APB PROTOCOL

SUBMITTED BY
MELVIN RIJOHN T
15/03/2025

MASTER

```
`timescale 1ns / 1ns

module master_bridge (
    input [8:0] apb_write_paddr, // Write address for APB bus
    apb_read_paddr,              // Read address for APB bus
    input [7:0] apb_write_data, // Data to be written to APB bus
    PRDATA,                      // Data read from APB bus
    input PRESETn,               // Active-low reset signal
    PCLK,                        // Clock signal
    READ_WRITE,                  // Read (1) or Write (0) control signal
    transfer,                    // Transfer enable signal
    PREADY,                      // Peripheral ready signal
    output PSEL1, PSEL2,         // Peripheral select signals
    output reg PENABLE,          // Enable signal for APB transaction
    output reg [8:0] PADDR,      // APB address bus
    output reg PWRITE,           // Write enable signal
    output reg [7:0] PWDATA,     // Data to be written
    apb_read_data_out,           // Output for read data
    output PSLVERR               // Slave error flag
);

  // State registers
  reg [2:0] state, next_state;
  reg invalid_setup_error, setup_error, invalid_read_paddr, invalid_write_paddr,
invalid_write_data;

  // FSM State Encoding
  parameter IDLE = 3'b001, SETUP = 3'b010, ENABLE = 3'b100;

  // Sequential logic: State transition
  always @(posedge PCLK) begin
    if (!PRESETn) state <= IDLE; // Reset to IDLE state
    else state <= next_state;
  end

  // Combinational logic: Next state logic and output control
  always @(state, transfer, PREADY) begin
    if (!PRESETn) next_state = IDLE;
    else begin
      PWRITE = ~READ_WRITE; // Determine read/write operation
      case (state)

        IDLE: begin
          PENABLE = 0;
          if (!transfer) next_state = IDLE;
          else next_state = SETUP;
```

```
      end

      SETUP: begin
        PENABLE = 0;
        if (READ_WRITE) PADDR = apb_read_paddr; // Load read address
        else begin
          PADDR  = apb_write_paddr; // Load write address
          PWDATA = apb_write_data;  // Load write data
        end

        if (transfer && !PSLVERR) next_state = ENABLE;
        else next_state = IDLE;
      end

      ENABLE: begin
        if (PSEL1 || PSEL2) PENABLE = 1; // Enable APB transaction

        if (transfer & !PSLVERR) begin
          if (PREADY) begin
            if (!READ_WRITE) next_state = SETUP; // Continue writing
            else begin
              next_state = SETUP;
              apb_read_data_out = PRDATA; // Store read data
            end
          end else next_state = ENABLE; // Wait for PREADY
        end else next_state = IDLE;
      end

      default: next_state = IDLE;
    endcase
  end
end

// Assign peripheral select signals based on address
assign {PSEL1, PSEL2} = ((state != IDLE) ? (PADDR[8] ? {1'b0, 1'b1} : {1'b1, 1'b0}) :
2'd0);

// Error detection logic
always @(*) begin
  if (!PRESETn) begin
    setup_error = 0;
    invalid_read_paddr = 0;
    invalid_write_paddr = 0;
    invalid_write_data = 0;
  end else begin
    if (state == IDLE && next_state == ENABLE) setup_error = 1;
    else setup_error = 0;

    if ((apb_write_data === 8'dx) && (!READ_WRITE) && (state == SETUP || state == ENABLE))
      invalid_write_data = 1;
```

```
      else invalid_write_data = 0;

      if ((apb_read_paddr === 9'dx) && READ_WRITE && (state == SETUP || state == ENABLE))
        invalid_read_paddr = 1;
      else invalid_read_paddr = 0;

      if ((apb_write_paddr === 9'dx) && (!READ_WRITE) && (state == SETUP || state == ENABLE))
        invalid_write_paddr = 1;
      else invalid_write_paddr = 0;

      if (state == SETUP) begin
        if (PWRITE) begin
          if (PADDR == apb_write_paddr && PWDATA == apb_write_data) setup_error = 1'b0;
          else setup_error = 1'b1;
        end else begin
          if (PADDR == apb_read_paddr) setup_error = 1'b0;
          else setup_error = 1'b1;
        end
      end else setup_error = 1'b0;
    end

    // Generate overall error flag
    invalid_setup_error = setup_error || invalid_read_paddr || invalid_write_data ||
invalid_write_paddr;
  end

  assign PSLVERR = invalid_setup_error; // Assign error signal

endmodule
```

