EE533_Lab8_Report

1. Instruction Part

1.1 Purpose

- Bubble sort the packet payload data
  - Each component in payload is 64-bit wide.
  - First Line would be the array size.
  - Following elements are the array elements.

1.2 Definition

- Instruction opcode lookup table

Instr	OP Code [31:26]
noop	000000
addi	000001
movi	000010
lw	000011
sw	000100
beq	000101
bgt	000110
blt	000111
j	001000
subi	001001

- Instruction Table

Addr	Label	Instr	OP Code [31:26]	Rs [25:21]	Rt [20:16]	Offset [15:0]
0		lw r1, r0(#0)	000011	5'd0	5'd1	16'd0
1		movi r2, #0	000010	5'd0	5'd2	16'd0
2		movi r3, #1	000010	5'd0	5'd3	16'd1
3		noop	000000	5'd0	5'd0	16'd0
4		noop	000000	5'd0	5'd0	16'd0
5	outer_loop	beq r1, r0, end	000101	5'd0	5'd1	16'd27

Addr	Label	Instr	OP Code [31:26]	Rs [25:21]	Rt [20:16]	Offset [15:0]
6	inner_loop	noop	000000	5'd0	5'd0	16'd0
7		bgt r3, r1, next_out	000110	5'd1	5'd3	16'd21
8		noop	000000	5'd0	5'd0	16'd0
9		lw r4, r2(#1)	000011	5'd2	5'd4	16'd1
10		lw r5, r3(#1)	000011	5'd3	5'd5	16'd1
11		noop	000000	5'd0	5'd0	16'd0
12		noop	000000	5'd0	5'd0	16'd0
13		noop	000000	5'd0	5'd0	16'd0
14		blt r4, r5, no_swap	000111	5'd5	5'd4	16'd18
15		noop	000000	5'd0	5'd0	16'd0
16		sw r4, r3(#1)	000100	5'd3	5'd4	16'd1
17		sw r5, r2(#1)	000100	5'd2	5'd5	16'd1
18	no_swap	addi r2, r2, #1	000001	5'd2	5'd2	16'd1
19		addi r3, r3, #1	000001	5'd3	5'd3	16'd1
20		j inner_loop	001000	5'd0	5'd0	16'd6
21		noop	000000	5'd0	5'd0	16'd0
22	next_out	subi r1, r1, #1	001001	5'd1	5'd1	16'd1
23		movi r2, #0	000010	5'd0	5'd2	16'd0
24		movi r3, #1	000010	5'd0	5'd3	16'd1
25		j outer_loop	001000	5'd0	5'd0	16'd5
26		noop	000000	5'd0	5'd0	16'd0
27	end	j end	001000	5'd0	5'd0	16'd27

- Instruction Memory Initialization FIle

```
000011 00000 00001 0000000000000000 = 0000 1100 0000 0001 0000 0000 0000 0000 =
0c010000
000010 00000 00010 0000000000000000 = 0000 1000 0000 0010 0000 0000 0000 0000 =
08020000
000010 00000 00011 0000000000000001 = 0000 1000 0000 0011 0000 0000 0000 0001 =
08030001
```

```
000000 00000 00000 0000000000000000 = 0000 0000 0000 0000 0000 0000 0000 0000 =
00000000
000000 00000 00000 0000000000000000 = 0000 0000 0000 0000 0000 0000 0000 0000 =
00000000
000101 00000 00001 0000000000011011 = 0001 0100 0000 0001 0000 0000 0001 1011 =
1401001B
000000 00000 00000 0000000000000000 = 0000 0000 0000 0000 0000 0000 0000 0000 =
00000000
000110 00001 00011 0000000000010101 = 0001 1000 0010 0011 0000 0000 0001 0101 =
18230015
000000 00000 00000 0000000000000000 = 0000 0000 0000 0000 0000 0000 0000 0000 =
00000000
000011 00010 00100 0000000000000001 = 0000 1100 0100 0100 0000 0000 0000 0001 =
0C440001
000011 00011 00101 0000000000000001 = 0000 1100 0110 0101 0000 0000 0000 0001 =
0C650001
000000 00000 00000 0000000000000000 = 0000 0000 0000 0000 0000 0000 0000 0000 =
00000000
000000 00000 00000 0000000000000000 = 0000 0000 0000 0000 0000 0000 0000 0000 =
00000000
000000 00000 00000 0000000000000000 = 0000 0000 0000 0000 0000 0000 0000 0000 =
00000000
000111 00101 00100 0000000000010010 = 0001 1100 1010 0100 0000 0000 0001 0010 =
1CA40012
000000 00000 00000 0000000000000000 = 0000 0000 0000 0000 0000 0000 0000 0000 =
00000000
000100 00011 00100 0000000000000001 = 0001 0000 0110 0100 0000 0000 0000 0001 =
10640001
000100 00010 00101 0000000000000001 = 0001 0000 0100 0101 0000 0000 0000 0001 =
10450001
000001 00010 00010 0000000000000001 = 0000 0100 0100 0010 0000 0000 0000 0001 =
04420001
000001 00011 00011 0000000000000001 = 0000 0100 0110 0011 0000 0000 0000 0001 =
04630001
001000 00000 00000 0000000000000110 = 0010 0000 0000 0000 0000 0000 0000 0111 =
20000006
000000 00000 00000 0000000000000000 = 0000 0000 0000 0000 0000 0000 0000 0000 =
00000000
001001 00001 00001 0000000000000001 = 0010 0100 0010 0001 0000 0000 0000 0001 =
24210001
000010 00000 00010 0000000000000000 = 0000 1000 0000 0010 0000 0000 0000 0000 =
08020000
000010 00000 00011 0000000000000001 = 0000 1000 0000 0011 0000 0000 0000 0001 =
08030001
001000 00000 00000 0000000000000101 = 0010 0000 0000 0000 0000 0000 0000 0101 =
20000005
000000 00000 00000 0000000000000000 = 0000 0000 0000 0000 0000 0000 0000 0000 =
00000000
001000 00000 00000 0000000000011011 = 0010 0000 0000 0000 0000 0000 0001 1011 =
2000001B
```

2. Packet Part

2.1 Packet Header Format Design

VER	HLEN	Service	Total Length	Identification	Flags	Fragmentation Offset
4 bit	4 bit	8 bits	16 bits	16 bits	3 bits	13 bits
TTL		Protocol	Header Checksum	Source IP Address		
8 bits		8 bits	16 bits	32 bits		
Destination IP Address				Option		
32 bits				32 bits		

2.2 Sample Initial Packet

VER	HLEN	Service	Total Length	Identification	Flags	Fragmentation Offset
4 bit	4 bit	8 bits	16 bits	16 bits	3 bits	13 bits
IPv4	24 Byte	0	64 Bytes	1C 46	(0, D, M)	0
0100	0110	0000 0000	0000 0000 0100 0000	0001 1100 0100 0110	010	0 0000 0000 0000
TTL		Protocol	Header Checksum	Source IP Address		
8 bits		8 bits	16 bits	32 bits		
4		TCP	2A6D	10.0.13.3 (node 1)		
0000 0100		0000 0110	0010 1010 0110 1101	0000 1010 0000 0000 0000 1101 0000 0011		
Destination IP Address				Option		
32 bits				32 bits		
10.0.14.3 (node 2)				0		
0000 1010 0000 0000 0000 1110 0000 0011				0000 0000 0000 0000 0000 0000 0000 0000		
Payload						
64 bits						
3						
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0100						
4						
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0100						
3						
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0011						
2						
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0010						
1						
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0001						

2.3 Packet Expected to get after Bubble Sort by Pipeline Processor

VER	HLEN	Service	Total Length	Identification	Flags	Fragmentation Offset
4 bit	4 bit	8 bits	16 bits	16 bits	3 bits	13 bits
IPv4	24 Byte	0	64 Bytes	1C 46	(0, D, M)	0
0100	0110	0000 0000	0000 0000 0100 0000	0001 1100 0100 0110	010	0 0000 0000 0000
TTL		Protocol	Header Checksum	Source IP Address		
8 bits		8 bits	16 bits	32 bits		
4		TCP	2A6D	10.0.13.3 (node 1)		
0000 0100		0000 0110	0010 1010 0110 1101	0000 1010 0000 0000 0000 1101 0000 0011		
Destination IP Address				Option		
32 bits				32 bits		
10.0.14.3 (node 2)				0		
0000 1010 0000 0000 0000 1110 0000 0011				0000 0000 0000 0000 0000 0000 0000 0000		
Payload						
64 bits						
3						
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0100						
1						
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0001						
2						
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0010						
3						
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0011						
4						
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0100						

