Richard Ojo

Lab 4 report

## APB slave

```
/*APB Slave module*/
module apb(input PCLK, input PRESET, input [3:0] PWAIT, input PSEL, input PENABLE, input [7:0] PADDR, input PWRITE,
                    input [31:0] PWDATA, input [31:0] MEMRDATA, output logic PREADY, output logic [31:0] PRDATA,
output logic [7:0] addr, output logic [31:0]  data, output logic ce, output logic wren, output logic rden);
//Counter vars
logic [7:0] count;
logic [7:0] next_count;
// Store the current and next state
enum logic [1:0]{
        IDLE = 2'b00,
        ACCESS = 2'b01,
        WAIT = 2'b10,
        WAIT2 = 2'b11
} next_state, curr_state;

task clearAPB;//Reset APB
        begin
                assign PREADY = 0;
                assign PRDATA = 0;
                assign addr = 0;
                assign ce = 0;
                assign wren = 0;
                assign rden = 0;
                assign data = 0;//Write Data to Memory
                next_count = PWAIT;//If we have a wait value, wait those additional clock cycles until access
        end
endtask: clearAPB

task readyToRW(input read_on, input write_on);//Push values for reading and writting to mem
        begin
                assign wren = write_on;
                assign rden = read_on;
                assign addr = PADDR;
                assign ce = 1;
        end

endtask:readyToRW
//Current State logic
always_ff@(posedge PCLK or negedge PRESET) begin
        if(PRESET)  begin
                curr_state <= IDLE; //If reset -> go to the first state
                count <= 0;
                end
        else begin
                curr_state <= next_state; //go to the next state
                count <= next_count;
                end
end
//Next State Logic
always_comb begin
        case(curr_state)
                IDLE: begin
                        clearAPB;

                        if(PSEL) begin
                                readyToRW(!PWRITE, PWRITE);//Prepare memory to be read or written to
                                if(!PWAIT)
                                        next_state = ACCESS;// ACESS STATE
                                else if(PWAIT)
                                        next_state = WAIT;// WAIT STATE
                                else if(!PENABLE)
                                        next_state = next_state.first;//IDLE STATE
                                end
                        else
                                next_state = next_state.first;//IDLE STATE
                end

                ACCESS: begin
```

```
                    if(!ce) begin
                        readyToRW(!PWRITE, PWRITE); //IF we are performing another transfer
                    end
                    if(!PSEL && !PENABLE) begin
                      clearAPB;
                        next_state = next_state.first;//IDLE STATE
                    end
                    else if(next_count > 4'b0001) begin
                        next_state = WAIT;//Wait value is greater than 1
                    end
                    else if(PSEL && PENABLE && PWRITE && next_count <= 4'b0001) begin // Ready to Write
                        assign PREADY = 1;
                        assign PRDATA = 0;//Read Data from Memory
                        assign data = PWDATA;//Write Data to Memory

                        end
                    else if(PSEL && PENABLE && PWRITE == 1'b0 && next_count <= 4'b0001) begin // Ready to Read
                        assign PREADY = 1;
                        assign PRDATA = MEMRDATA;//Read Data from Memory
                        assign data = 0;//Write Data to Memory

                        end
                    else begin //PREADY == 1 and another transfer
                        next_state = ACCESS;
                        clearAPB;
                        end
                    end

                WAIT: begin
                    if(!PENABLE) begin
                        next_state = WAIT;//Keep waiting
                    end
                    else if(PENABLE) begin
                        next_count = next_count - 1;//1 Clock Cycle Passed
                        if(next_count == 0) begin
                            next_state = ACCESS; //All Clock Cycles done so time to read/write
                        end
                        else begin
                            next_state = WAIT2;//Keep waiting
                        end
                    end
                end

                WAIT2: begin//This is a 2nd wait state for waits > 2 cycles
                    if(!PENABLE) begin
                        next_state = WAIT;//Keep waiting
                    end
                    else if(PENABLE) begin
                        next_count = next_count - 1;//1 Clock Cycle Passed
                        if(next_count == 0) begin
                            next_state = ACCESS; //All Clock Cycles done so time to read/write
                        end
                        else begin
                            next_state = WAIT;//Keep waiting
                        end
                    end
                end
            endcase
        end
        endmodule: apb
```

## APB slave and Memory

```
module apbMem(input logic PCLK, input logic PRESET,input logic [3:0] PWAIT, input logic PSEL,
input logic PENABLE, input logic [7:0] PADDR, input logic PWRITE,
                    input logic [31:0] PWDATA, output logic PREADY, output logic [31:0] PRDATA);
logic ce, wren, rden;
logic [7:0] addr;
logic [31:0] data, MEMRDATA;
apb bus(.*);
memory_c mem(addr, ce, wren, rden, data, MEMRDATA, PCLK);
endmodule: apbMem
```