SLAVE

```
`timescale 1ns / 1ns

module slave (
    input PCLK,         // Clock signal
    PRESETn,            // Active-low reset signal
    input PSEL,         // Peripheral select signal
    PENABLE,            // Enable signal for APB transaction
    PWRITE,             // Read (0) or Write (1) control signal
    input [7:0] PADDR,  // Address bus
    PWDATA,             // Write data bus
    output [7:0] PRDATA, // Read data bus
    output reg PREADY    // Ready signal
);

  reg [7:0] reg_addr;    // Register to store address for read operation
  reg [7:0] mem[0:63];   // Memory array of 64 locations (8-bit each)
```

```
  // Assign output read data from memory
  assign PRDATA = mem[reg_addr];

  // APB Slave behavior based on PSEL, PENABLE, and PWRITE signals
  always @(*) begin
    if (!PRESETn) PREADY = 0; // Reset condition

    // Setup phase (Read operation): Address is latched
    else if (PSEL && !PENABLE && !PWRITE) begin
      PREADY = 0;
    end

    // Enable phase (Read operation): Provide read data
    else if (PSEL && PENABLE && !PWRITE) begin
      PREADY   = 1;
      reg_addr = PADDR;
    end

    // Setup phase (Write operation): Address and data setup
    else if (PSEL && !PENABLE && PWRITE) begin
      PREADY = 0;
    end

    // Enable phase (Write operation): Write data to memory
    else if (PSEL && PENABLE && PWRITE) begin
      PREADY = 1;
      mem[PADDR] = PWDATA;
    end

    else PREADY = 0; // Default case: Not ready
  end
endmodule
```

TOP MODULE

```
`timescale 1ns / 1ns

`include "master.sv" // Include Master Bridge module
`include "slave.sv"  // Include Slave module

module APB_Protocol (
    input PCLK,             // Clock signal
    PRESETn,                // Active-low reset signal
    transfer,               // Transfer enable signal
    READ_WRITE,             // Read (1) or Write (0) control signal
    input [8:0] apb_write_paddr, // APB write address
    input [7:0] apb_write_data,  // Data to be written
```

```
    input [8:0] apb_read_paddr,  // APB read address
    output PSLVERR,         // Slave error signal
    output [7:0] apb_read_data_out // Read data output
);

  // Internal signals
  wire [7:0] PWDATA, PRDATA, PRDATA1, PRDATA2; // Data buses
  wire [8:0] PADDR; // Address bus

  wire PREADY, PREADY1, PREADY2, PENABLE, PSEL1, PSEL2, PWRITE; // Control signals

  // Select appropriate ready signal based on address MSB
  assign PREADY = PADDR[8] ? PREADY2 : PREADY1;

  // Select appropriate read data based on address MSB when READ_WRITE is asserted
  assign PRDATA = READ_WRITE ? (PADDR[8] ? PRDATA2 : PRDATA1) : 8'dx;

  // Instantiate the Master Bridge module
  master_bridge dut_mas (
    apb_write_paddr,   // Write address
    apb_read_paddr,    // Read address
    apb_write_data,    // Write data
    PRDATA,            // Read data
    PRESETn,           // Reset signal
    PCLK,              // Clock signal
    READ_WRITE,        // Read/Write control
    transfer,          // Transfer enable
    PREADY,            // Ready signal
    PSEL1,             // Select signal for Slave 1
    PSEL2,             // Select signal for Slave 2
    PENABLE,           // Enable signal
    PADDR,             // Address bus
    PWRITE,            // Write enable signal
    PWDATA,            // Write data bus
    apb_read_data_out, // Read data output
    PSLVERR            // Slave error signal
  );

  // Instantiate Slave 1
  slave dut1 (
    PCLK,          // Clock signal
    PRESETn,       // Reset signal
    PSEL1,         // Select signal for Slave 1
    PENABLE,       // Enable signal
    PWRITE,        // Write enable signal
    PADDR[7:0],    // Address bus (lower 8 bits)
    PWDATA,        // Write data bus
    PRDATA1,       // Read data output
    PREADY1        // Ready signal
  );
```

```
  // Instantiate Slave 2
  slave dut2 (
    PCLK,          // Clock signal
    PRESETn,       // Reset signal
    PSEL2,         // Select signal for Slave 2
    PENABLE,       // Enable signal
    PWRITE,        // Write enable signal
    PADDR[7:0],    // Address bus (lower 8 bits)
    PWDATA,        // Write data bus
    PRDATA2,       // Read data output
    PREADY2        // Ready signal
  );

endmodule
```