2.4 Bubble Sort Expected process

loop	1	2	3	4
1	4	3	2	1
2	**3**	**4**	2	1
3	3	**2**	**4**	1
4	3	2	**1**	**4**
5	**2**	**3**	1	4

loop	1	2	3	4
6	2	**1**	**3**	4
7	**1**	**2**	3	4

3. Pipeline Updated Part

3.1 D_MEM Mode Definition

Mode Name	Mode Code	Description
FIFO_IN	00	BRAM working as FIFO and write in packet, WP <= WP + 1
FIFO_OUT	01	BRAM working as FIFO and read out packet, RP <= RP + 1
SRAM_PROCESSING	10	BRAM working as D_MEM in pipeline processor

3.2 RP (as Head Address)

3.2.1 RP_Reg

- Verilog

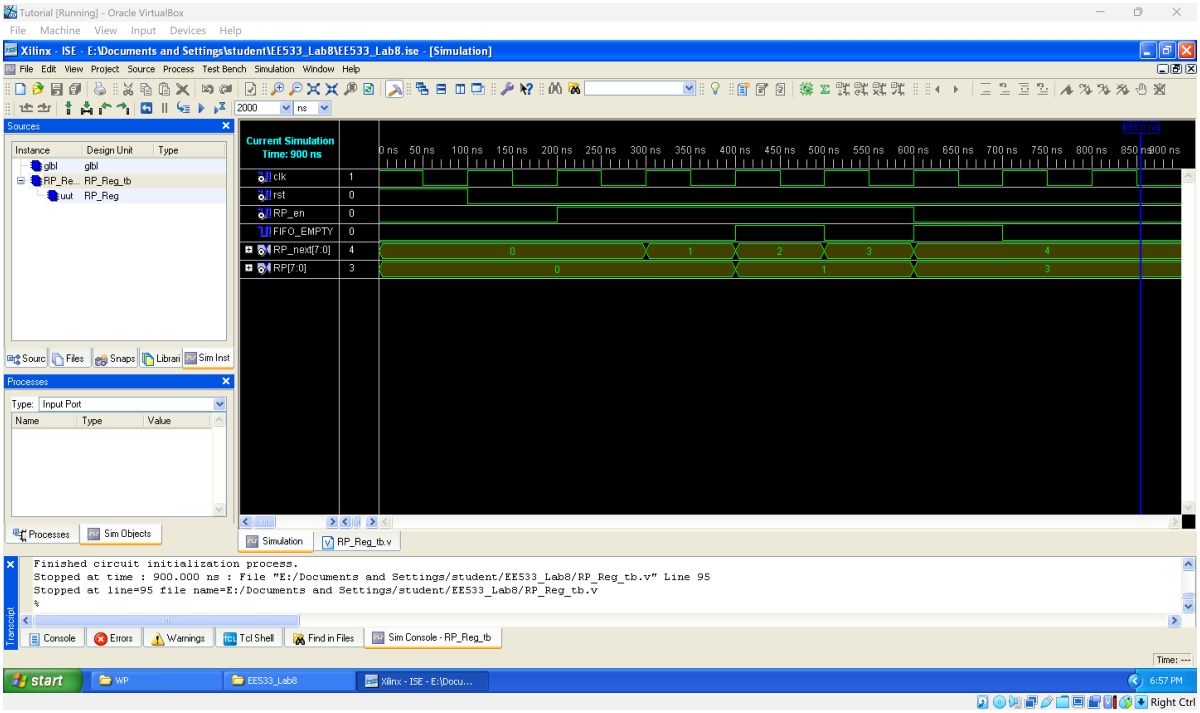
```
`timescale 1ns / 1ps

module RP_Reg
(
    input clk,
    input rst,
    input RP_en,
    input FIFO_EMPTY,
    input [7:0] RP_next,

    output reg [7:0] RP
);

    always @(posedge clk) begin
        if (rst) begin
            RP <= 0;
        end
        else if(RP_en && !FIFO_EMPTY) begin
            RP <= RP_next;
        end
    end

endmodule
```

- Testbench

```v
`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   13:59:37 03/08/2025
// Design Name:   RP_Reg
// Module Name:   E:/Documents and Settings/student/EE533_Lab8/RP_Reg_tb.v
// Project Name:  EE533_Lab8
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: RP_Reg
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module RP_Reg_tb;

    // Inputs
    reg clk;
    reg rst;
    reg RP_en;
    reg FIFO_EMPTY;
    reg [7:0] RP_next;

    // Outputs
    wire [7:0] RP;

    // Instantiate the Unit Under Test (UUT)
    RP_Reg uut (
        .clk(clk),
        .rst(rst),
        .RP_en(RP_en),
        .FIFO_EMPTY(FIFO_EMPTY),
        .RP_next(RP_next),
        .RP(RP)
    );

    always #50 clk = ~clk;

    initial begin
        // Initialize Inputs
        clk = 1;
        rst = 1;
        RP_en = 0;
        FIFO_EMPTY = 0;
        RP_next = 0;
```

```
        // Wait 100 ns for global reset to finish
        @(posedge clk);
        rst = 0;

        // Add stimulus here
        @(posedge clk);
        RP_en = 1;
        FIFO_EMPTY = 0;
        RP_next = 8'd0;

        @(posedge clk);
        RP_en = 1;
        FIFO_EMPTY = 0;
        RP_next = 8'd1;

        @(posedge clk);
        RP_en = 1;
        FIFO_EMPTY = 1;
        RP_next = 8'd2;

        @(posedge clk);
        RP_en = 1;
        FIFO_EMPTY = 0;
        RP_next = 8'd3;

        @(posedge clk);
        RP_en = 0;
        FIFO_EMPTY = 1;
        RP_next = 8'd4;

        @(posedge clk);
        RP_en = 0;
        FIFO_EMPTY = 0;
        RP_next = 8'd4;

        @(posedge clk);

        @(posedge clk);
        $stop;

    end

endmodule
```

- Waveform



3.2.2 RP_Adder

- Verilog

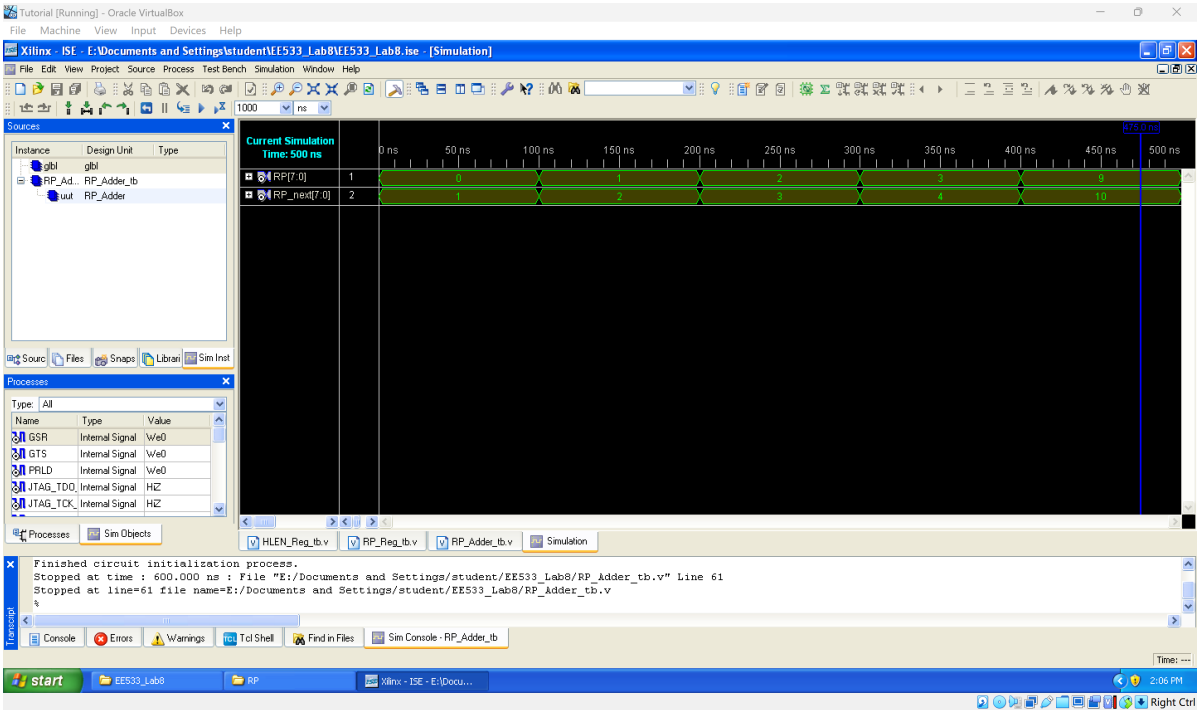
```
`timescale 1ns / 1ps

module RP_Adder
(
    input [7:0] RP,

    output [7:0] RP_next
);

    assign RP_next = RP + 1;

endmodule
```