## APB slave and Memory Interface

```
interface apbMemInterface(input logic PCLK);

logic PRESET, PSEL, PENABLE, PWRITE, PREADY;
logic [3:0] PWAIT;
logic [7:0] PADDR;
logic [31:0] PWDATA, PRDATA;

task mem_reset;
        PRESET = 1; #10; PRESET = 0; #10;// Apply reset wait
endtask: mem_reset

task mem_write; //Address to write to, write data, and wait time(cycles)
        input [7:0] addr_w;
        input [31:0] data_w;
        input [3:0] wait_w;

        @(posedge PCLK) begin
                PENABLE = 0;
                PSEL = 0;
                PADDR = addr_w;
                PWDATA =  data_w;
                PWRITE = 1'b1;
                PWAIT = wait_w;
                #10;
                if(!PSEL) begin
                        PSEL = 1; #10;
                end
                PENABLE = 1;
                for(int i = 0; i <= wait_w+1; i++) begin //Wait as long as it takes for PREADY == 1
                        #10;
                end
        end
endtask: mem_write

task mem_read;//Address to read from, and wait time(cycles)
        input [7:0] addr_r;
        input [3:0] wait_r;

        @(posedge PCLK) begin
                PENABLE = 0;
                PSEL = 0;
                PADDR = addr_r;
                PWRITE = 1'b0;
```

```
			PWAIT = wait_r;
			#10;
			if(PSEL == 1'b0) begin
				PSEL = 1; #10;
			end
			PENABLE = 1;
			for(int i = 0; i <= wait_r+1; i++) begin //Wait as long as it takes for PREADY == 1
				#10;
			end
		end
endtask: mem_read
endinterface: apbMemInterface
```

## Test Bench

```
module testbench;
//Instantiate Clock
logic PCLK = 1'b0;

// Instantiate interface
apbMemInterface intf(PCLK);

// Instantiate device under test
apbMem dut (PCLK, intf.PRESET, intf.PWAIT, intf.PSEL, intf.PENABLE, intf.PADDR, intf.PWRITE,
			intf.PWDATA, intf.PREADY, intf.PRDATA);

// generate clock
always begin
		PCLK = 1; #5; PCLK = 0; #5; // 10ns period
end

logic [1:0] wait_val;
logic [7:0] addr_val;
int data_val, counter = 1;

initial begin
		intf.mem_reset;
		//Demonstration test for read and write
					//addr_w, data_w, wait_w
		intf.mem_write(8'h11, 8'hAA, 4'b0000);
					//addr_r, wait_r
		intf.mem_read(8'h11, 4'b0000);
		intf.mem_read(8'h10, 4'b0000);//Should result in invalid


		for (int i = 0; i < 15; i++) begin

			std::randomize(wait_val); std::randomize(addr_val); std::randomize(data_val); // Generate random values to test with

			intf.mem_write(addr_val, data_val, wait_val);// Write Value to mem addr

			intf.mem_read(addr_val, wait_val);// Read value from same addr in mem

			$display("Test %d: wait = %h, addr = %h, data = %h", counter, wait_val, addr_val, data_val);
			$display("PRDATA = %h", intf.PRDATA);

			if(intf.PRDATA == data_val) begin
				$display("Success, the data values are equal!");
			end
			counter++;
		end


/*		intf.mem_write(8'h10, 8'hAC, 4'b0000);
		intf.mem_write(8'h11, 8'hAA, 4'b0000);
		intf.mem_write(8'h12, 8'hAB, 4'b0000);
```

```
          intf.mem_write(8'h13, 8'hAC, 4'b0000);
          intf.mem_write(8'h14, 8'hAD, 4'b0000);
          intf.mem_write(8'h15, 8'hAE, 4'b0000);
          intf.mem_write(8'h16, 8'hAF, 4'b0000);
                              //addr_r, wait_r
          intf.mem_read(8'h10, 4'b0001);
          intf.mem_read(8'h11, 4'b0001);
          intf.mem_read(8'h12, 4'h0010);
          intf.mem_read(8'h13, 4'h0011);
          intf.mem_read(8'h14, 4'h0100);
          intf.mem_read(8'h15, 4'h0101);
          intf.mem_read(8'h16, 4'h0110);*/
          $finish;
end
endmodule
```