TESTBENCH

```
`timescale 1ns / 1ns
`include "apb_top.sv" // Include the APB Protocol module

module test;

  // Declare testbench signals
  reg PCLK, PRESETn, transfer, READ_WRITE;
  reg [8:0] apb_write_paddr;
  reg [7:0] apb_write_data;
  reg [8:0] apb_read_paddr;
  wire [7:0] apb_read_data_out;
  wire PSLVERR;
  reg [7:0] data, expected;

  // Instantiate the APB Protocol module
  APB_Protocol dut_c (
      PCLK,
      PRESETn,
      transfer,
      READ_WRITE,
      apb_write_paddr,
      apb_write_data,
      apb_read_paddr,
      PSLVERR,
      apb_read_data_out
  );

  integer i, j;
```

```
  // Generate clock signal with 10ns period
  initial begin
    PCLK <= 0;
    forever #5 PCLK = ~PCLK;
  end

  initial begin
    // Initialize signals
    PRESETn  <= 0;
    transfer <= 0;
    READ_WRITE = 0;
    apb_write_paddr = 0;
    apb_write_data = 0;
    apb_read_paddr = 0;

    // Apply reset
    @(posedge PCLK) PRESETn = 1;

    // No write address available but request for write operation
    @(posedge PCLK) transfer = 1;
    repeat (2) @(posedge PCLK);
    @(negedge PCLK) Write_slave1;  // Perform write operation to slave 1

    repeat (3) @(posedge PCLK);
    Write_slave2;  // Perform write operation to slave 2

    @(posedge PCLK);
    apb_write_paddr = 9'd526;
    apb_write_data  = 9'd9;
    repeat (2) @(posedge PCLK);
    apb_write_paddr = 9'd22;
    apb_write_data  = 9'd35;
    repeat (2) @(posedge PCLK);

    // Initiate read operations
    @(posedge PCLK) READ_WRITE = 1;
    PRESETn  <= 0;
    transfer <= 0;
    @(posedge PCLK) PRESETn = 1;

    repeat (3) @(posedge PCLK) transfer = 1;  // No read address available but request for
read operation
    repeat (2) @(posedge PCLK) Read_slave1;  // Perform read operation from slave 1

    repeat (3) @(posedge PCLK);
    Read_slave2;  // Perform read operation from slave 2

    repeat (3) @(posedge PCLK);
    apb_read_paddr = 9'd45;  // Request read operation from an address not written to
    repeat (4) @(posedge PCLK);
```

```
  $finish;
end

// Task to write data to Slave 1
task Write_slave1;
  begin
    transfer = 1;
    for (i = 0; i < 8; i = i + 1) begin
      repeat (2)
        @(negedge PCLK) begin
          data = i;
          apb_write_data = 2 * i;
          apb_write_paddr = {1'b0, data};
        end
    end
  end
endtask

// Task to write data to Slave 2
task Write_slave2;
  begin
    for (i = 0; i < 8; i = i + 1) begin
      repeat (2)
        @(negedge PCLK) begin
          data = i;
          apb_write_paddr = {1'b1, data};
          apb_write_data = i;
        end
    end
  end
endtask

// Task to read data from Slave 1
task Read_slave1;
  begin
    for (j = 0; j < 8; j = j + 1) begin
      repeat (2)
        @(negedge PCLK) begin
          data = j;
          apb_read_paddr = {1'b0, data};
        end
    end
  end
endtask

// Task to read data from Slave 2
task Read_slave2;
  begin
    for (j = 0; j < 8; j = j + 1) begin
      repeat (2)
```