- Testbench

```
`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   14:04:08 03/08/2025
// Design Name:   RP_Adder
// Module Name:   E:/Documents and Settings/student/EE533_Lab8/RP_Adder_tb.v
// Project Name:  EE533_Lab8
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: RP_Adder
//
```

```
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

module RP_Adder_tb;

    // Inputs
    reg [7:0] RP;

    // Outputs
    wire [7:0] RP_next;

    // Instantiate the Unit Under Test (UUT)
    RP_Adder uut (
        .RP(RP),
        .RP_next(RP_next)
    );

    initial begin
        // Initialize Inputs
        RP = 0;

        // Wait 100 ns for global reset to finish
        #100;
        RP = 8'd1;

        // Add stimulus here
        #100;
        RP = 8'd2;

        #100;
        RP = 8'd3;

        #100;
        RP = 8'd9;

        #100;
        RP = 8'd1;

        #100;
        $stop;

    end

endmodule
```

- Waveform



3.2.3 RP_Controller

- Verilog

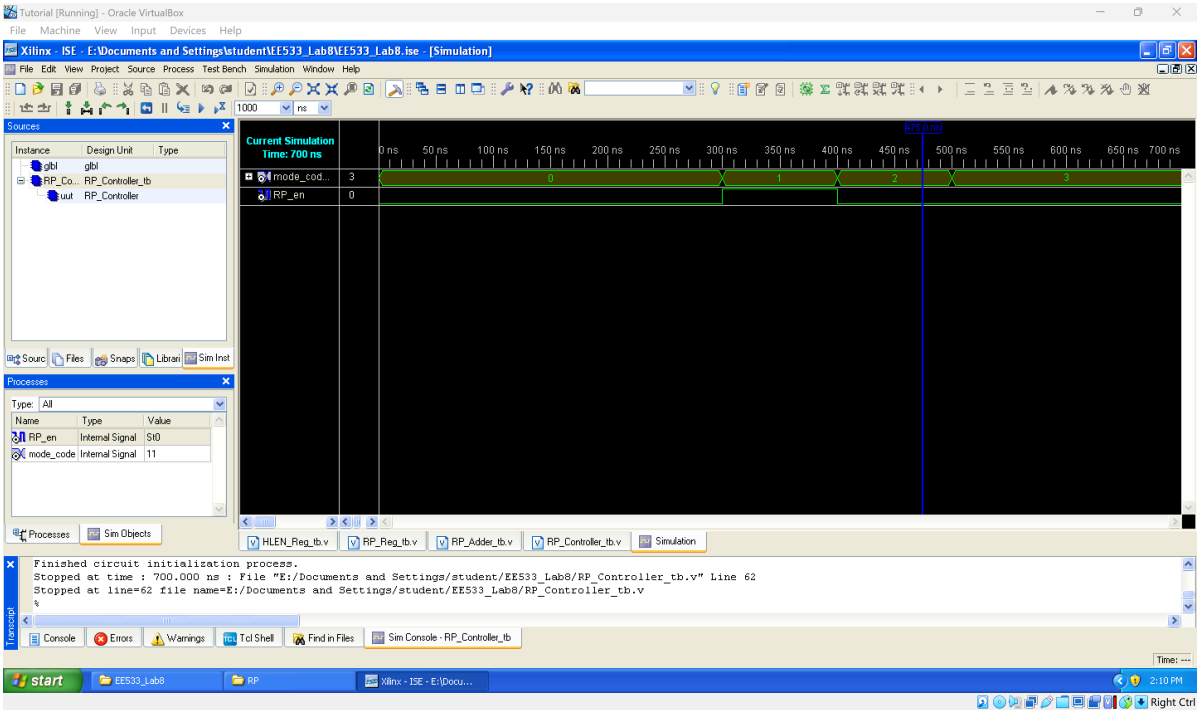
```
`timescale 1ns / 1ps

module RP_Controller
(
    input [1:0] mode_code,

    output RP_en
);

    assign RP_en = (mode_code == 2'b01) ? 1 : 0;

endmodule
```

- Testbench

```
`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   14:07:53 03/08/2025
// Design Name:   RP_Controller
// Module Name:   E:/Documents and Settings/student/EE533_Lab8/RP_Controller_tb.v
// Project Name:  EE533_Lab8
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: RP_Controller
//
```

```
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

module RP_Controller_tb;

    // Inputs
    reg [1:0] mode_code;

    // Outputs
    wire RP_en;

    // Instantiate the Unit Under Test (UUT)
    RP_Controller uut (
        .mode_code(mode_code),
        .RP_en(RP_en)
    );

    initial begin
        // Initialize Inputs
        mode_code = 2'b00;

        // Wait 100 ns for global reset to finish
        #100;

        // Add stimulus here
        #100;
        mode_code = 2'b00;

        #100;
        mode_code = 2'b01;

        #100;
        mode_code = 2'b10;

        #100;
        mode_code = 2'b11;

        #100;

        #100;
        $stop;

    end

endmodule
```

- Waveform



3.2.4 RP_addr_MUX

- Verilog

```
`timescale 1ns / 1ps

module RP_addr_MUX
(
    input RP_ctrl,

    input [7:0] SRAM_addr,
    input [7:0] RP,

    output [7:0] D_raddr
);

    assign D_raddr = RP_ctrl ? RP : SRAM_addr;

endmodule
```

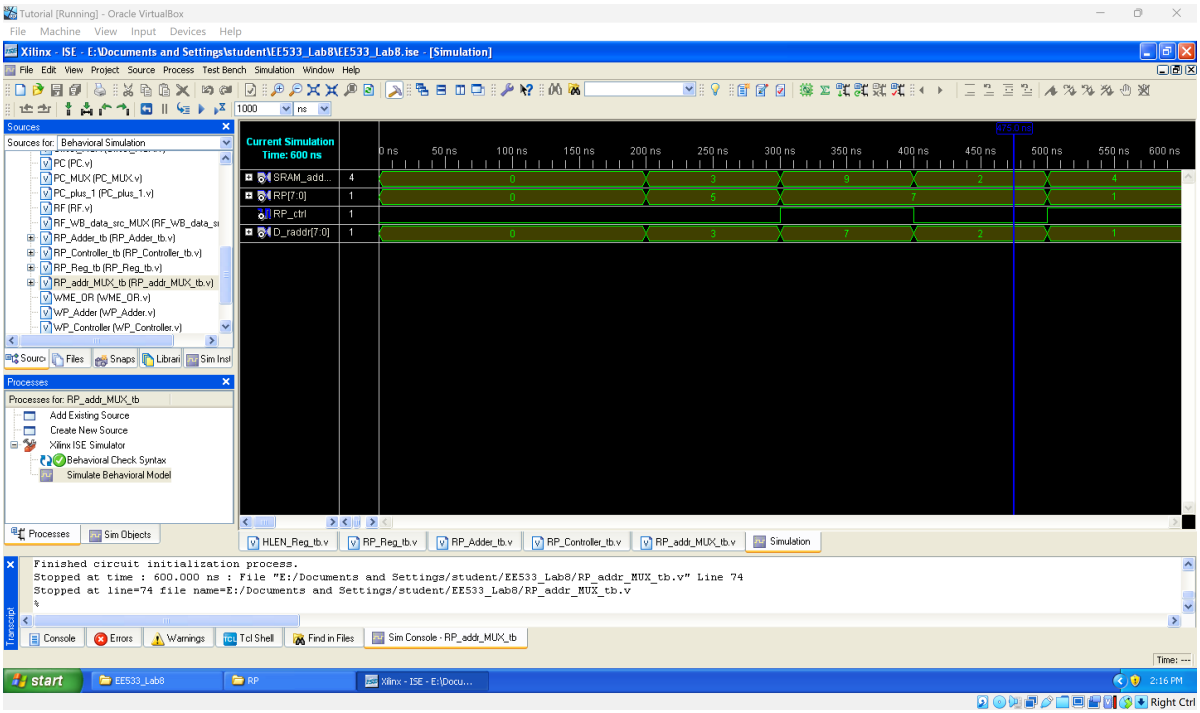
- Testbench

```
`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   14:13:34 03/08/2025
// Design Name:   RP_addr_MUX
// Module Name:   E:/Documents and Settings/student/EE533_Lab8/RP_addr_MUX_tb.v
// Project Name:  EE533_Lab8
// Target Device:
// Tool versions:
// Description:
```

```
//
// Verilog Test Fixture created by ISE for module: RP_addr_MUX
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module RP_addr_MUX_tb;

    // Inputs
    reg RP_ctrl;
    reg [7:0] SRAM_addr;
    reg [7:0] RP;

    // Outputs
    wire [7:0] D_raddr;

    // Instantiate the Unit Under Test (UUT)
    RP_addr_MUX uut (
        .RP_ctrl(RP_ctrl),
        .SRAM_addr(SRAM_addr),
        .RP(RP),
        .D_raddr(D_raddr)
    );

    initial begin
        // Initialize Inputs
        RP_ctrl = 0;
        SRAM_addr = 0;
        RP = 0;

        // Wait 100 ns for global reset to finish
        #100;

        // Add stimulus here
        #100;
        RP_ctrl = 0;
        SRAM_addr = 8'd3;
        RP = 8'd5;

        #100;
        RP_ctrl = 1;
        SRAM_addr = 8'd9;
        RP = 8'd7;

        #100;
        RP_ctrl = 0;
        SRAM_addr = 8'd2;
        RP = 8'd7;

        #100;
        RP_ctrl = 1;
```

```
        SRAM_addr = 8'd4;
        RP = 8'd1;

        #100;
        $stop;

    end

endmodule
```

