# ADVANCE PERIPHERAL BUS(APB) PROTOCOL VERIFICATION

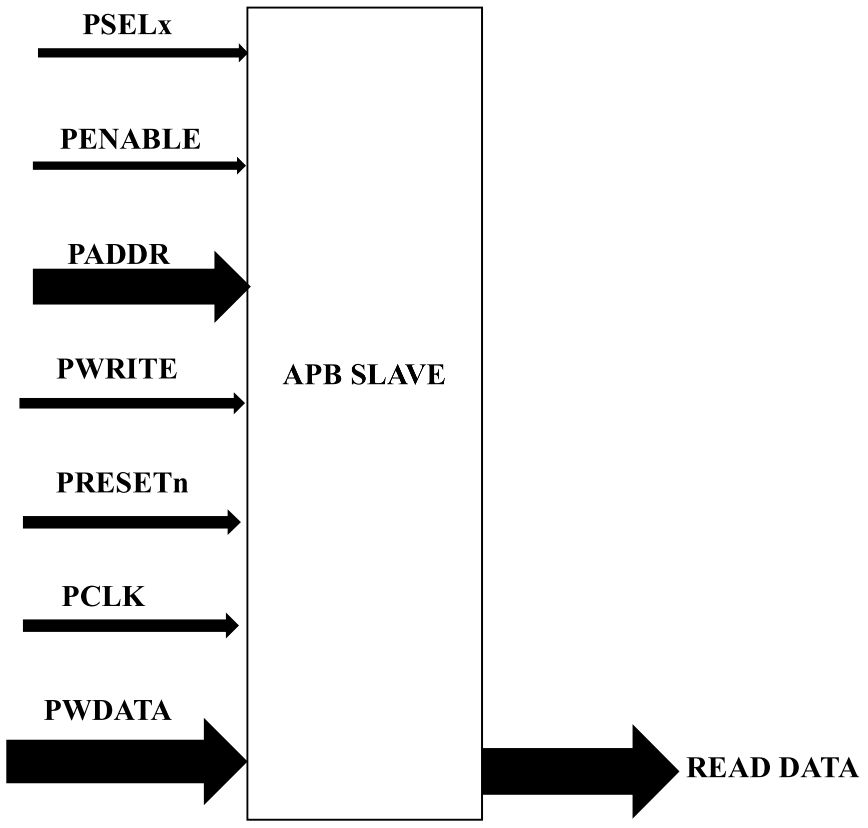
1.Aim: To verify APB protocol.

2.Software used: EDA playground.

3.Project Description:

APB is a part of AMBA protocol family. The APB protocol is non-pipelined and it supports to low-bandwidth peripherals that do not require the high performance of the AXI protocol.

**Block Diagram:**



**Signal Description:**

PSELx is a slave selection signal which indicates number of slaves that need to be included. Slave selection is done in such a way that maximum width of the signal is 32. The reason behind why only 32 because the main intension of the APB protocol is to make it quite simple not complicated.

Reset and clock are the basic signals. All the transactions are based on the clock signal. To initialize all other signal reset signal is used.

