Assignment 2

Name: Darshit Jain
Roll Number: B19EE024

- **Programming language used:** Verilog
- **Category:**
  Model AMBA APB **Write** and **Read Transfer With No Wait State**
- **Input format:**
  - The user is not required to provide any kind of input
  - **Testbench** has been created for both Write Transfer With No Wait State and Reach Transfer With No Wait State separately
- **Output format:**
  - Output can be verified from the **waveform** obtained on running the Verilog program with the waveform present in the *AMBA APB Protocol Specification* PDF provided with lab documents
  - An image of both the waveforms (from the program and the resource provided) have been attached below
- **Resources:**
  - https://youtu.be/LPTPe65o_Kk
  - https://www.youtube.com/watch?v=ZQa8DIJfa2s

- The design fairly and accurately mimics the **Write** and **Read Transfer With No Wait State** waveform given in the resource and can be used on a small scale to study about the protocol
- The output waveform exactly matches with the reference waveforms given
- Having used Verilog, we were able to implement transfers that actually happens in a manufactured product/real hardware system
- A proper, well-commented testbench was also used for providing the necessary inputs
- Necessary updates and operations are performed when the **currentPhase** changes, and that happens at the **rising edge** of the clock **PCLK**

Output Waveforms:

- **Write Transfer With No Wait State**

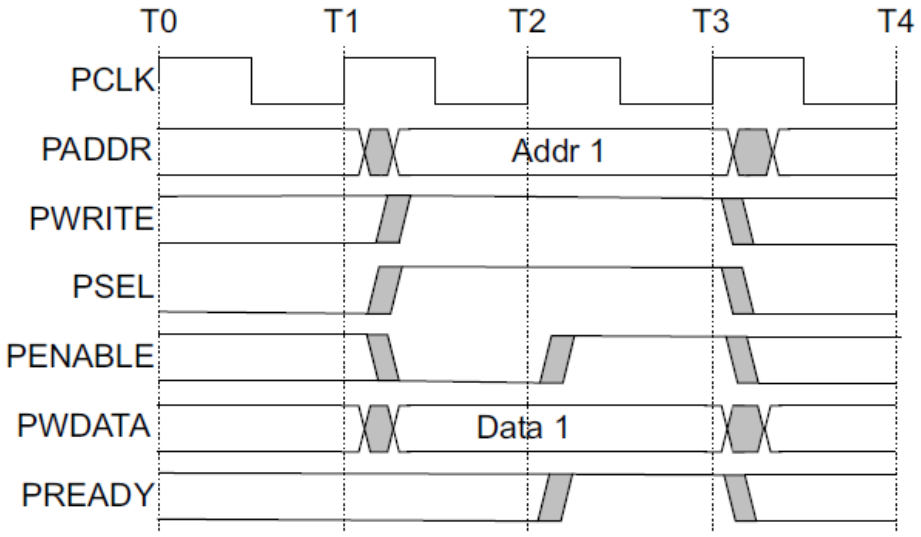

Figure: Reference waveform

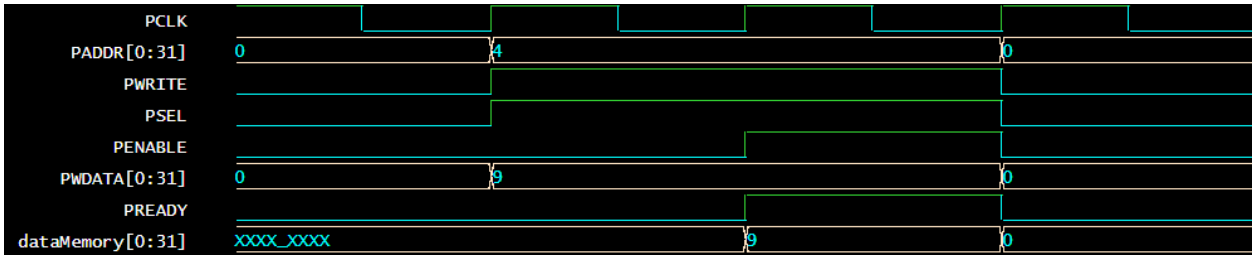

Figure: Waveform obtained

```
[2022-03-05 01:07:05 EST] iverilog '-Wall' design.sv testbench.sv  && unbuffer vvp a.out
VCD info: dumpfile dump.vcd opened for output.
Current Phase: IDLE, 0
Current Phase: SETUP, 40000
Current Phase: ACCESS, 80000
Current Phase: IDLE, 120000
Finding VCD file...
./dump.vcd
[2022-03-05 01:07:05 EST] Opening EPWave...
Done
```

