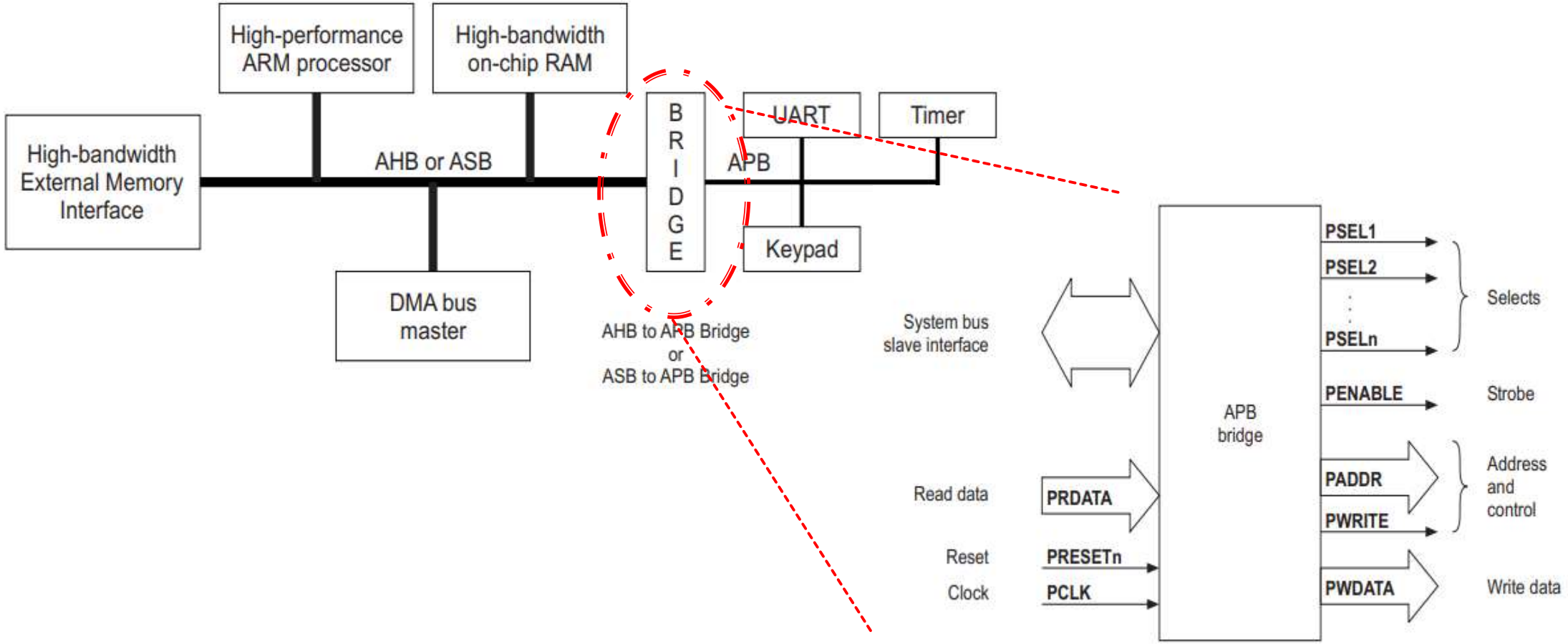
# APB PROTOCOL

Name: Mark Amgad

# Content

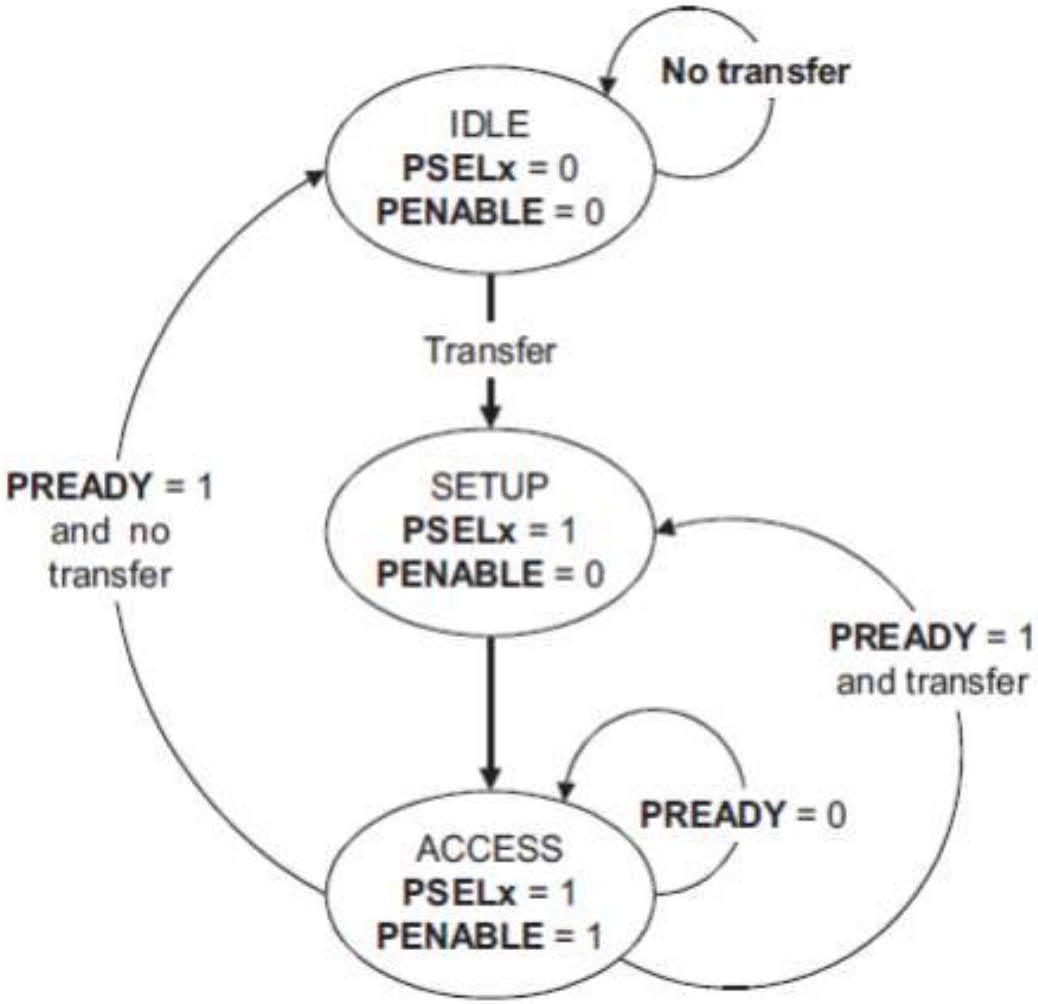
- AMBA Bus Architecture.
- APB Interface Diagram.
- Operation Of APB.
- BIN Description.
- Verilog Design Code , Test bench and Do file.
- Questa Sim Wave Form Simulation.
- XILINX Vivado Elaborated Design Schematic.
- XILINX Vivado Synthesized Design Schematic.
- Report Timing Summary.
- Tools.
- Sources.