- Waveform



3.3 WP (as Tail Address)

3.3.1 WP_Reg

- Verilog

```
`timescale 1ns / 1ps

module WP_Reg
(
    input clk,
    input rst,
    input WP_en,
    input FIFO_FULL,
    input [7:0] WP_next,

    output reg [7:0] WP
);

    always @(posedge clk) begin
        if (rst) begin
            WP <= 0;
        end
        else if(WP_en && !FIFO_FULL) begin
```

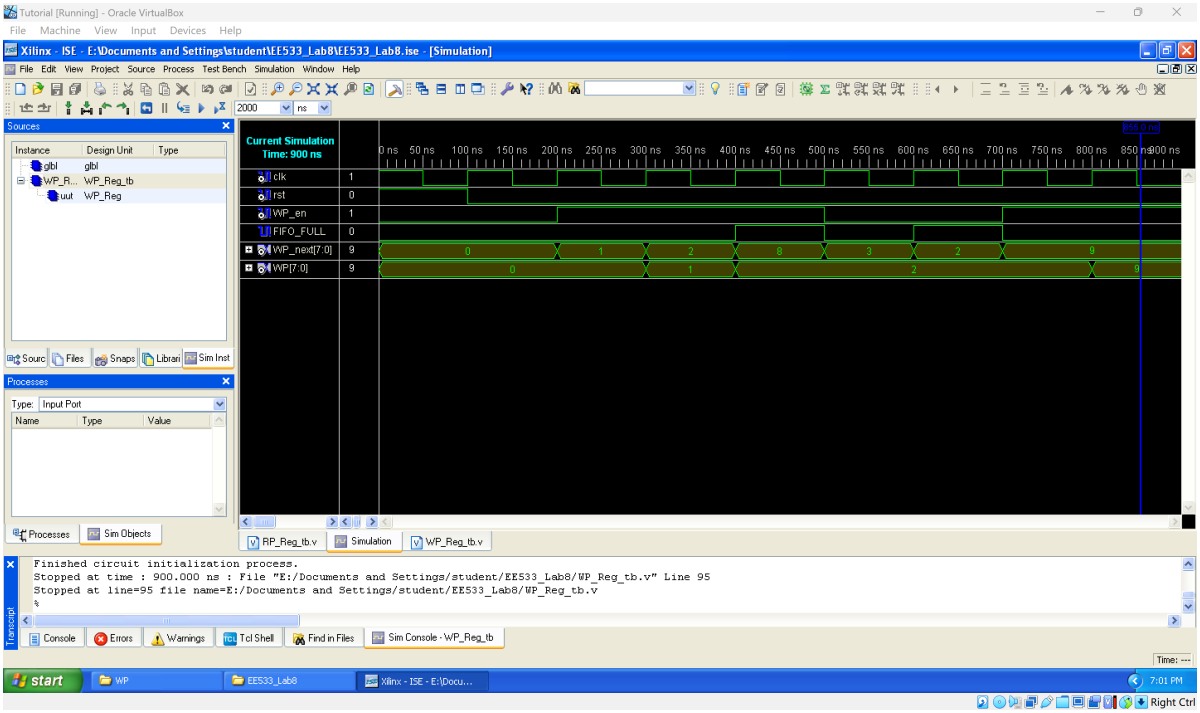
```
            WP <= WP_next;
        end
    end

endmodule
```

- Testbench

```
`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   14:18:15 03/08/2025
// Design Name:   WP_Reg
// Module Name:   E:/Documents and Settings/student/EE533_Lab8/WP_Reg_tb.v
// Project Name:  EE533_Lab8
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: WP_Reg
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module WP_Reg_tb;

    // Inputs
    reg clk;
    reg rst;
    reg WP_en;
    reg FIFO_FULL;
    reg [7:0] WP_next;

    // Outputs
    wire [7:0] WP;

    // Instantiate the Unit Under Test (UUT)
    WP_Reg uut (
        .clk(clk),
        .rst(rst),
        .WP_en(WP_en),
        .FIFO_FULL(FIFO_FULL),
        .WP_next(WP_next),
        .WP(WP)
    );

    always #50 clk = ~clk;
```

```
    initial begin
        // Initialize Inputs
        clk = 1;
        rst = 1;
        WP_en = 0;
        FIFO_FULL = 0;
        WP_next = 0;

        // Wait 100 ns for global reset to finish
        @(posedge clk);
        rst = 0;

        // Add stimulus here
        @(posedge clk);
        WP_en = 1;
        FIFO_FULL = 0;
        WP_next = 8'd1;

        @(posedge clk);
        WP_en = 1;
        FIFO_FULL = 0;
        WP_next = 8'd2;

        @(posedge clk);
        WP_en = 1;
        FIFO_FULL = 1;
        WP_next = 8'd8;

        @(posedge clk);
        WP_en = 0;
        FIFO_FULL = 0;
        WP_next = 8'd3;

        @(posedge clk);
        WP_en = 0;
        FIFO_FULL = 1;
        WP_next = 8'd2;

        @(posedge clk);
        WP_en = 1;
        FIFO_FULL = 0;
        WP_next = 8'd9;

        @(posedge clk);

        @(posedge clk);
        $stop;

    end

endmodule
```

- Waveform



3.3.2 WP_Adder

- Verilog

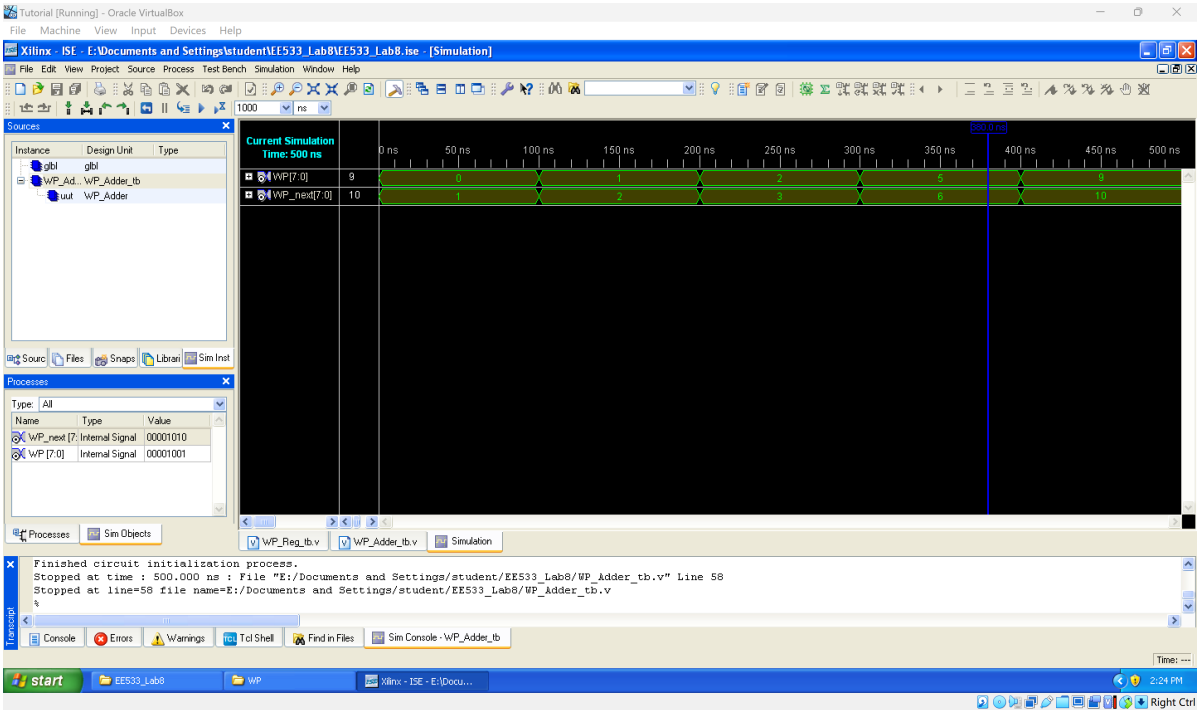
```
`timescale 1ns / 1ps

module WP_Adder
(
    input [7:0] WP,

    output [7:0] WP_next
);

    assign WP_next = WP + 1;

endmodule
```