Figure: Console Logs

(Time instant is displayed after the Current State)

```
`timescale 1ns / 1ps
module APBProtocolWRITE(PCLK, PADDR, PWRITE, PSEL, PENABLE, PWDATA, PREADY, dataMemory);
  input PCLK, PWRITE, PSEL, PENABLE;
  input [0:31] PADDR;
  input [0:31] PWDATA;

  reg [0:1] currentPhase = 0;
  reg [0:1] nextPhase = 0;

  output reg PREADY;
  output reg [0:31]dataMemory;

  always @(posedge PCLK)
    begin
      currentPhase <= nextPhase;
      if(nextPhase == 0 && $time != 160000)
        begin
          $display("Current Phase: IDLE, %0t", $time);
        end
      else if(nextPhase == 1 && $time != 160000)
        begin
          $display("Current Phase: SETUP, %0t", $time);
        end
      else if(nextPhase == 2 && $time != 160000)
        begin
          $display("Current Phase: ACCESS, %0t", $time);
        end
    end

  // Perform operations whenever the Current Phase changes
  always @(currentPhase)
    begin
      // IDLE PHASE
      if(currentPhase == 0)
        begin
          PREADY = 0;
          nextPhase <= 1;
        end
      // SETUP PHASE
      else if(currentPhase == 1)
        begin
          nextPhase <= 2;
        end
      // ACCESS PHASE
      else if(currentPhase == 2 && PSEL == 1 && PENABLE == 0)
        begin
          PREADY = 1;
          nextPhase <= 1; // Move to SETUP PHASE if more transfers are remaining
        end
      else
        begin
          PREADY = 1;
          dataMemory = PWDATA;
          nextPhase <= 0; // Move to IDLE PHASE if more transfers are remaining
        end
    end
endmodule
```

Figure: Design Code

```
`timescale 1ns / 1ps
module testbenchWRITE;
  reg PCLK, PWRITE, PSEL, PENABLE;
  reg [0:31] PADDR;
  reg [0:31] PWDATA;
  wire PREADY;
  wire [0:31]dataMemory;

  APBProtocolWRITE noWait(PCLK, PADDR, PWRITE, PSEL, PENABLE, PWDATA, PREADY, dataMemory);

  task writeTransferWithNoWaitStates;
    begin
      // IDLE PHASE
      PCLK = 1; PADDR = 0; PWRITE = 0; PSEL = 0; PENABLE = 0; PWDATA = 0;

      // SETUP PHASE
      // PSEL is asserted
      // Until PREADY becomes high, PADDR, PWRITE and PWDATA must be valid/ stable
      @(posedge PCLK);
        PSEL = 1;
        PADDR = {$urandom}%32; // Index between 0 and 31
        PWRITE = 1;
        PWDATA = {$urandom}%10; // Selecting a random value between 0 and 10

      // State jumps to ACCESS PHASE in the next cycle
      // PENABLE and PREADY are asserted to perform write operations
      @(posedge PCLK);
        PENABLE = 1;

      // Assuming no more transfers are required, we jump to IDLE PHASE
      // PENABLE and PSEL are deasserted and other signals too
      @(posedge PCLK);
        PADDR = 0;
        PWRITE = 0;
        PSEL = 0;
        PENABLE = 0;
        PWDATA = 0;
    end
  endtask

    initial
      begin
        $dumpfile("dump.vcd");
        $dumpvars;
        writeTransferWithNoWaitStates;
      end

  always #20 PCLK = ~PCLK;
  initial #160 $finish;

endmodule
```