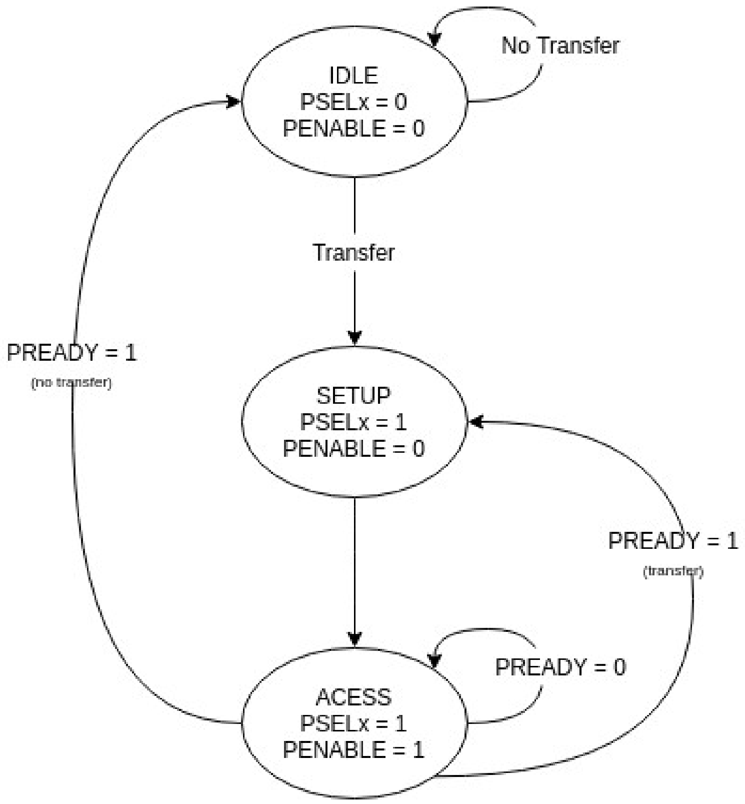
Address and write are the control signals based on this signals it is decided that whether we are going to read from the bus or write from the bus. Address indicates to which location we are going to read or write from. PWRITE=1 indicates write operation and PWRITE=0 indicates read operation.

Enable signal indicates that when the access has to actually happen. Slave signal indicates how many slaves you are going to support.

PWDATA, PRDATA are the data buses, PWDATA bus will carry what data has to be written to the slave. Whereas PRDATA bus will carry what data is to be read from the slave side to the master side.

**Operation of APB:**

APB has three operating states they are IDLE, SETUP, ACCESS.



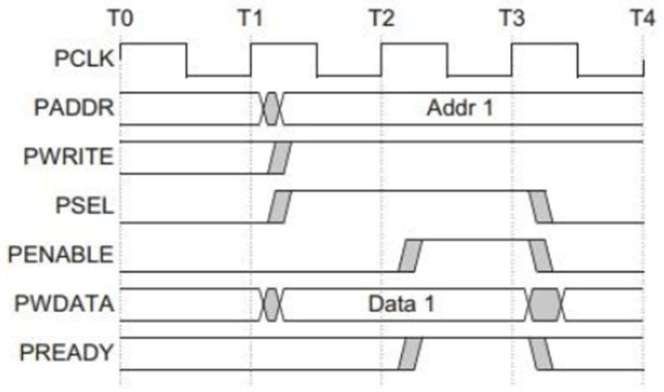
The default state where APB stays is IDLE state. When the reset occurs it will be in IDLE state. In IDLE state enable is zero and no slave is selected (PSELx=0). No transfer is also going to takes place because there is no slave selected, there is no slave at another end to communicate to receive the data.

SETUP, ACCESS are the two states where the data transaction actually depend on. We check for control signal in the SETUP state and data transactions(either PWDATA written to the slave or PRDATA is read from the slave) takes place in the ACCESS state. When we talk about the control signals PADDR, PWRITE, PRESET_n are checked.

In ACCESS state the initial step is to check whether PREADY is zero or not or is it equal to one.

If it is equal to one we can have transaction at that time. If it is zero it enters into wait state i.e it will stay in ACCESS state only and waits till PREADY is equal to one. PREADY indicates whether the slave is read for transaction or not. When it is in wait state no need to change any control signals because control signals are already registered in SETUP state. So, in ACCESS state no change in control signal just we check the reset. Once PREADY becomes high transfer will be completed, if there is a next transfer waiting in the cycle it will enter into the SETUP phase once again and then cycle repeats. If PREADY is equal to one and no transfer, ACCESS state enters into IDLE state again once after reaching the IDLE state it look for other transfers and continues the same process.