- Testbench

```
`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   14:22:37 03/08/2025
// Design Name:   WP_Adder
// Module Name:   E:/Documents and Settings/student/EE533_Lab8/WP_Adder_tb.v
// Project Name:  EE533_Lab8
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: WP_Adder
//
```

```
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

module WP_Adder_tb;

    // Inputs
    reg [7:0] WP;

    // Outputs
    wire [7:0] WP_next;

    // Instantiate the Unit Under Test (UUT)
    WP_Adder uut (
        .WP(WP),
        .WP_next(WP_next)
    );

    initial begin
        // Initialize Inputs
        WP = 0;

        // Wait 100 ns for global reset to finish
        #100;
        WP = 8'd1;

        // Add stimulus here
        #100;
        WP = 8'd2;

        #100;
        WP = 8'd5;

        #100;
        WP = 8'd9;

        #100;
        $stop;

    end

endmodule
```

- Waveform



3.3.3 WP_Controller

- Verilog

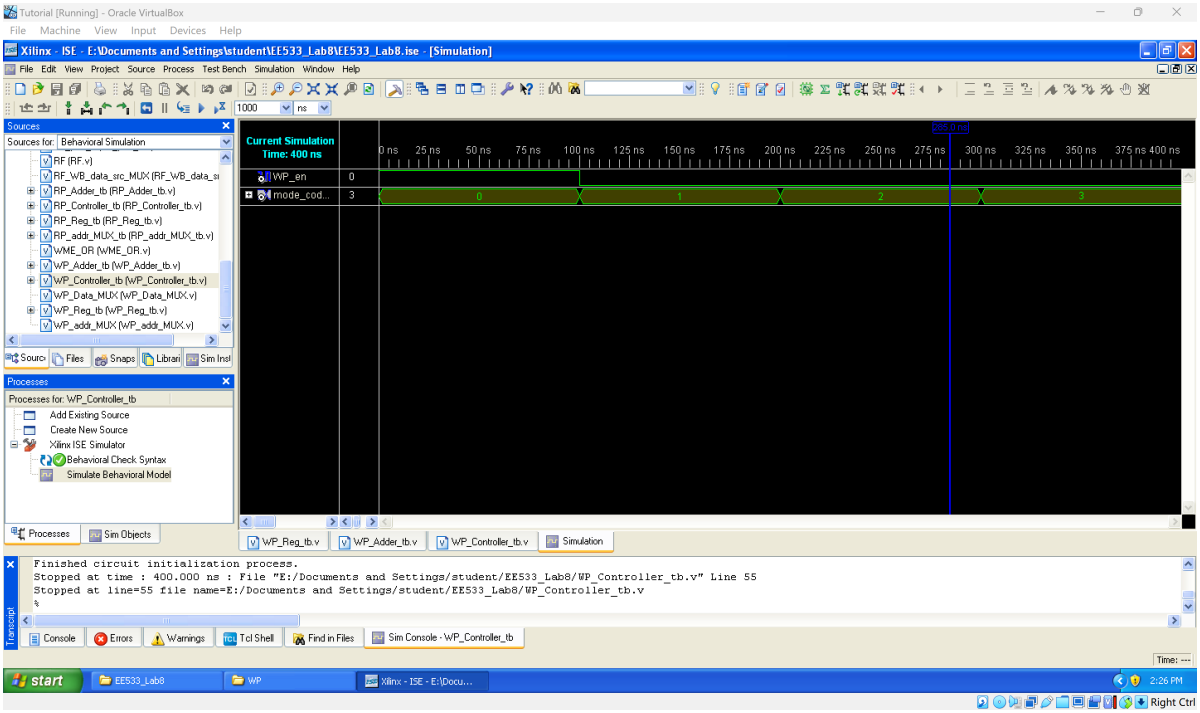
```
`timescale 1ns / 1ps

module WP_Controller
(
    input [1:0] mode_code,

    output WP_en
);

    assign WP_en = (mode_code == 2'b00) ? 1 : 0;

endmodule
```

- Testbench

```
`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   14:25:30 03/08/2025
// Design Name:   WP_Controller
// Module Name:   E:/Documents and Settings/student/EE533_Lab8/WP_Controller_tb.v
// Project Name:  EE533_Lab8
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: WP_Controller
//
```

```
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

module WP_Controller_tb;

    // Inputs
    reg [1:0] mode_code;

    // Outputs
    wire WP_en;

    // Instantiate the Unit Under Test (UUT)
    WP_Controller uut (
        .mode_code(mode_code),
        .WP_en(WP_en)
    );

    initial begin
        // Initialize Inputs
        mode_code = 2'b00;

        // Wait 100 ns for global reset to finish
        #100;
        mode_code = 2'b01;

        // Add stimulus here
        #100;
        mode_code = 2'b10;

        #100;
        mode_code = 2'b11;

        #100;
        $stop;

    end

endmodule
```

- Waveform



3.3.4 WP_addr_MUX

- Verilog

```verilog
`timescale 1ns / 1ps

module WP_addr_MUX
(
    input WP_ctrl,

    input [7:0] SRAM_addr,
    input [7:0] WP,

    output [7:0] D_waddr
);

    assign D_waddr = WP_ctrl ? WP : SRAM_addr;

endmodule
```

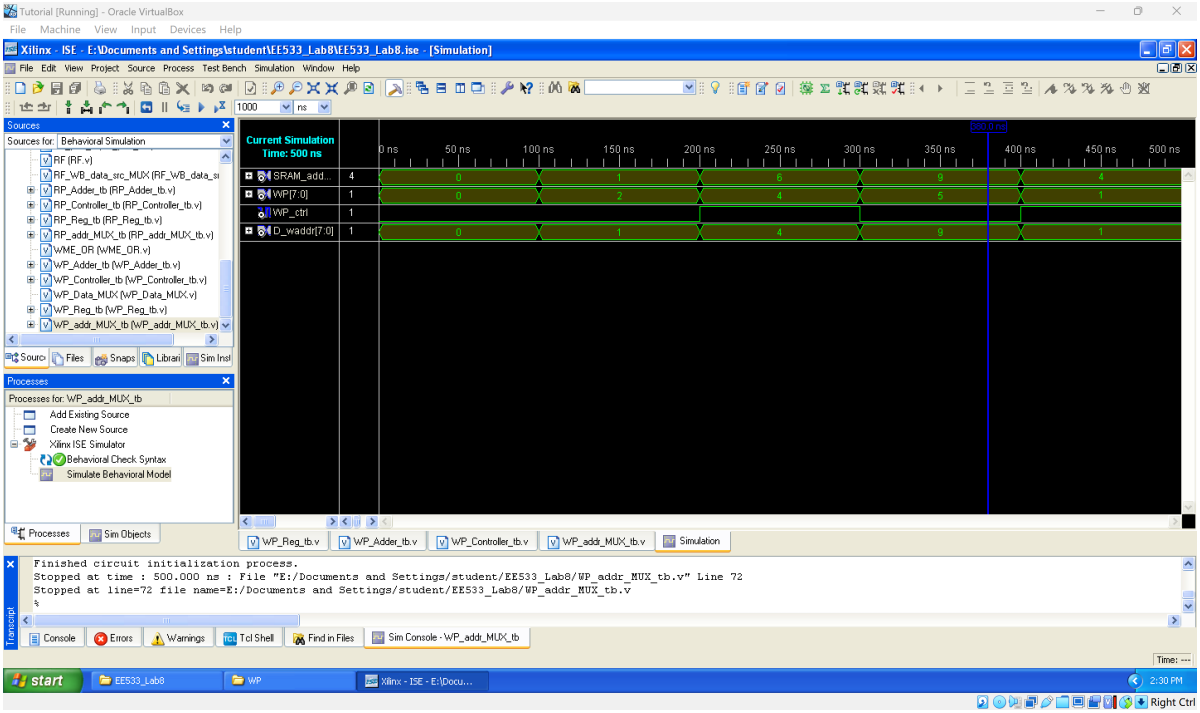
- Testbench

```verilog
`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   14:27:52 03/08/2025
// Design Name:   WP_addr_MUX
// Module Name:   E:/Documents and Settings/student/EE533_Lab8/WP_addr_MUX_tb.v
// Project Name:  EE533_Lab8
// Target Device:
// Tool versions:
// Description:
```

```
//
// Verilog Test Fixture created by ISE for module: WP_addr_MUX
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module WP_addr_MUX_tb;

    // Inputs
    reg WP_ctrl;
    reg [7:0] SRAM_addr;
    reg [7:0] WP;

    // Outputs
    wire [7:0] D_waddr;

    // Instantiate the Unit Under Test (UUT)
    WP_addr_MUX uut (
        .WP_ctrl(WP_ctrl),
        .SRAM_addr(SRAM_addr),
        .WP(WP),
        .D_waddr(D_waddr)
    );

    initial begin
        // Initialize Inputs
        WP_ctrl = 0;
        SRAM_addr = 0;
        WP = 0;

        // Wait 100 ns for global reset to finish
        #100;
        WP_ctrl = 0;
        SRAM_addr = 8'd1;
        WP = 8'd2;

        // Add stimulus here
        #100;
        WP_ctrl = 1;
        SRAM_addr = 8'd6;
        WP = 8'd4;

        #100;
        WP_ctrl = 0;
        SRAM_addr = 8'd9;
        WP = 8'd5;

        #100;
        WP_ctrl = 1;
        SRAM_addr = 8'd4;
        WP = 8'd1;
```

```
        #100;
        $stop;

    end

endmodule
```