# AMBA ARCHITECTURE



# Operation of APB

- IDLE
- SETUP
- ACCESS



# PIN Description

| SIGNAL | SOURCE | Description | WIDTH(Bit) |
|---|---|---|---|
| Transfer | System Bus | APB enable signal. If high APB is activated else APB is disabled | 1 |
| PCLK | Clock Source | All APB functionality occurs at rising edge. | 1 |
| PRESETn | System Bus | An active low signal. | 1 |
| PADDR | APB bridge | The APB address bus can be up to 32 bits. | 32 |
| PSEL | APB bridge | There is a PSEL for each slave. It's an active high signal. | 1 |
| PENABLE | APB bridge | It indicates the 2nd cycle of a data transfer. It's an active high signal. | 1 |
| PWRITE | APB bridge | Indicates the data transfer direction.<br>PWRITE=1 indicates APB write access(Master to slave)<br>PWRITE=0 indicates APB read access(Slave to master) | 1 |
| PREADY | Slave Interface | This is an input from Slave. It is used to enter access state. | 1 |
| PRDATA | Slave Interface | Read Data. The selected slave drives this bus during read operation | 32 |
| PWDATA | Slave Interface | Write data. This bus is driven by the peripheral bus bridge unit during write cycles when PWRITE is high. | 32 |

# Verilog Design Code

D:\APB\APB_Master.v - Sublime Text (UNREGISTERED)

APB_Master.v | APB_SLAVE.v | APB_WRAPPER.v | APB_tb.v | RUN_APB.do | Constraints_basys3.xdc

```verilog
module APB_Master(
    input PCLK,              //Clock. The rising edge of PCLK times all transfers on the APB.
    input PRESETn,           //Reset. The APB reset signal is active LOW.
    input TRANSFER,          //APB enable signal. If high APB is activated else APB is disabled
    input [31:0] PRDATA,     //Read Data from slave.The selected slave drives this bus during read cycles when PWRITE is LOW.
    input PREADY,            //Ready. The slave uses this signal to extend an APB transfer.
    input [31:0] address,    // Address for the APB transaction.
    input [31:0] write_data, // Data to be written (for write operations)
    input write_en,          // Write enable (1 for write, 0 for read)

    output reg PSELx,        //Select. The APB bridge unit generates this signal to each peripheral bus slave. It indicates that the slave device is selected and that a data transfer is required.
    output reg PENABLE,      //Enable. This signal indicates the second and subsequent cycles of an APB transfer.
    output reg PWRITE,       //Direction. This signal indicates an APB write access when HIGH and an APB read access when LOW.
    output reg [31:0] PADDR, //Address. This is the APB address bus. It is driven by the peripheral bus bridge unit.
    output reg [31:0] PWDATA, //Write data. This bus is driven by the peripheral bus bridge unit during write cycles when PWRITE is HIGH.
    output reg [31:0] read_data
    );

    localparam IDLE = 0;
    localparam SETUP = 1;
    localparam ACCESS = 2;

    reg [1:0] state , next_state;

    // STATE
    always @(posedge PCLK or negedge PRESETn) begin
        if (!PRESETn) begin
            state <= IDLE;  // Reset to IDLE state
        end else begin
            state <= next_state;  // Move to the next state
        end
    end

    // NEXT STATE
    always @(*) begin
        case (state)
            IDLE: begin
                if (TRANSFER) begin
                    next_state = SETUP;  // Move to SETUP on transfer signal
                end else begin
                    next_state = IDLE;   // Remain in IDLE
                end
            end

            SETUP: begin
                next_state = ACCESS;  // Go to ACCESS after setup
            end

            ACCESS: begin
                if (PREADY) begin
                    next_state = IDLE;  // Return to IDLE after successful transfer
                end else begin
                    next_state = ACCESS; // Remain in ACCESS until PREADY
                end
            end
```

# Verilog Design Code

```
            ACCESS: begin
                if (PREADY) begin
                    next_state = IDLE;  // Return to IDLE after successful transfer
                end else begin
                    next_state = ACCESS; // Remain in ACCESS until PREADY
                end
            end

            default: next_state = IDLE;  // Default state is IDLE
        endcase
    end

    // OUTPUT
    always @(posedge PCLK or negedge PRESETn) begin
        if (!PRESETn) begin
            PADDR <= 32'b0;
            PSELx <= 1'b0;
            PENABLE <= 1'b0;
            PWRITE <= 1'b0;
            PWDATA <= 32'b0;
            read_data <= 32'b0;
        end else begin
            case (state)
                IDLE: begin
                    PSELx <= 1'b0;
                    PENABLE <= 1'b0;
                end

                SETUP: begin
                    PADDR <= address;     // Set the address bus
                    PWRITE <= write_en;   // Set write direction based on write_en
                    PSELx <= 1'b1;        // Select the slave
                    PWDATA <= write_data; // Set the write data
                    PENABLE <= 1'b0;      // PENABLE is low in the setup phase
                end

                ACCESS: begin
                    PENABLE <= 1'b1;  // Enable data transfer in the access phase
                    if (!write_en && PREADY) begin
                        read_data <= PRDATA; // Capture the read data during read operation
                    end
                end
            endcase
        end
    end

endmodule
```