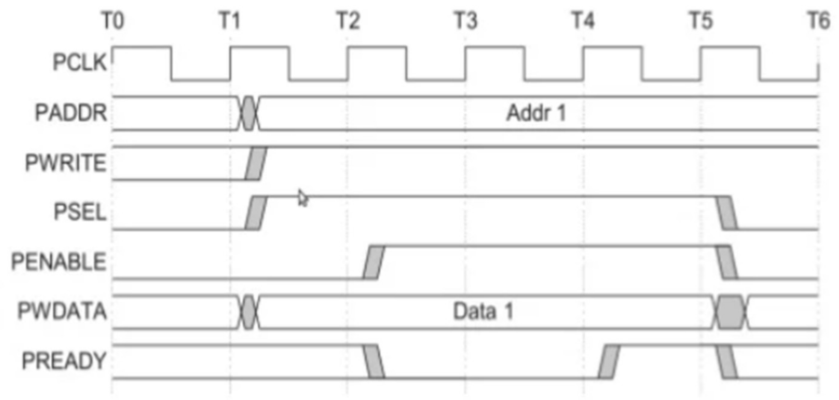
**Write transfer with no wait state:**



From above waveforms it is observed that initially the clock is in IDLE state, then SETUP state and at last it is in ACCESS state. In IDLE state, PSEL and PENABLE are zeroes, in SETUP state PSEL is high (1) and PENABLE is zero whereas in ACCESS state both PSEL and PENABLE are high (1).

PREADY signal is high in ACCESS state, whenever PREADY signal is high we will get the read data. When PWRITE is high it indicates that we are writing the data to the slave.

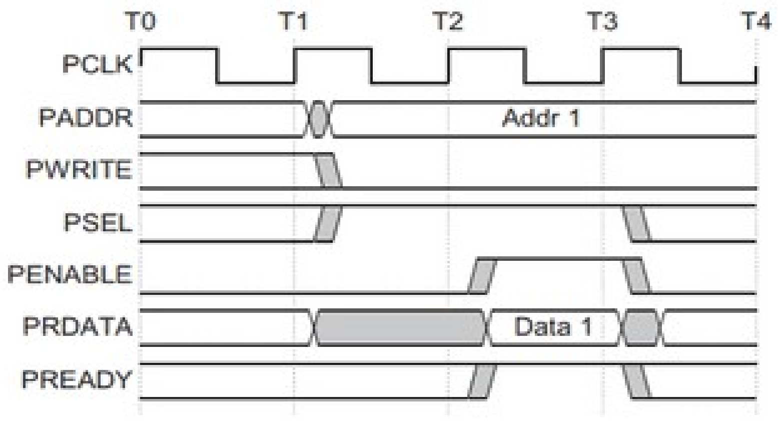
**Write transfer with wait state:**



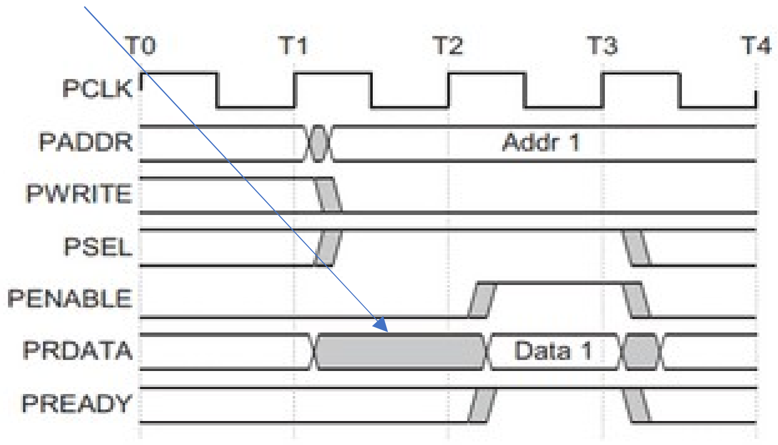
Above waveforms describes that in IDLE state both PSEL and PENABLE are zeroes. In SETUP state PSEL is high and PENABLE is zero. In ACCESS state both PSEL, PENABLE are high (1).