- Waveform



3.3.5 WP_Data_MUX

- Verilog

```
`timescale 1ns / 1ps

module WP_Data_MUX
(
    input WP_ctrl,

    input [63:0] SRAM_Din,
    input [63:0] FIFO_Din,

    output [63:0] D_MEM_Din
);

    assign D_MEM_Din = WP_ctrl ? FIFO_Din : SRAM_Din;

endmodule
```

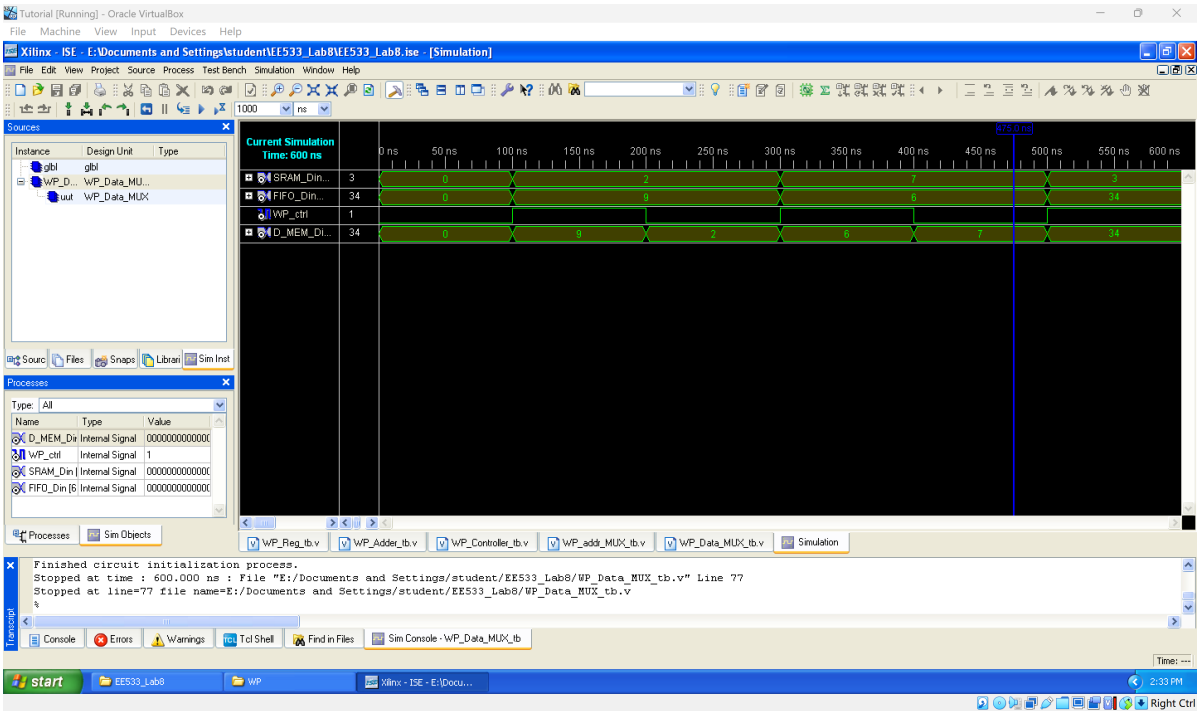
- Testbench

```
`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////
```

```
// Company:
// Engineer:
//
// Create Date:   14:31:20 03/08/2025
// Design Name:   WP_Data_MUX
// Module Name:   E:/Documents and Settings/student/EE533_Lab8/WP_Data_MUX_tb.v
// Project Name:  EE533_Lab8
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: WP_Data_MUX
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module WP_Data_MUX_tb;

    // Inputs
    reg WP_ctrl;
    reg [63:0] SRAM_Din;
    reg [63:0] FIFO_Din;

    // Outputs
    wire [63:0] D_MEM_Din;

    // Instantiate the Unit Under Test (UUT)
    WP_Data_MUX uut (
        .WP_ctrl(WP_ctrl),
        .SRAM_Din(SRAM_Din),
        .FIFO_Din(FIFO_Din),
        .D_MEM_Din(D_MEM_Din)
    );

    initial begin
        // Initialize Inputs
        WP_ctrl = 0;
        SRAM_Din = 0;
        FIFO_Din = 0;

        // Wait 100 ns for global reset to finish
        #100;
        WP_ctrl = 1;
        SRAM_Din = 64'd2;
        FIFO_Din = 64'd9;

        // Add stimulus here
        #100;
        WP_ctrl = 0;
        SRAM_Din = 64'd2;
        FIFO_Din = 64'd9;
```

```
        #100;
        WP_ctrl = 1;
        SRAM_Din = 64'd7;
        FIFO_Din = 64'd6;

        #100;
        WP_ctrl = 0;
        SRAM_Din = 64'd7;
        FIFO_Din = 64'd6;

        #100;
        WP_ctrl = 1;
        SRAM_Din = 64'd3;
        FIFO_Din = 64'd34;

        #100;
        $stop;

    end

endmodule
```