## Display

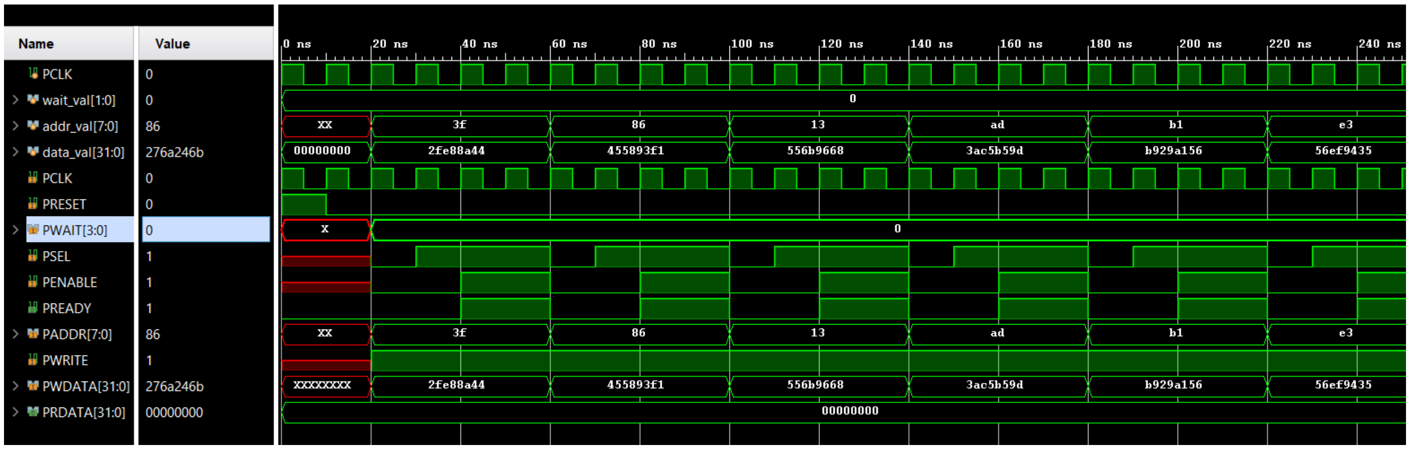
## *This is from a run of the testbench

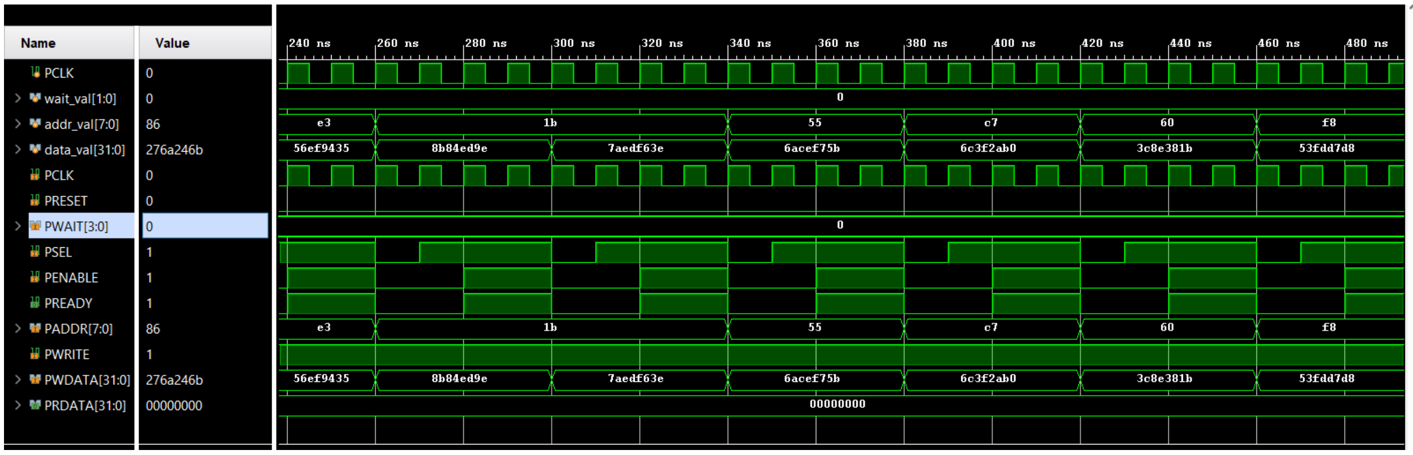
```
Compiling package std.std
Compiling module xil_defaultlib.apbMemInterface
Compiling module xil_defaultlib.apb
Compiling module xil_defaultlib.memory_c
Compiling module xil_defaultlib.apbMem
Compiling module xil_defaultlib.testbench
Compiling module xil_defaultlib.glbl
Built simulation snapshot testbench_behav
INFO: [USF-XSim-69] 'elaborate' step finished in '3' seconds
Vivado Simulator 2019.1
Time resolution is 1 ps
Test          1: wait = 3, addr = 44, data = 14c79a86
PRDATA = 14c79a86
Success, the data values are equal!
Test          2: wait = 1, addr = 13, data = 556b9668
PRDATA = 556b9668
Success, the data values are equal!
Test          3: wait = 1, addr = 9d, data = 6c58bdb1
PRDATA = 6c58bdb1
Success, the data values are equal!
Test          4: wait = 2, addr = e3, data = 56ef9435
PRDATA = 56ef9435
Success, the data values are equal!
Test          5: wait = 3, addr = 9e, data = 6ab0ad1b
PRDATA = 6ab0ad1b
Success, the data values are equal!
Test          6: wait = 2, addr = 55, data = 6acef75b
PRDATA = 6acef75b
Success, the data values are equal!
Test          7: wait = 3, addr = b0, data = 7653e560
PRDATA = 7653e560
Success, the data values are equal!
relaunch_sim: Time (s): cpu = 00:00:03 ; elapsed = 00:00:12 . Memory (MB): peak = 903.527 ; gain = 0.000
run all
Test          8: wait = 3, addr = f8, data = 53fdd7d8
PRDATA = 53fdd7d8
Success, the data values are equal!
Test          9: wait = 3, addr = 24, data = 003d0302
PRDATA = 003d0302
Success, the data values are equal!
Test         10: wait = 2, addr = 86, data = 276a246b
PRDATA = 276a246b
Success, the data values are equal!
Test         11: wait = 1, addr = 9e, data = 397033e4
PRDATA = 397033e4
Success, the data values are equal!
Test         12: wait = 2, addr = 42, data = 88047bb3
PRDATA = 88047bb3
Success, the data values are equal!
Test         13: wait = 3, addr = 51, data = 56f156a1
PRDATA = 56f156a1
Success, the data values are equal!
Test         14: wait = 1, addr = 9c, data = 838f4741
PRDATA = 838f4741
Success, the data values are equal!
Test         15: wait = 0, addr = a4, data = 13d2ebfd
PRDATA = 13d2ebfd
Success, the data values are equal!
$finish called at time : 1940 ns : File "C:/Users/rdipn/Desktop/ENEE459D/Lab34/Lab4/lab4.srcs/sources_1/imports/examples/testbench.sv" Line
83
```