# Verilog Design Code

D:\APB\APB_SLAVE.v - Sublime Text (UNREGISTERED)

APB_SLAVE.v | APB_WRAPPER.v | APB_tb

```
module APB_Slave (
    input PCLK,
    input PRESETn,
    input PSELx,
    input PENABLE,
    input PWRITE,
    input [31:0] PADDR,
    input [31:0] PWDATA,
    output reg PREADY,
    output reg [31:0] PRDATA
);

    reg [31:0] memory [0:7];

    always @(posedge PCLK or negedge PRESETn) begin
        if (!PRESETn) begin
            PRDATA <= 32'b0;
            PREADY <= 1'b0;
        end
        else begin
            if (PSELx && PENABLE) begin
                if (PWRITE) begin
                    memory[PADDR[2:0]] <= PWDATA;
                    PREADY <= 1'b1;
                end
                else begin
                    PRDATA <= memory[PADDR[2:0]];
                    PREADY <= 1'b1;
                end
            end
            else begin
                PREADY <= 1'b0;
            end
        end
    end
endmodule
```

# Verilog Design Code

D:\APB\APB_WRAPPER.v - Sublime Text (UNREGISTERED)

APB_SLAVE.v | APB_WRAPPER.v | APB_tb.v

```verilog
module APB_WRAPPER(
    input PCLK,             // Clock source
    input PRESETn,          // Active LOW Reset
    input TRANSFER,         // Signal to start the transfer (Handshake)
    input [31:0] address,   // Address for the APB transaction
    input [31:0] write_data, // Data to be written (for write operations)
    input write_en,         // Write enable (1 for write, 0 for read)

    output [31:0] read_data // Read data from the slave
);

    wire PSELx;
    wire PENABLE;
    wire PWRITE;
    wire [31:0] PADDR;
    wire [31:0] PWDATA;
    wire [31:0] PRDATA;
    wire PREADY;

    APB_Master MASTER (
        .PCLK(PCLK),
        .PRESETn(PRESETn),
        .PREADY(PREADY),
        .PRDATA(PRDATA),
        .TRANSFER(TRANSFER),
        .address(address),
        .write_data(write_data),
        .write_en(write_en),
        .PADDR(PADDR),
        .PSELx(PSELx),
        .PENABLE(PENABLE),
        .PWRITE(PWRITE),
        .PWDATA(PWDATA),
        .read_data(read_data)
    );

    APB_Slave SLAVE (
        .PCLK(PCLK),
        .PRESETn(PRESETn),
        .PSELx(PSELx),
        .PENABLE(PENABLE),
        .PREADY(PREADY),
        .PWRITE(PWRITE),
        .PADDR(PADDR),
        .PWDATA(PWDATA),
        .PRDATA(PRDATA)
    );

endmodule
```

# Test bench

D:\APB\APB_tb.v - Sublime Text (UNREGISTERED)