- Waveform



3.4 HLEN & Offset Updated Part

3.4.1 HLEN_Reg

- Verilog

```
`timescale 1ns / 1ps

module HELN_Reg
(
    input clk,
```

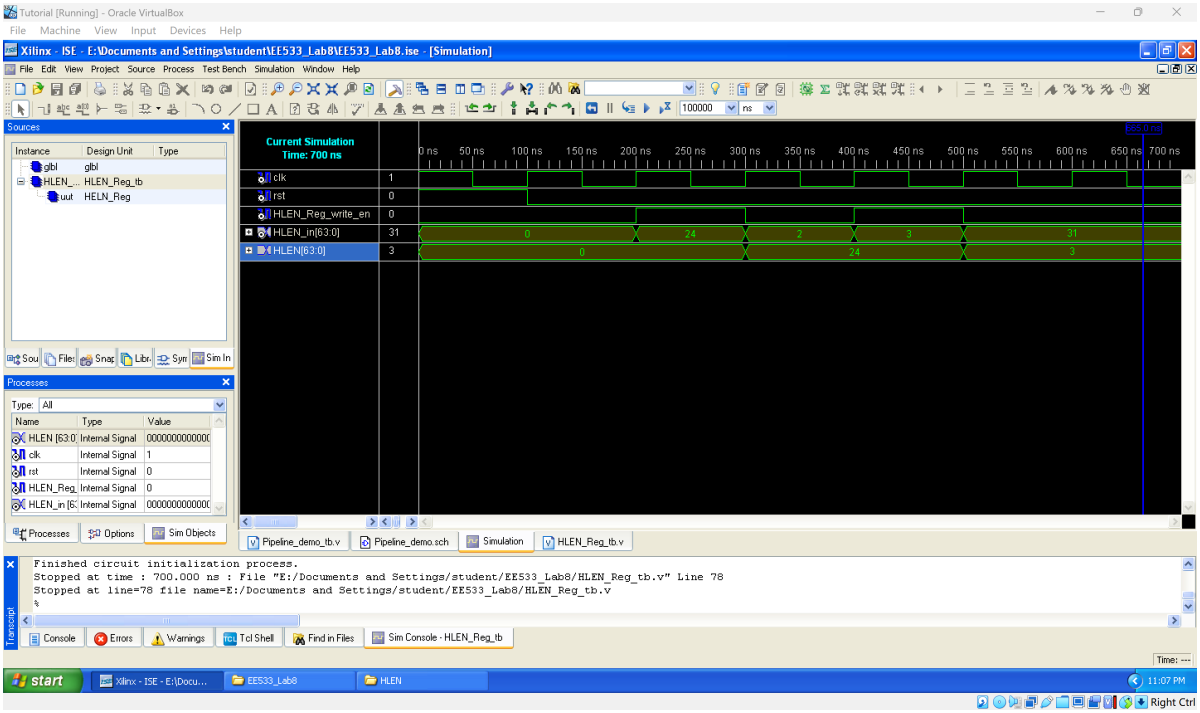
```
    input rst_FIFO,
    input HLEN_Reg_write_en,
    input [63:0] HLEN_in,

    output [63:0] HLEN_out
);

    reg [63:0] HLEN;

    assign HLEN_out = HLEN;

    always @(posedge clk) begin
        if (rst_FIFO) begin
            HLEN <= 0;
        end
        else if (HLEN_Reg_write_en) begin
            HLEN <= HLEN_in;
        end
    end

endmodule
```

- Testbench

```
`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   13:29:22 03/08/2025
// Design Name:   HELN_Reg
// Module Name:   E:/Documents and Settings/student/EE533_Lab8/HLEN_Reg_tb.v
// Project Name:  EE533_Lab8
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: HELN_Reg
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module HLEN_Reg_tb;

    // Inputs
    reg clk;
    reg rst;
    reg HLEN_Reg_write_en;
    reg [63:0] HLEN_in;
```

```
    // Outputs
    wire [63:0] HLEN;

    // Instantiate the Unit Under Test (UUT)
    HELN_Reg uut (
        .clk(clk),
        .rst(rst),
        .HLEN_Reg_write_en(HLEN_Reg_write_en),
        .HLEN_in(HLEN_in),
        .HLEN(HLEN)
    );

    always #50 clk = ~clk;

    initial begin
        // Initialize Inputs
        clk = 1;
        rst = 1;
        HLEN_Reg_write_en = 0;
        HLEN_in = 0;

        // Wait 100 ns for global reset to finish
        @(posedge clk);
        rst = 0;

        // Add stimulus here
        @(posedge clk);
        HLEN_Reg_write_en = 1;
        HLEN_in = 64'd24;

        @(posedge clk);
        HLEN_Reg_write_en = 0;
        HLEN_in = 64'd2;

        @(posedge clk);
        HLEN_Reg_write_en = 1;
        HLEN_in = 64'd3;

        @(posedge clk);
        HLEN_Reg_write_en = 0;
        HLEN_in = 64'd31;

        @(posedge clk);

        @(posedge clk);
        $stop;

    end

endmodule
```

- Waveform



3.4.2 HLEN_Offset_Adder

- Verilog

```
`timescale 1ns / 1ps

module HLEN_Offset_Adder
(
    input [63:0] Offset,
    input [63:0] HLEN,

    output [63:0] result
);

    assign result = Offset + HLEN;

endmodule
```

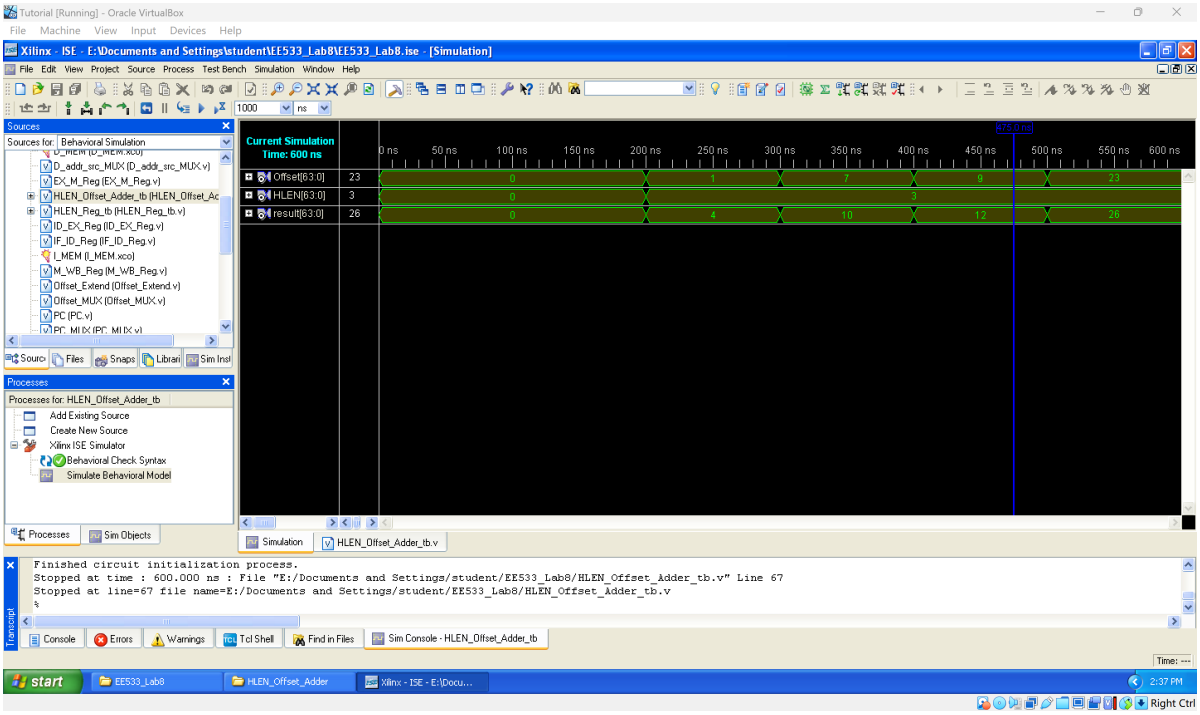
- Testbench

```
`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   14:34:57 03/08/2025
// Design Name:   HLEN_Offset_Adder
// Module Name:   E:/Documents and
Settings/student/EE533_Lab8/HLEN_Offset_Adder_tb.v
// Project Name:  EE533_Lab8
// Target Device:
// Tool versions:
// Description:
//
```

```
// Verilog Test Fixture created by ISE for module: HLEN_Offset_Adder
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module HLEN_Offset_Adder_tb;

    // Inputs
    reg [63:0] Offset;
    reg [63:0] HLEN;

    // Outputs
    wire [63:0] result;

    // Instantiate the Unit Under Test (UUT)
    HLEN_Offset_Adder uut (
        .Offset(Offset),
        .HLEN(HLEN),
        .result(result)
    );

    initial begin
        // Initialize Inputs
        Offset = 0;
        HLEN = 0;

        // Wait 100 ns for global reset to finish
        #100;

        // Add stimulus here
        #100;
        Offset = 64'd1;
        HLEN = 64'd3;

        #100;
        Offset = 64'd7;
        HLEN = 64'd3;

        #100;
        Offset = 64'd9;
        HLEN = 64'd3;

        #100;
        Offset = 64'd23;
        HLEN = 64'd3;

        #100;
        $stop;

    end
```

```
endmodule
```

- Waveform



3.4.3 Offset_MUX

- Verilog

```
`timescale 1ns / 1ps

module Offset_MUX
(
    input LW_EX,
    input SW_EX,

    input [63:0] HLEN_Offset_Adder_result,
    input [63:0] Offset_EX,

    output [63:0] ALU_src_MUX_in
);

    wire Offset_MUX_ctrl;

    assign Offset_MUX_ctrl = LW_EX || SW_EX;

    assign ALU_src_MUX_in = Offset_MUX_ctrl ? HLEN_Offset_Adder_result :
Offset_EX;

endmodule
```