When both PSEL and PENABLE are high we can directly send the right data and read data also can be accepted and also it is observed that PREADY is zero in ACCESS state that means slave is not ready to receive. It will get the read data if and only if PREADY is high which indicates that the slave is ready to receive the data.

**Read transfer with no wait state:**

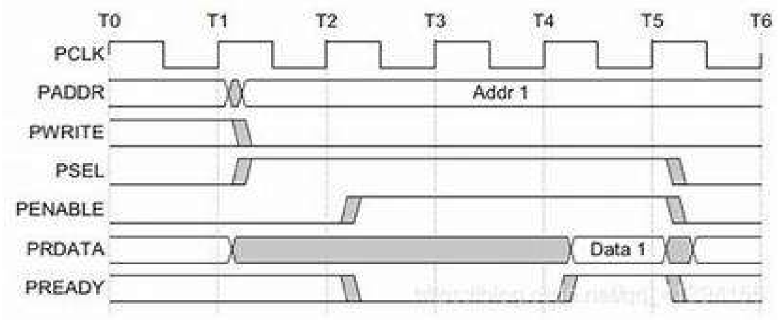


When PWRITE is low it indicates read transfer. When PRDATA exists slave is not ready to accept when PENABLE is zero.



Salve accepts the data only when PENABLE and PREADY are high. Data (Data 1) during third cycle is acceptable by the slave.

**Read transfer with wait state:**



Read transfer with wait state indicates that PREADY is low which indicates that slave is in other transfer. When PREADY is high then only it is reading the data even it got the data during second cycle. So we can conclude that data can be read only when PENABLE and PREADY are high.

**Code:**

Design:

```
module apb_slave(output reg[31:0]prdata, output reg pready, input[31:0] paddr, pwdata,
                input penable, psel, pwrite, pclk, prst_n);

  bit[1:0] present_state; // holds the current state
  bit[1:0] next_state; //holds the next state
  reg[31:0] mem[128:0];

  typedef enum {IDLE,SETUP,ACCESS,WAIT}fsm_state;
  fsm_state state;

  always@(posedge pclk, negedge prst_n) begin
    if(!prst_n)
      present_state<=IDLE;
    else
      present_state<=next_state;
  end
  always@(*) begin
    case(present_state)
IDLE:
  if(psel) // if psel is high transition to the SETUP state
    next_state=SETUP;
      else
        next_state=IDLE;
```

```
     SETUP:
  if(penable)
    next_state=ACCESS;
else
next_state=SETUP;

ACCESS:
  if(pready) begin
    if(!psel)
      next_state=IDLE;
    else
      next_state=SETUP;
  end
   else next_state=WAIT;

WAIT:
  if(pready)
    next_state=ACCESS;
      else
        next_state=WAIT;
default:next_state=IDLE;
    endcase
  end

  always@(posedge pclk, negedge prst_n) begin
    if(!prst_n) begin
      prdata<=0;
    end
    else begin
      if(present_state==ACCESS) begin
```

```
        if(pwrite) begin
          mem[paddr] <=pwdata;
        end
        else begin
          prdata= mem[paddr];
        end
      end
    end
  end
  initial begin
    pready=1;
    #300; force pready=0;
    #30; release pready;
  end
endmodule
```

Testbench:

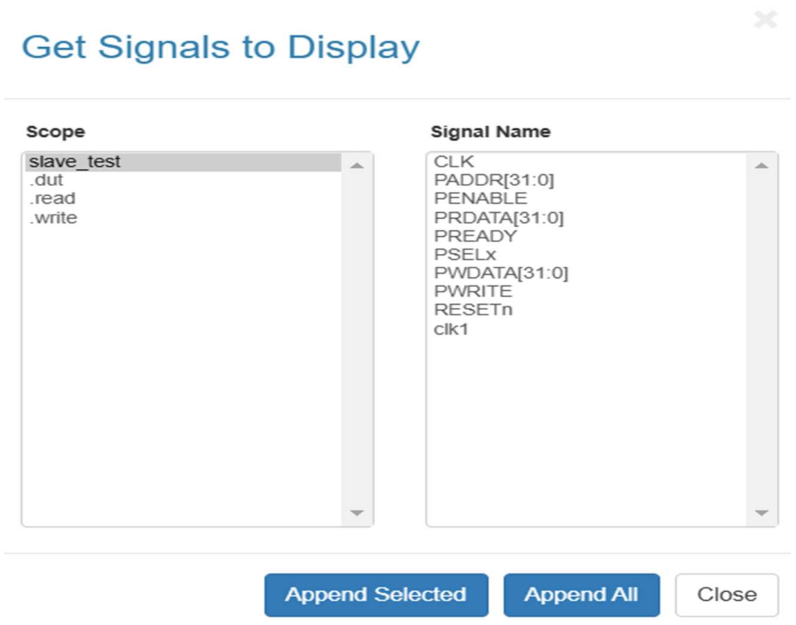
```
module slave_test;
   reg CLK;
  wire clk1;
  reg RESETn;
  reg [31:0] PADDR;
  reg [31:0] PWDATA;
  reg PWRITE;
  reg PSELx;
  reg PENABLE;
  wire PREADY;
  wire [31:0] PRDATA;

  //instantiating the DUT
```

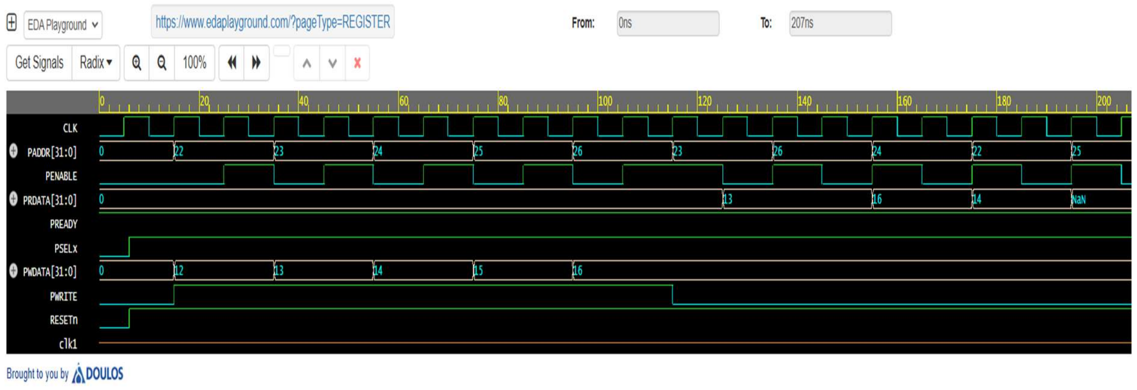
```
apb_slave dut(PRDATA,PREADY,PADDR,PWDATA,PENABLE,PSELx,PWRITE,CLK,RESETn);

always #5 CLK= ~CLK;   //clock generation
 task initialize;
  begin
   CLK=0;
   RESETn=0;
   PADDR=0;
   PWDATA=0;
   PWRITE=0;
   PSELx=0;
   PENABLE=0;
  end
 endtask

 task reset;
  begin
   CLK=0;
   RESETn=0;
   @(posedge CLK);
   #1;
   RESETn=1;
  end
 endtask

 task start;
  begin
   PSELx=1;
  end
 endtask
```