## Wave Forms

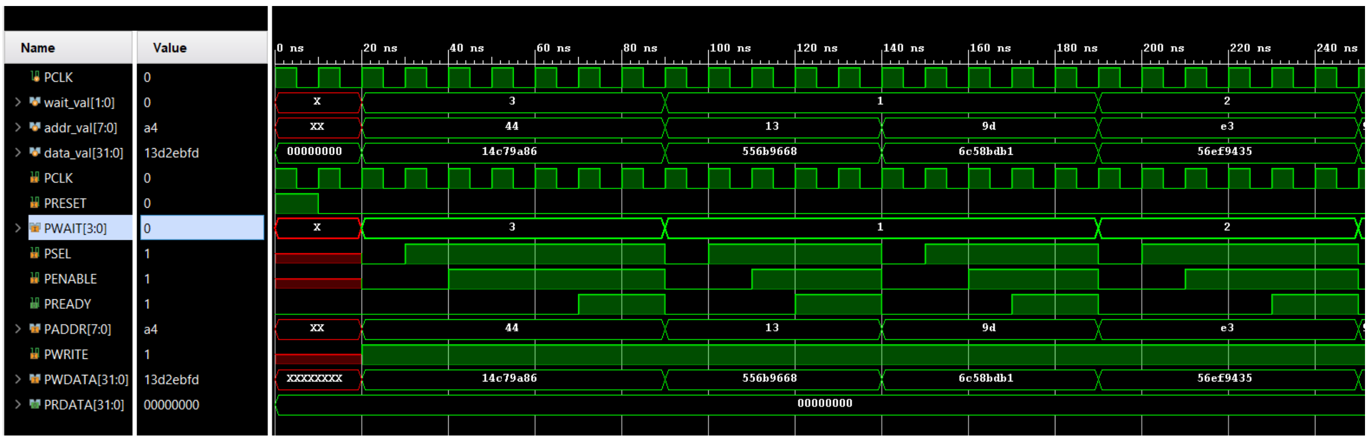
## *Wave forms are from 2 runs (1 with wait states and 1 without)