Figure: Testbench Code

- **Read Transfer With No Wait State**

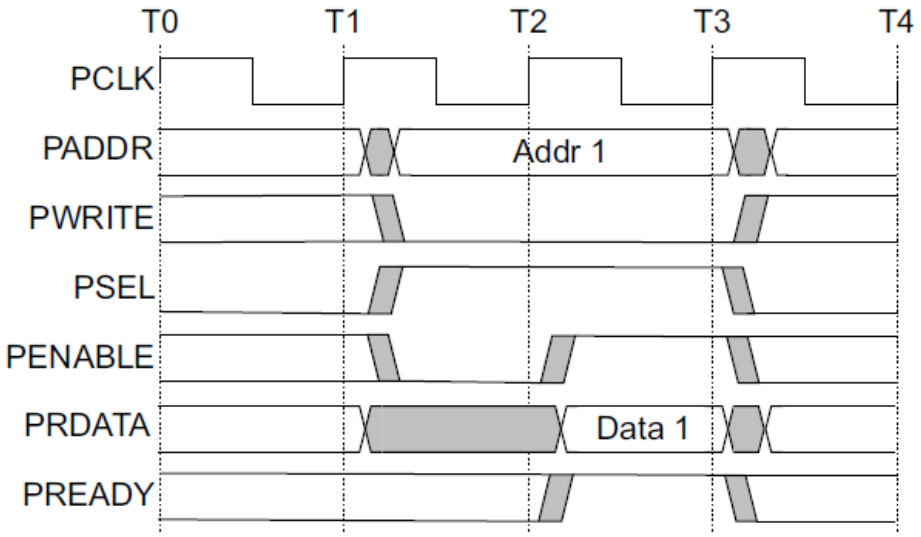

Figure: Reference waveform

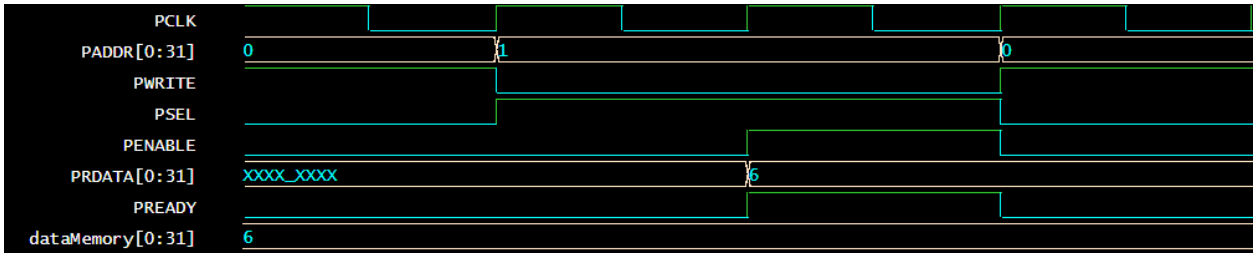

Figure: Waveform obtained

```
[2022-03-05 01:08:18 EST] iverilog '-Wall' design.sv testbench.sv  && unbuffer vvp a.out
VCD info: dumpfile dump.vcd opened for output.
Current Phase: IDLE, 0
Current Phase: SETUP, 40000
Current Phase: ACCESS, 80000
Current Phase: IDLE, 120000
Finding VCD file...
./dump.vcd
[2022-03-05 01:08:18 EST] Opening EPWave...
Done
```