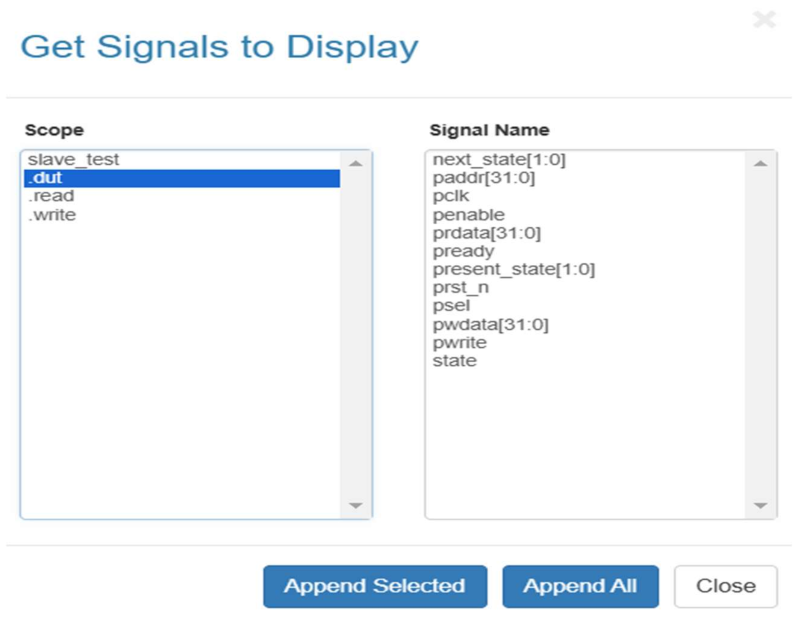
```
task write;
  input [31:0] data_in;
  input [31:0] adress_in;
  begin
    @(posedge CLK);
    PENABLE=0;
    PWRITE=1;
    PWDATA=data_in;
    PADDR=adress_in;
    @(posedge CLK);
    PENABLE=1;
  end
endtask

task read;
  input [31:0] adress_in;
  begin
    @(posedge CLK);
    PWRITE=0;
    PADDR=adress_in;
     PENABLE=1;
    @(posedge CLK);
    PENABLE=0;
  end
endtask

initial begin
  $dumpfile("APB.vcd");
  $dumpvars;
  initialize;
```

```
    reset;
    start;
    write(12,22);
    write(13,23);
    write(14,24);
    write(15,25);
    write(16,26);
    read(23);
    read(26);
    read(24);
    read(22);
    read(25);
    #2;
    $finish;
  end
endmodule
```