## Write transfer with no wait states

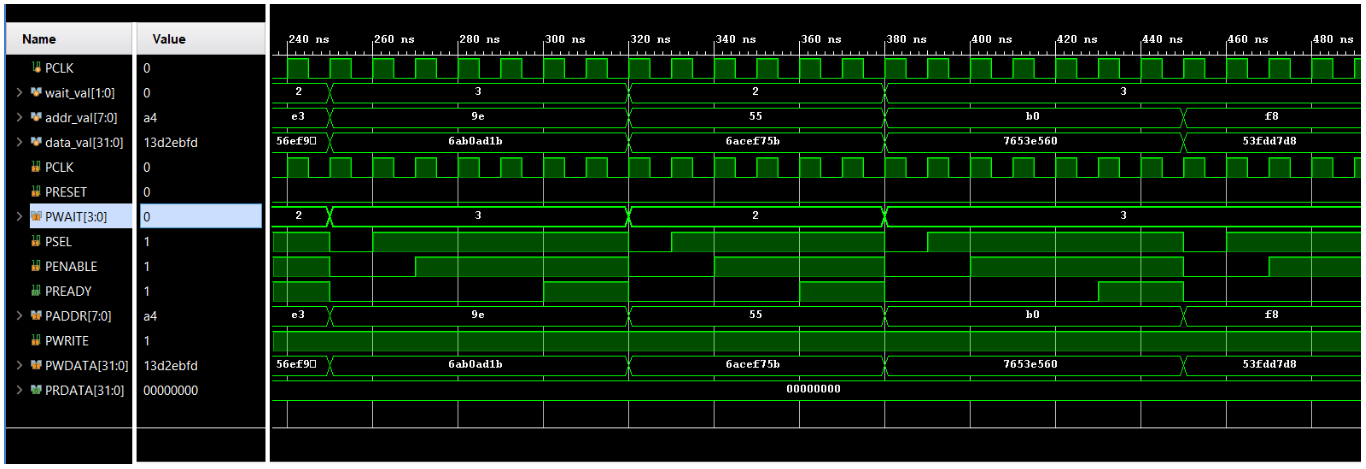




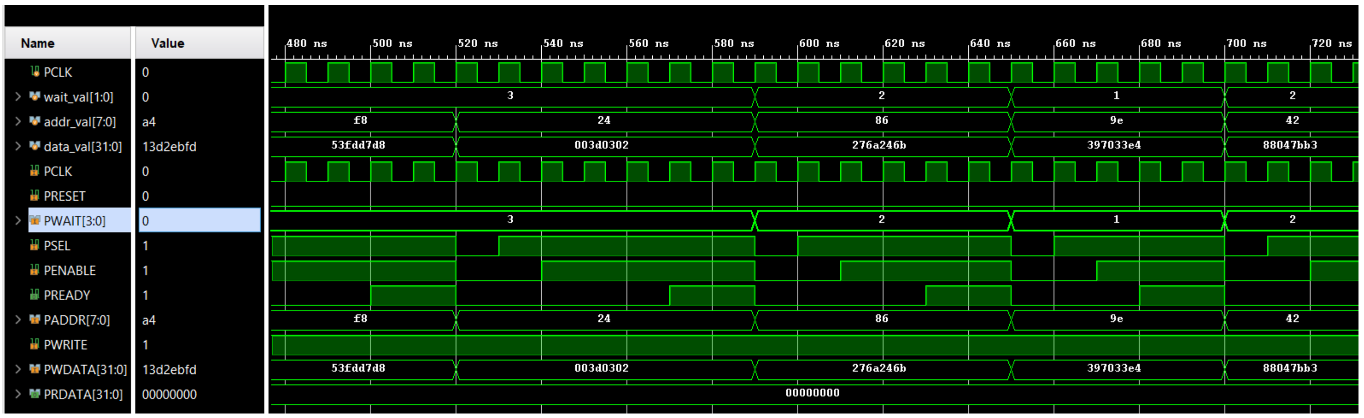
# Write transfer with wait states



| Name | Value |
|---|---|
| PCLK | 0 |
| wait_val[1:0] | 0 |
| addr_val[7:0] | a4 |
| data_val[31:0] | 13d2ebfd |
| PCLK | 0 |
| PRESET | 0 |
| PWAIT[3:0] | 0 |
| PSEL | 1 |
| PENABLE | 1 |
| PREADY | 1 |
| PADDR[7:0] | a4 |
| PWRITE | 1 |
| PWDATA[31:0] | 13d2ebfd |
| PRDATA[31:0] | 00000000 |