Figure: Console Logs

(Time instant is displayed after the Current State)

```
`timescale 1ns / 1ps
module APBProtocolREAD(PCLK, PADDR, PWRITE, PSEL, PENABLE, PRDATA, PREADY, dataMemory);
  input PCLK, PWRITE, PSEL, PENABLE;
  input [0:31] PADDR;
  input [0:31] dataMemory;

  reg [0:1] currentPhase = 0;
  reg [0:1] nextPhase = 0;

  output reg PREADY;
  output reg [0:31] PRDATA;

  always @(posedge PCLK)
    begin
      currentPhase <= nextPhase;
      if(nextPhase == 0 && $time != 160000)
        begin
          $display("Current Phase: IDLE, %0t", $time);
        end
      else if(nextPhase == 1 && $time != 160000)
        begin
          $display("Current Phase: SETUP, %0t", $time);
        end
      else if(nextPhase == 2 && $time != 160000)
        begin
          $display("Current Phase: ACCESS, %0t", $time);
        end
    end

  // Perform operations whenever the Current Phase changes
  always @(currentPhase)
    begin
      // IDLE PHASE
      if(currentPhase == 0)
        begin
          PREADY = 0;
          nextPhase <= 1;
        end
      // SETUP PHASE
      else if(currentPhase == 1)
        begin
          nextPhase <= 2;
        end
      // ACCESS PHASE
      else if(currentPhase == 2 && PSEL == 1 && PENABLE == 0)
        begin
            PREADY = 1;
            nextPhase <= 1; // Move to SETUP PHASE if more transfers are remaining
        end
      else
        begin
            PREADY = 1;
            PRDATA = dataMemory;
            nextPhase <= 0; // Move to IDLE PHASE if more transfers are remaining
        end
    end
endmodule
```

Figure: Design Code

```
`timescale 1ns / 1ps
module testbenchWRITE;
  reg PCLK, PWRITE, PSEL, PENABLE;
  reg [0:31] PADDR;
  reg [0:31] dataMemory;
  wire PREADY;
  wire [0:31] PRDATA;

  APBProtocolREAD noWait(PCLK, PADDR, PWRITE, PSEL, PENABLE, PRDATA, PREADY, dataMemory);

  task readTransferWithNoWaitStates;
    begin
      // IDLE PHASE
      PCLK = 1; PADDR = 0; PWRITE = 1; PSEL = 0; PENABLE = 0; dataMemory = {$urandom}%10;

      // SETUP PHASE
      // PSEL is asserted
      // Until PREADY becomes high, PADDR and PWRITE must be valid/ stable
      @(posedge PCLK);
        PSEL = 1;
        PENABLE = 0;
        PADDR = {$urandom}%32; // Index between 0 and 31
        PWRITE = 0; // Selecting a random value between 0 and 10

      // State jumps to ACCESS PHASE in the next cycle
      // PENABLE and PREADY are asserted to perform write operations
      @(posedge PCLK);
        PENABLE = 1;

      // Assuming no more transfers are required, we jump to IDLE PHASE
      // PENABLE and PSEL are deasserted and other signals too
      @(posedge PCLK);
        PADDR = 0;
        PWRITE = 1;
        PSEL = 0;
        PENABLE = 0;
    end
  endtask

    initial
      begin
        $dumpfile("dump.vcd");
        $dumpvars;
        readTransferWithNoWaitStates;
      end

  always #20 PCLK = ~PCLK;
  initial #160 $finish;

endmodule
```

Figure: Testbench Code

Output comments:

- It can be noticed that the **PREADY** signal is asserted at the same time when **PENABLE** is asserted. If it had been a wait state transfer, **PREADY** would have waited for one complete cycle after **PENABLE** is asserted, to get asserted
- It was made sure that when the select signal, **PSEL**, is asserted, **PADDR, PWRITE,** and **PWDATA**/**PRDATA** are also asserted and are stable/valid before **PENABLE** and **PREADY** are asserted
- The data is stored/read from the **dataMemory** based on the address provided.
    - In **Write** transfer, data from **PWDATA** is written to **dataMemory** once all **PWRITE, PSEL, PENABLE,** and **PREADY** are high/asserted
    - In **Read** transfer, data from **dataMemory** is stored/read into **PRDATA** only when **PWRITE** is low, and **PSEL, PENABLE,** and **PREADY** are high/asserted
- It can be noticed that all the transfers and updating operations take place during the positive edge of the clock **PCLK**
- It is assumed that only one transfer takes place and hence, at the end of the transfer, **PENABLE**, **PSEL**, **PREADY,** and **PWRITE** are de-asserted (**PWRITE** becomes low when data is to be read and then becomes high once the data is ready in the Read Transfer Protocol)