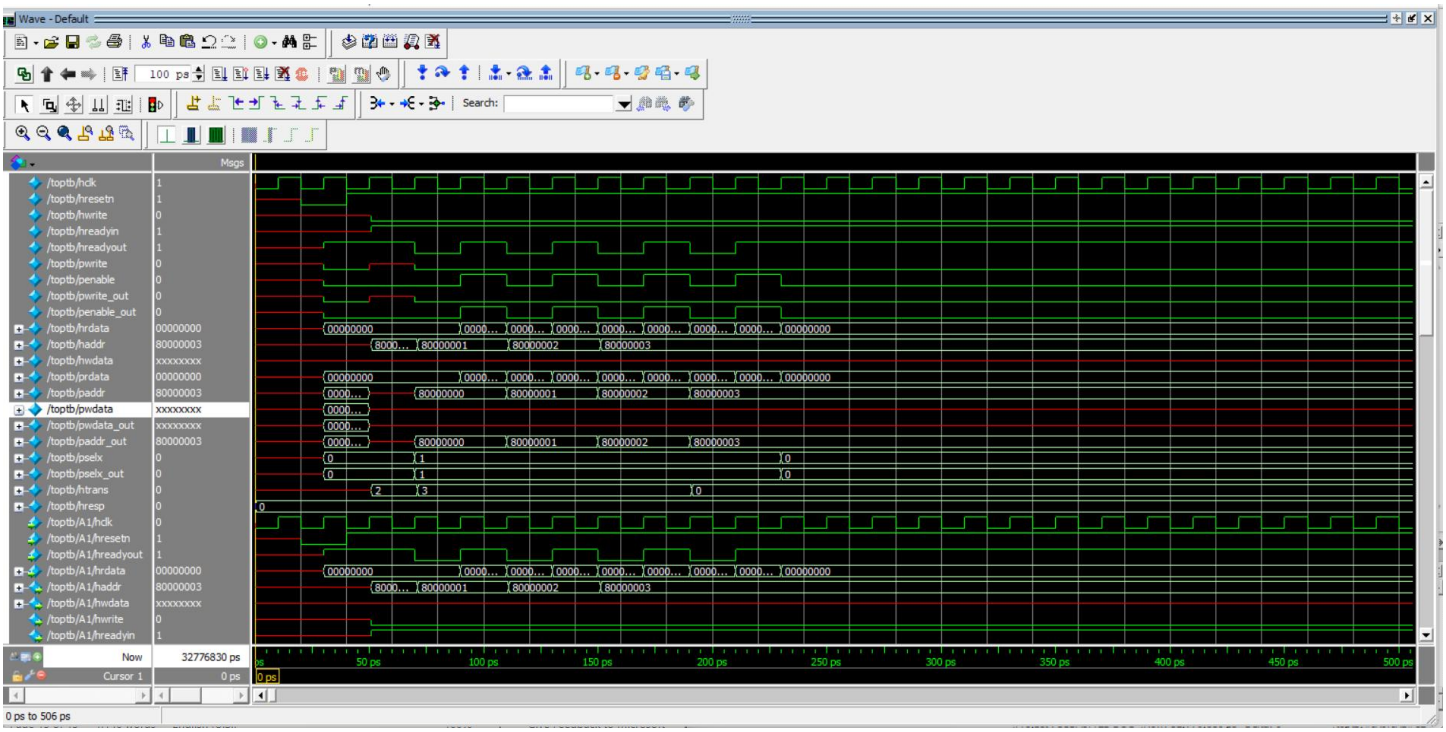
3) A1.burst_write_incr():



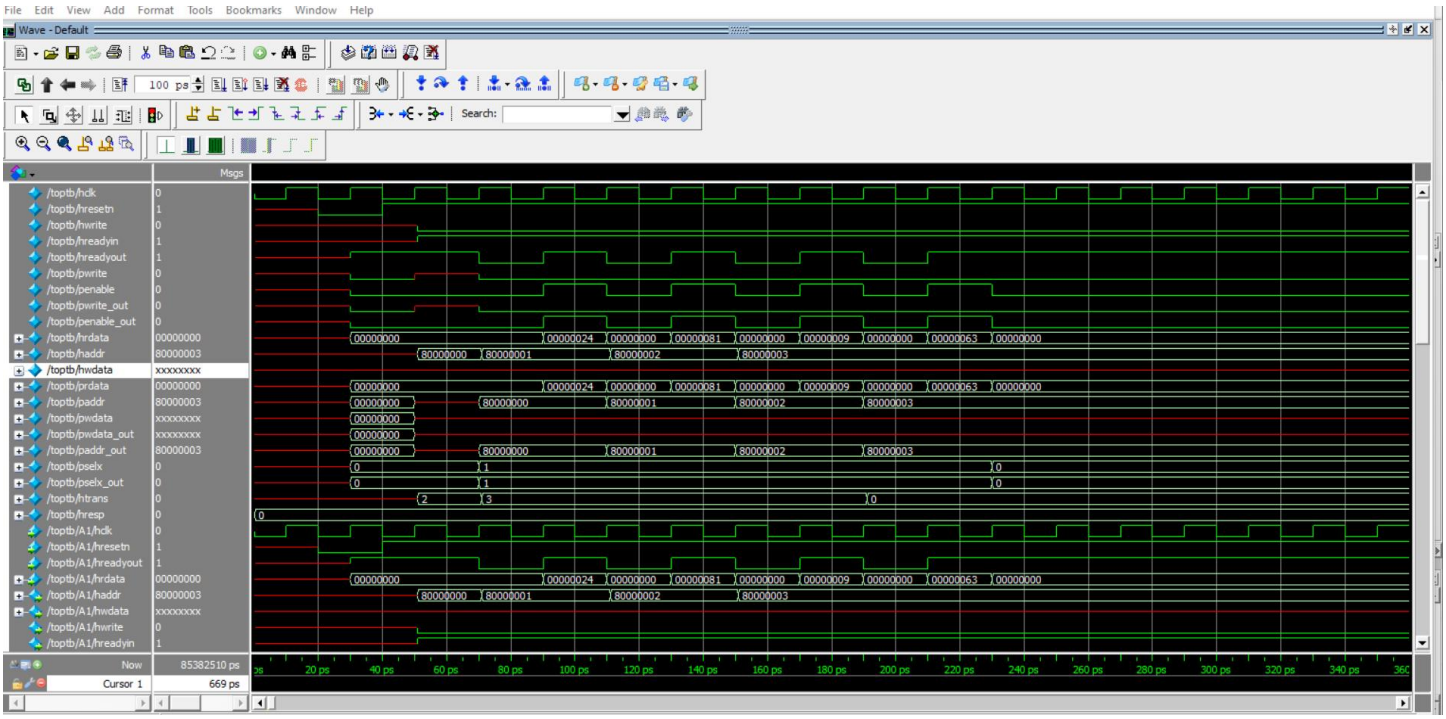
4) A1.burst_write_wrap():



5) A1.burst_read_incr():



6) A1.burst_read_wrap():



==========PROJECT COMPLETED==========