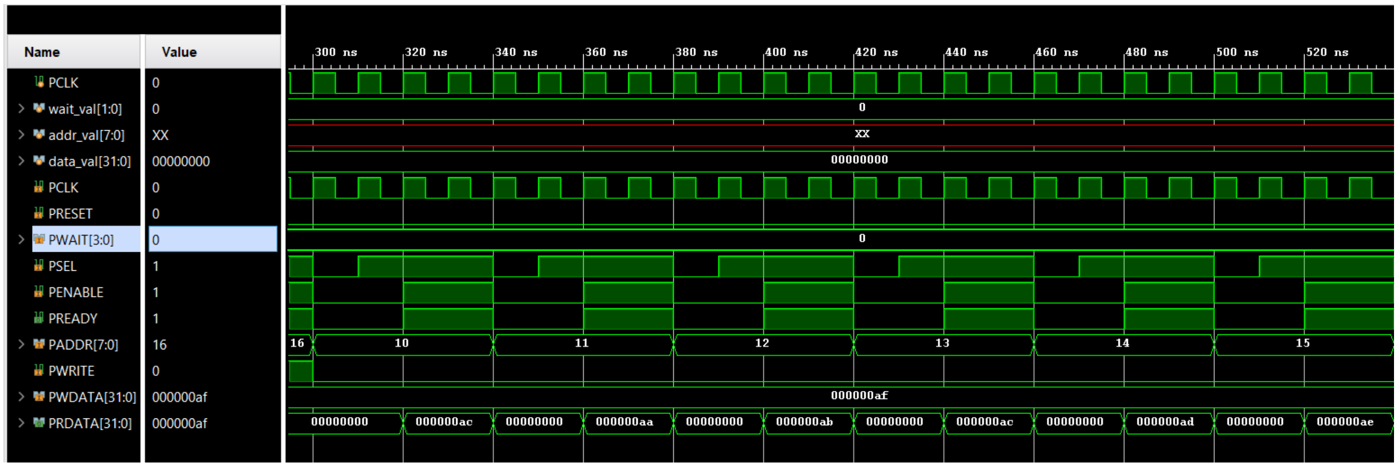


| Name | Value |
|---|---|
| PCLK | 0 |
| wait_val[1:0] | 0 |
| addr_val[7:0] | a4 |
| data_val[31:0] | 13d2ebfd |
| PCLK | 0 |
| PRESET | 0 |
| PWAIT[3:0] | 0 |
| PSEL | 1 |
| PENABLE | 1 |
| PREADY | 1 |
| PADDR[7:0] | a4 |
| PWRITE | 1 |
| PWDATA[31:0] | 13d2ebfd |
| PRDATA[31:0] | 00000000 |

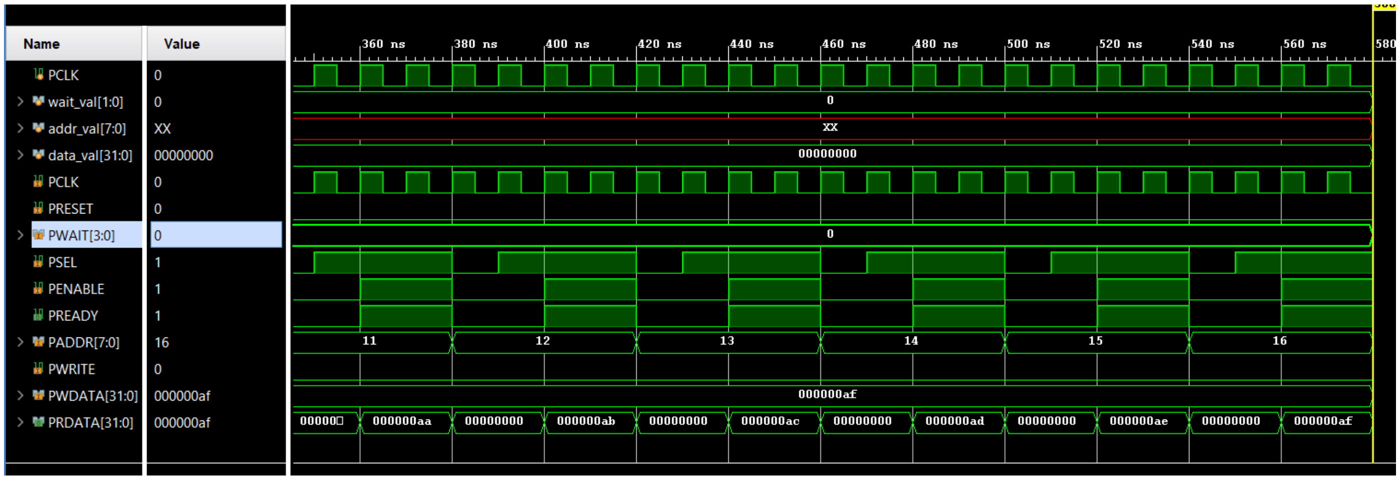


| Name | Value |
|---|---|
| PCLK | 0 |
| wait_val[1:0] | 0 |
| addr_val[7:0] | a4 |
| data_val[31:0] | 13d2ebfd |
| PCLK | 0 |
| PRESET | 0 |
| PWAIT[3:0] | 0 |
| PSEL | 1 |
| PENABLE | 1 |
| PREADY | 1 |
| PADDR[7:0] | a4 |
| PWRITE | 1 |
| PWDATA[31:0] | 13d2ebfd |
| PRDATA[31:0] | 00000000 |