**Output Waveforms:**

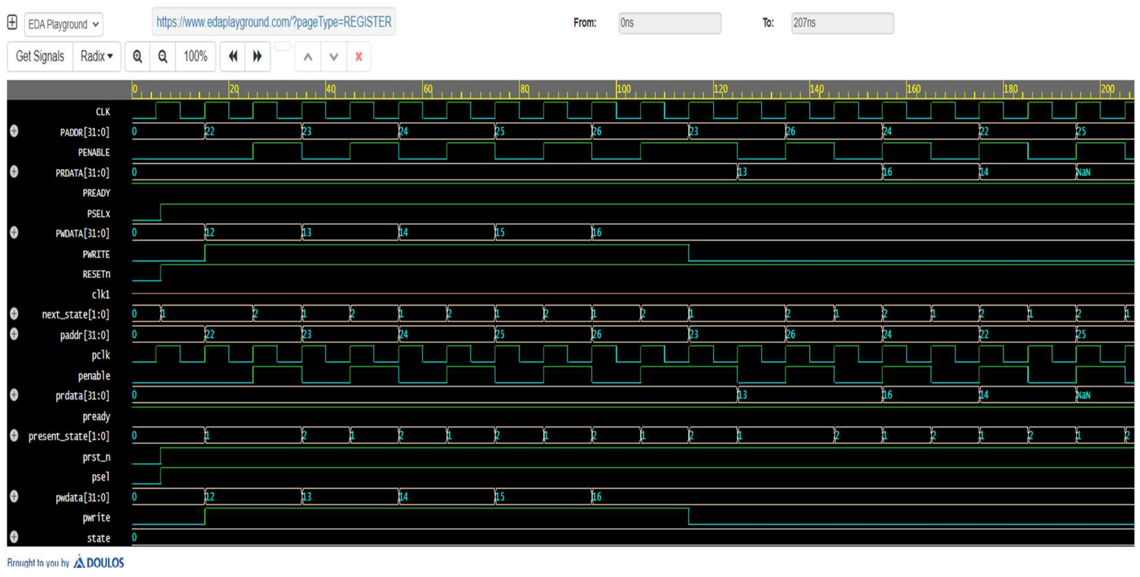


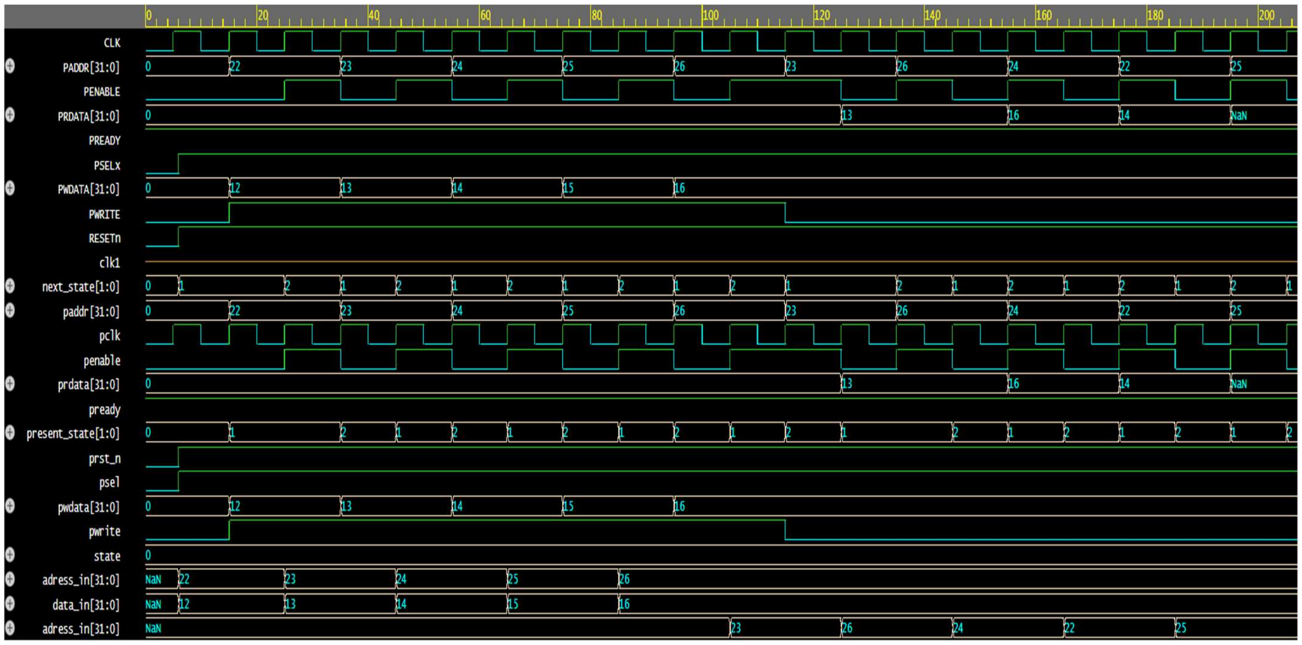



## Get Signals to Display

| Scope | Signal Name |
|---|---|
| slave_test | next_state[1:0] |
| .dut | paddr[31:0] |
| .read | pclk |
| .write | penable |
| | prdata[31:0] |
| | pready |
| | present_state[1:0] |
| | prst_n |
| | psel |
| | pwdata[31:0] |
| | pwrite |
| | state |

Append Selected　Append All　Close





Above waveform is the final waveform obtained when all the signals are appended.