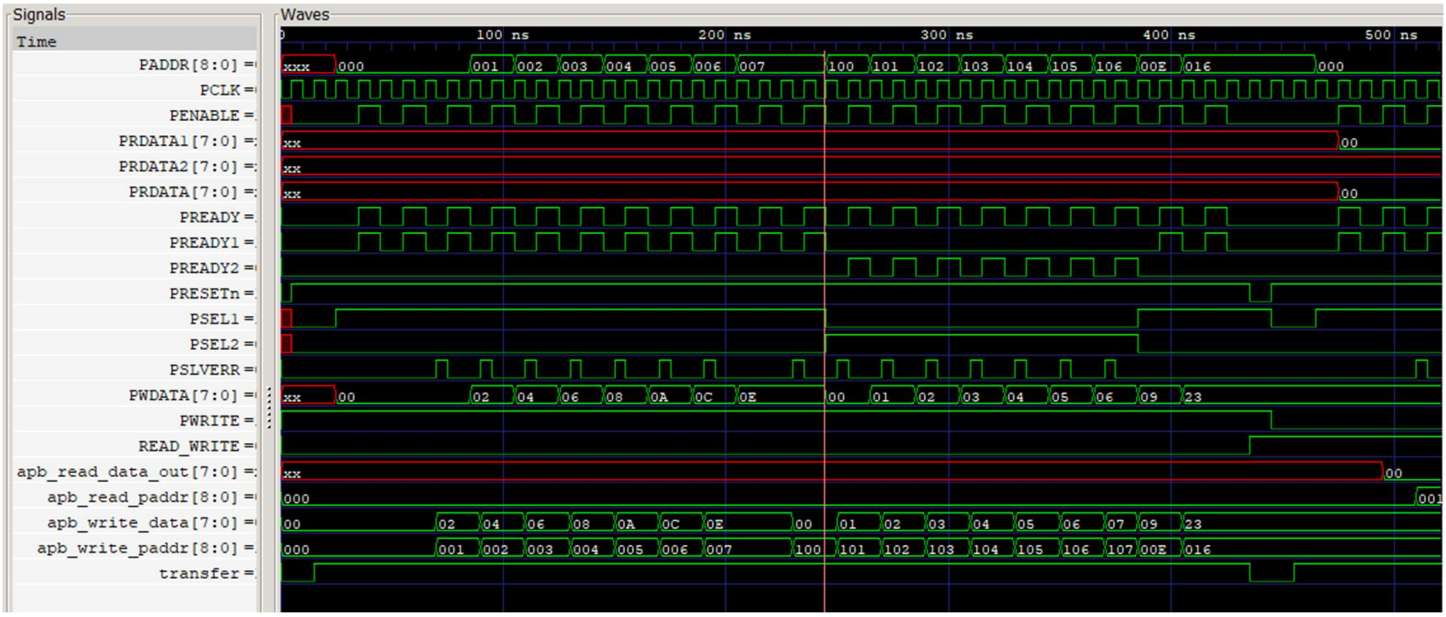
```
            @(negedge PCLK) begin
              data = j;
              apb_read_paddr = {1'b1, data};
            end
        end
      end
    endtask

    // Dump waveform for simulation
    initial begin
      $dumpfile("dump.vcd");
      $dumpvars;
    end

endmodule
```

OUTPUT

1. Write Operation



2. Read Operation