## Read transfer with no wait states

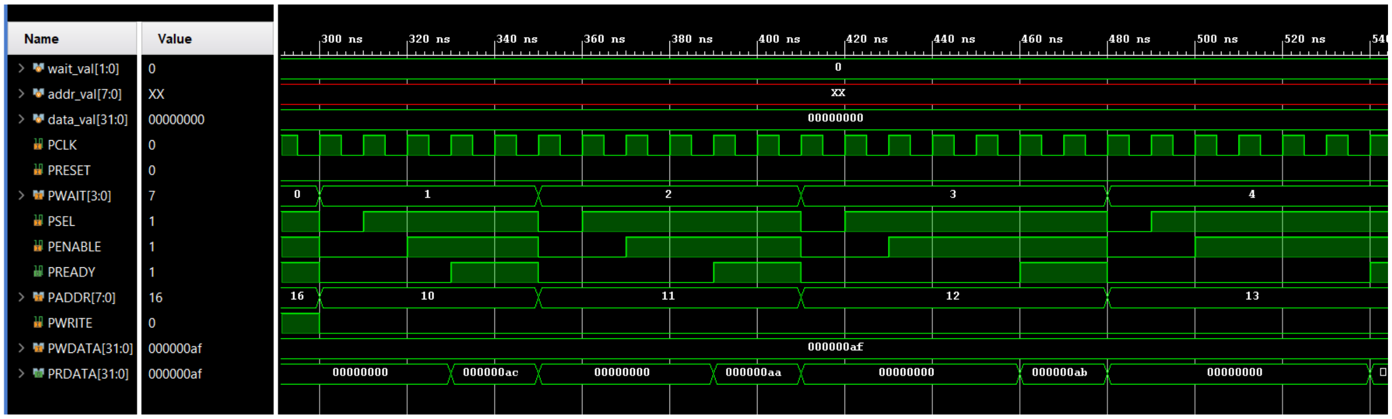


| Name | Value |
|---|---|
| PCLK | 0 |
| wait_val[1:0] | 0 |
| addr_val[7:0] | XX |
| data_val[31:0] | 00000000 |
| PCLK | 0 |
| PRESET | 0 |
| PWAIT[3:0] | 0 |
| PSEL | 1 |
| PENABLE | 1 |
| PREADY | 1 |
| PADDR[7:0] | 16 |
| PWRITE | 0 |
| PWDATA[31:0] | 000000af |
| PRDATA[31:0] | 000000af |