APB_SLAVE.v | APB_WRAPPER.v | APB_tb.v | RUN_APB.c

```
module APB_tb;

    reg PCLK;
    reg PRESETn;
    reg TRANSFER;
    reg [31:0] address;
    reg [31:0] write_data;
    reg write_en;
    wire [31:0] read_data;

    // Instantiate APB Wrapper
    APB_WRAPPER WRAPPER (
        .PCLK(PCLK),
        .PRESETn(PRESETn),
        .TRANSFER(TRANSFER),
        .address(address),
        .write_data(write_data),
        .write_en(write_en),
        .read_data(read_data)
    );

    // Clock generation
    initial begin
        PCLK = 0;
        forever
        #5 PCLK = ~PCLK;
    end

    initial begin
        PRESETn = 0;
        TRANSFER = 0;
        address = 32'h0;
        write_data = 32'h0;
        write_en = 0;

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

        // Test Write Operation (write to address 4)
        address = 32'h4;
        write_data = 32'hABCD;  // Data to write
        write_en = 1;
        TRANSFER = 1;           // Start transfer (setup phase)
        @(negedge PCLK);
        // Remain in transfer for the access phase
        @(negedge PCLK);        // Allow one more clock for access
        TRANSFER = 0;           // End transfer
        @(negedge PCLK);

        // Wait for a few clock cycles to ensure the write completes
        repeat(2) @(negedge PCLK);

        // Test Read Operation (read from address 4)
        address = 32'h4;        // Same address
        write_en = 0;
```

Line 35, Column 1

# Test bench

```
        // Test Write Operation (write to address 4)
        address = 32'h4;
        write_data = 32'hABCD;  // Data to write
        write_en = 1;
        TRANSFER = 1;           // Start transfer (setup phase)
        @(negedge PCLK);
        // Remain in transfer for the access phase
        @(negedge PCLK);        // Allow one more clock for access
        TRANSFER = 0;           // End transfer
        @(negedge PCLK);

        // Wait for a few clock cycles to ensure the write completes
        repeat(2) @(negedge PCLK);

        // Test Read Operation (read from address 4)
        address = 32'h4;        // Same address
        write_en = 0;
        TRANSFER = 1;           // Start transfer (setup phase)
        @(negedge PCLK);
        // Remain in transfer for the access phase
        @(negedge PCLK);        // Allow one more clock for access
        TRANSFER = 0;           // End transfer
        @(negedge PCLK);

        // Wait for a few clock cycles for read completion
        @(negedge PCLK);
        @(negedge PCLK);

        // Display the read data value
        $display("Read Data: %h", read_data);  // Should be 32'hABCD

        $stop;
    end

endmodule
```

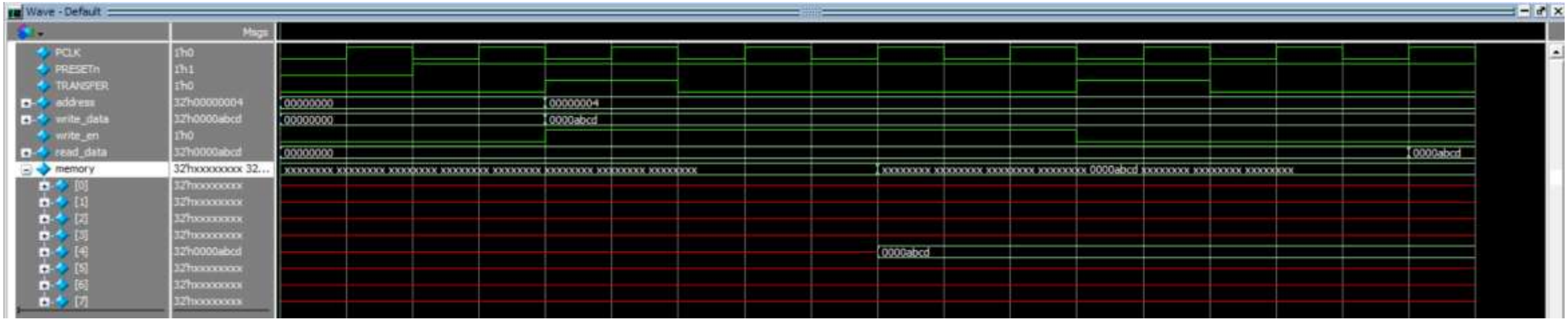
# Do file

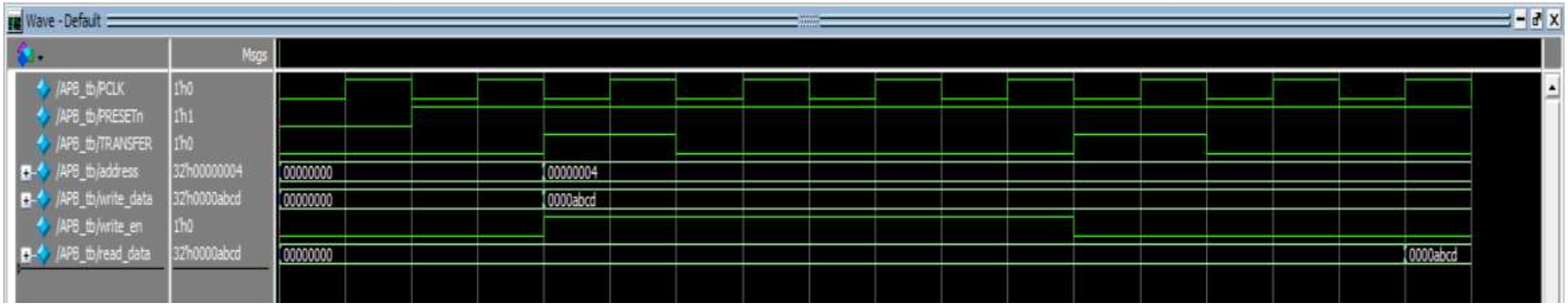
D:\APB\RUN_APB.do - Sublime Text (UNREGISTERED)

APB_Master.v | APB_SLAVE.v | APB_WRAPPER.v