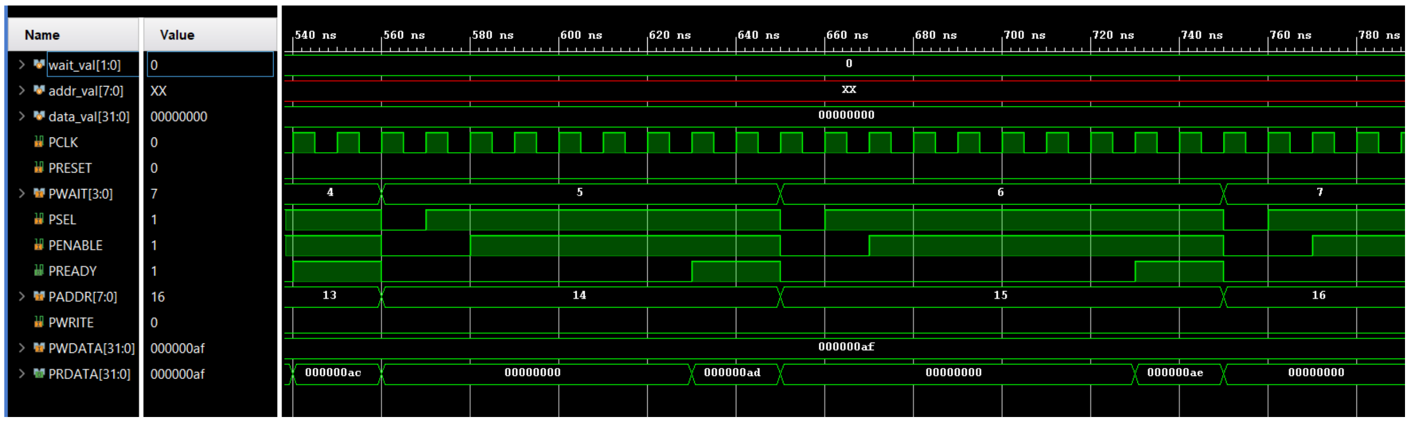


| Name | Value |
|---|---|
| PCLK | 0 |
| wait_val[1:0] | 0 |
| addr_val[7:0] | XX |
| data_val[31:0] | 00000000 |
| PCLK | 0 |
| PRESET | 0 |
| PWAIT[3:0] | 0 |
| PSEL | 1 |
| PENABLE | 1 |
| PREADY | 1 |
| PADDR[7:0] | 16 |
| PWRITE | 0 |
| PWDATA[31:0] | 000000af |
| PRDATA[31:0] | 000000af |

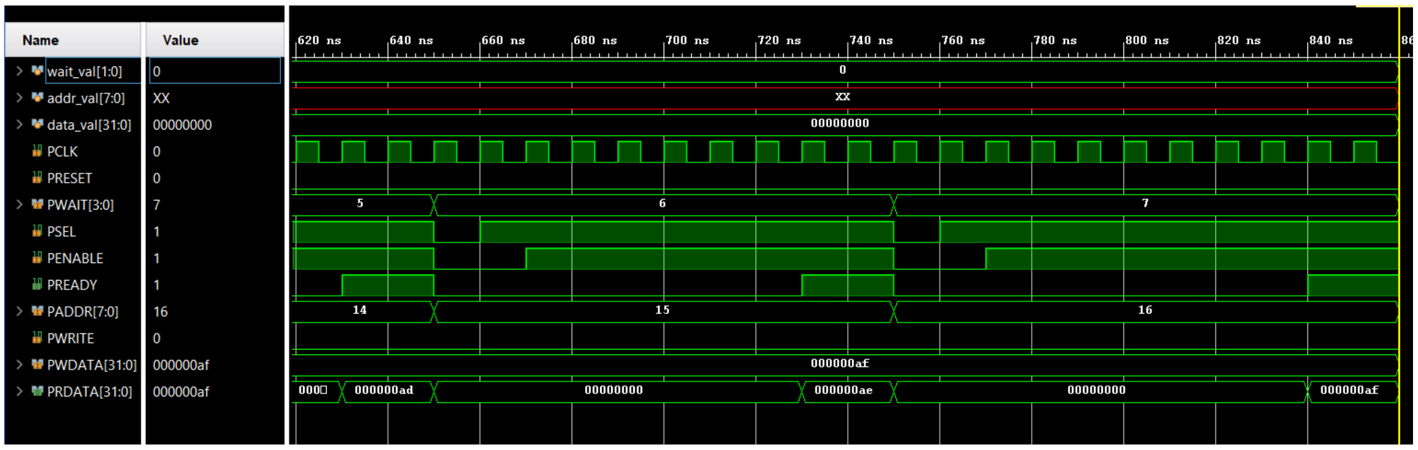
## Read transfer with wait states



| Name | Value |
|---|---|
| wait_val[1:0] | 0 |
| addr_val[7:0] | XX |
| data_val[31:0] | 00000000 |
| PCLK | 0 |
| PRESET | 0 |
| PWAIT[3:0] | 7 |
| PSEL | 1 |
| PENABLE | 1 |
| PREADY | 1 |
| PADDR[7:0] | 16 |
| PWRITE | 0 |
| PWDATA[31:0] | 000000af |
| PRDATA[31:0] | 000000af |



| Name | Value |
|---|---|
| wait_val[1:0] | 0 |
| addr_val[7:0] | XX |
| data_val[31:0] | 00000000 |
| PCLK | 0 |
| PRESET | 0 |
| PWAIT[3:0] | 7 |
| PSEL | 1 |
| PENABLE | 1 |
| PREADY | 1 |
| PADDR[7:0] | 16 |
| PWRITE | 0 |
| PWDATA[31:0] | 000000af |
| PRDATA[31:0] | 000000af |



| Name | Value |
|---|---|
| wait_val[1:0] | 0 |
| addr_val[7:0] | XX |
| data_val[31:0] | 00000000 |
| PCLK | 0 |
| PRESET | 0 |
| PWAIT[3:0] | 7 |
| PSEL | 1 |
| PENABLE | 1 |
| PREADY | 1 |
| PADDR[7:0] | 16 |
| PWRITE | 0 |
| PWDATA[31:0] | 000000af |
| PRDATA[31:0] | 000000af |