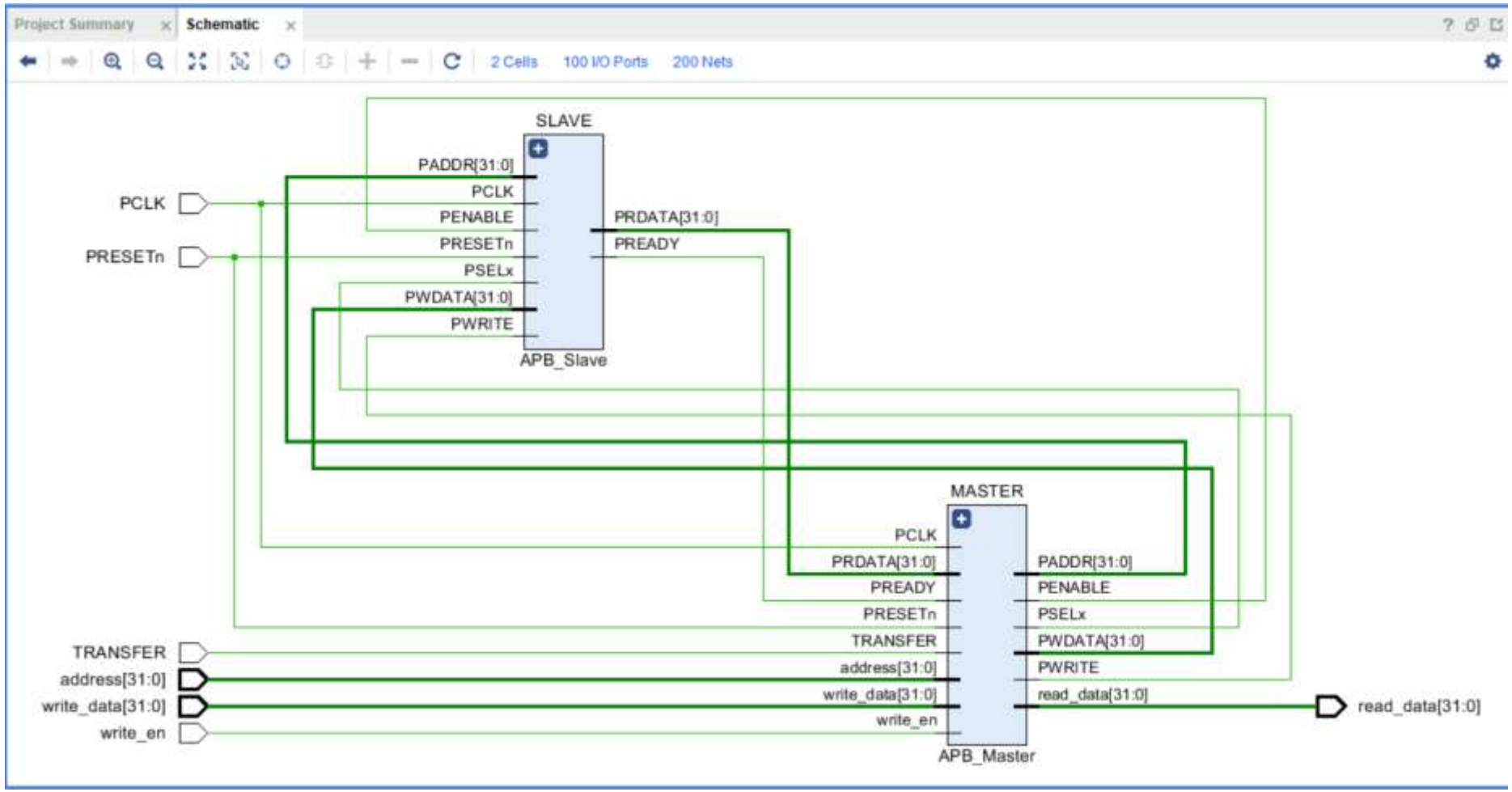
```
vlib work
vlog APB_Master.v APB_SLAVE.v APB_WRAPPER.v APB_tb.v
vsim -voptargs=+acc work.APB_tb
add wave *
run -all
#quit -sim
```

# Wave Form

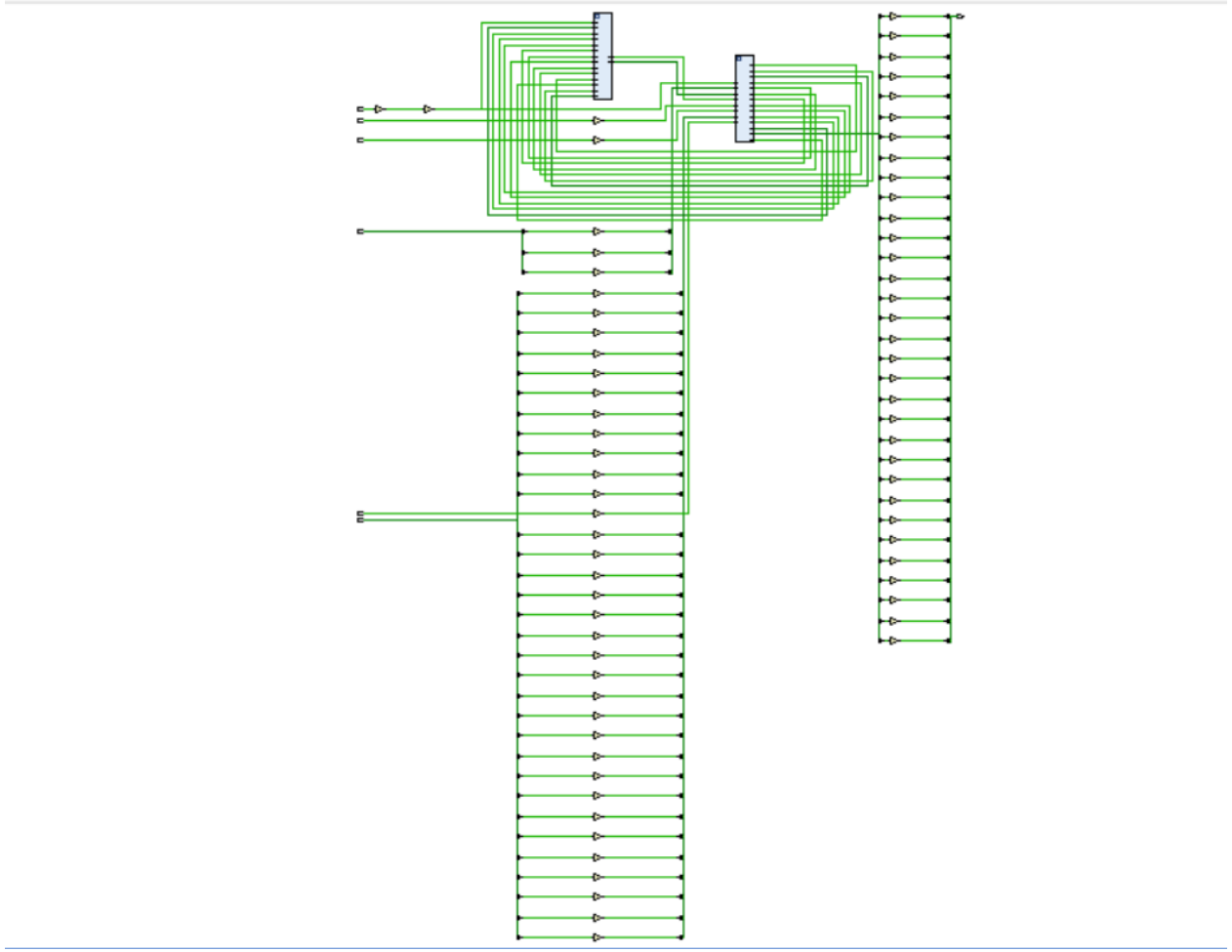

Wave - Default
Msgs
PCLK
PRESETn
TRANSFER
address
write_data
write_en
read_data
memory
[0]
[1]
[2]
[3]
[4]
[5]
[6]
[7]
1'h0
1'h1
1'h0
32'h00000004
32'h0000abcd
1'h0
32'h0000abcd
32'hxxxxxxxx 32...
32'hxxxxxxxx
32'h0000abcd
00000000
00000004
0000abcd
xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
0000abcd


Wave - Default
Msgs
/APB_tb/PCLK
/APB_tb/PRESETn
/APB_tb/TRANSFER
/APB_tb/address
/APB_tb/write_data
/APB_tb/write_en
/APB_tb/read_data
1'h0
1'h1
1'h0
32'h00000004
32'h0000abcd
1'h0
32'h0000abcd
00000000
00000004
0000abcd

# ELABORATED DESIGN



# SYNTHESIZED DESIGN



# REPORT TIMING SUMMARY

**Design Timing Summary**

| Setup | | Hold | | Pulse Width | |
|---|---|---|---|---|---|
| Worst Negative Slack (WNS): | 6.921 ns | Worst Hold Slack (WHS): | 0.144 ns | Worst Pulse Width Slack (WPWS): | 4.500 ns |
| Total Negative Slack (TNS): | 0.000 ns | Total Hold Slack (THS): | 0.000 ns | Total Pulse Width Negative Slack (TPWS): | 0.000 ns |
| Number of Failing Endpoints: | 0 | Number of Failing Endpoints: | 0 | Number of Failing Endpoints: | 0 |
| Total Number of Endpoints: | 682 | Total Number of Endpoints: | 682 | Total Number of Endpoints: | 363 |

**All user specified timing constraints are met.**

# Tools

- Sublime
- Questa Sim
- Xilinx Vivado

# Sources

- AMBA® APBProtocol:

https://www.eecs.umich.edu/courses/eecs373/readings/IHI0024C_amba_apb_protocol_spec.pdf
https://documentation-service.arm.com/static/60d5b505677cf7536a55c245?token=