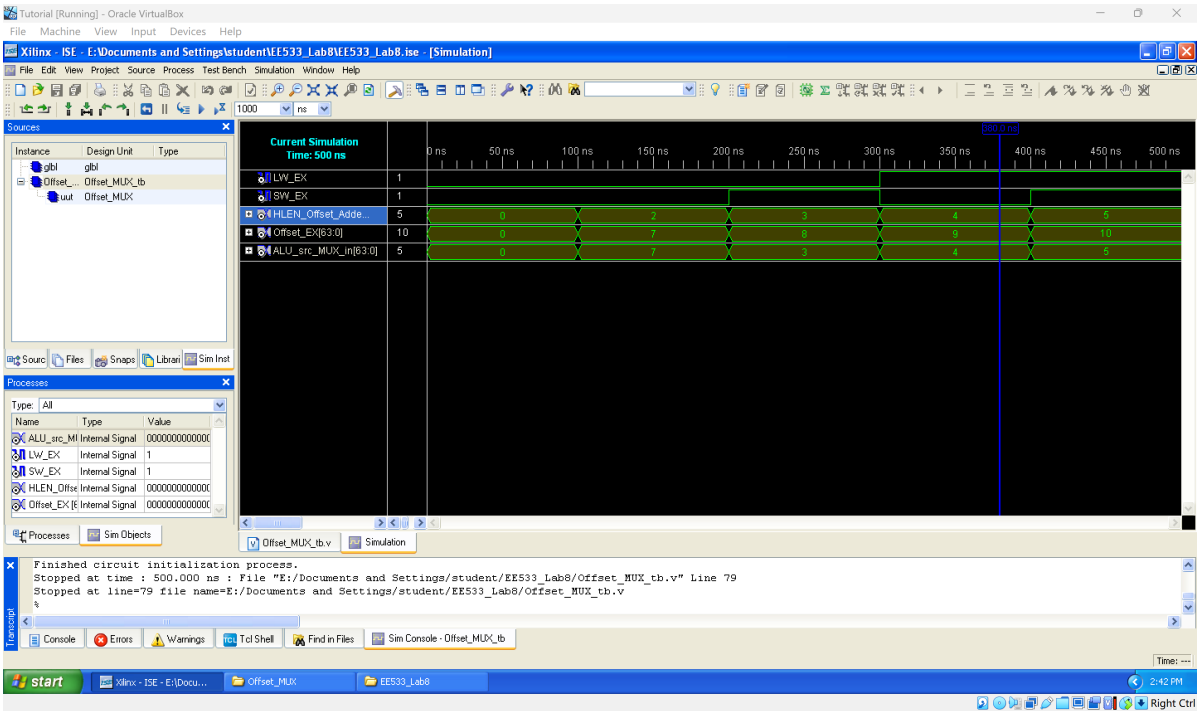
- Testbench

```
`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////
```

```
// Company:
// Engineer:
//
// Create Date:   14:39:47 03/08/2025
// Design Name:   Offset_MUX
// Module Name:   E:/Documents and Settings/student/EE533_Lab8/Offset_MUX_tb.v
// Project Name:  EE533_Lab8
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: Offset_MUX
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module Offset_MUX_tb;

    // Inputs
    reg LW_EX;
    reg SW_EX;
    reg [63:0] HLEN_Offset_Adder_result;
    reg [63:0] Offset_EX;

    // Outputs
    wire [63:0] ALU_src_MUX_in;

    // Instantiate the Unit Under Test (UUT)
    Offset_MUX uut (
        .LW_EX(LW_EX),
        .SW_EX(SW_EX),
        .HLEN_Offset_Adder_result(HLEN_Offset_Adder_result),
        .Offset_EX(Offset_EX),
        .ALU_src_MUX_in(ALU_src_MUX_in)
    );

    initial begin
        // Initialize Inputs
        LW_EX = 0;
        SW_EX = 0;
        HLEN_Offset_Adder_result = 0;
        Offset_EX = 0;

        // Wait 100 ns for global reset to finish
        #100;
        LW_EX = 0;
        SW_EX = 0;
        HLEN_Offset_Adder_result = 64'd2;
        Offset_EX = 64'd7;

        // Add stimulus here
```

```
        #100;
        LW_EX = 0;
        SW_EX = 1;
        HLEN_Offset_Adder_result = 64'd3;
        Offset_EX = 64'd8;

        #100;
        LW_EX = 1;
        SW_EX = 0;
        HLEN_Offset_Adder_result = 64'd4;
        Offset_EX = 64'd9;

        #100;
        LW_EX = 1;
        SW_EX = 1;
        HLEN_Offset_Adder_result = 64'd5;
        Offset_EX = 64'd10;

        #100;
        $stop;

    end

endmodule
```

- Waveform



3.5 SRAM_addr_Adder

- Verilog

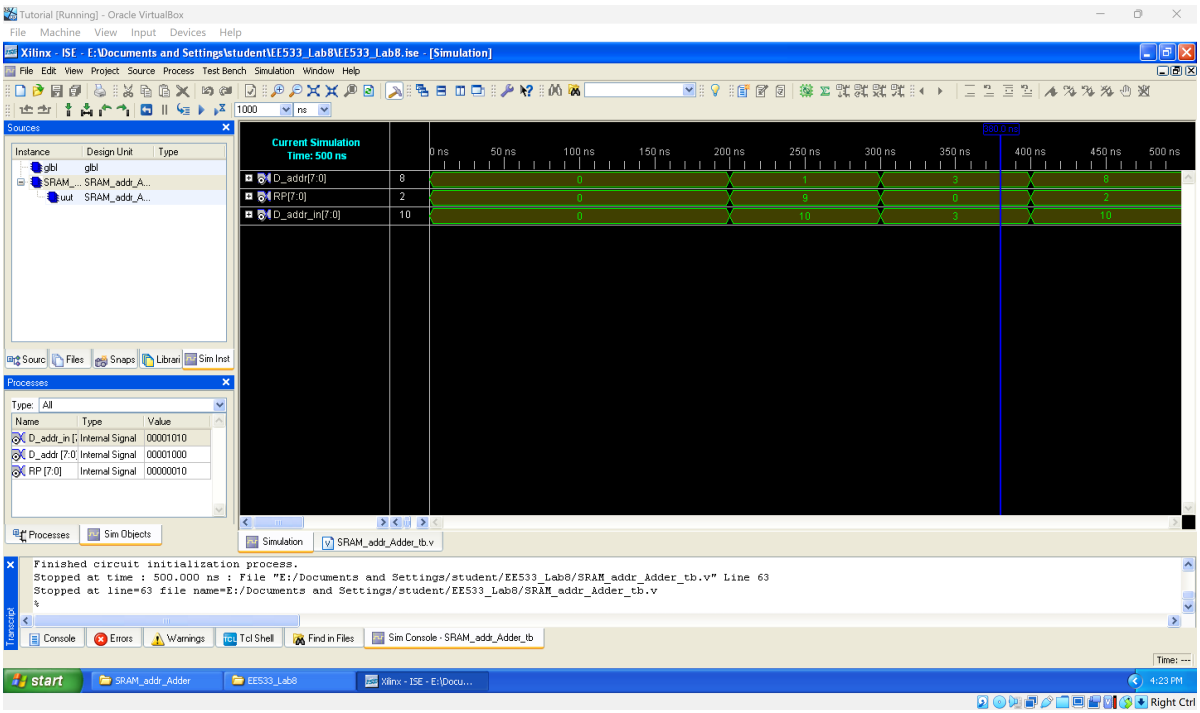
```
`timescale 1ns / 1ps

module SRAM_addr_Adder
(
    input [7:0] D_addr,
    input [7:0] RP,

    output [7:0] D_addr_in
);

    assign D_addr_in = D_addr + RP;

endmodule
```

- Testbench

```
`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   16:20:10 03/08/2025
// Design Name:   SRAM_addr_Adder
// Module Name:   E:/Documents and
Settings/student/EE533_Lab8/SRAM_addr_Adder_tb.v
// Project Name:  EE533_Lab8
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: SRAM_addr_Adder
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

module SRAM_addr_Adder_tb;

    // Inputs
    reg [7:0] D_addr;
    reg [7:0] RP;

    // Outputs
    wire [7:0] D_addr_in;

    // Instantiate the Unit Under Test (UUT)
    SRAM_addr_Adder uut (
        .D_addr(D_addr),
        .RP(RP),
```

```
        .D_addr_in(D_addr_in)
    );

    initial begin
        // Initialize Inputs
        D_addr = 0;
        RP = 0;

        // Wait 100 ns for global reset to finish
        #100;

        // Add stimulus here
        #100;
        D_addr = 8'd1;
        RP = 8'd9;

        #100;
        D_addr = 8'd3;
        RP = 8'd0;

        #100;
        D_addr = 8'd8;
        RP = 8'd2;

        #100;
        $stop;

    end

endmodule
```

- Waveform



3.6 WME_OR

- Verilog

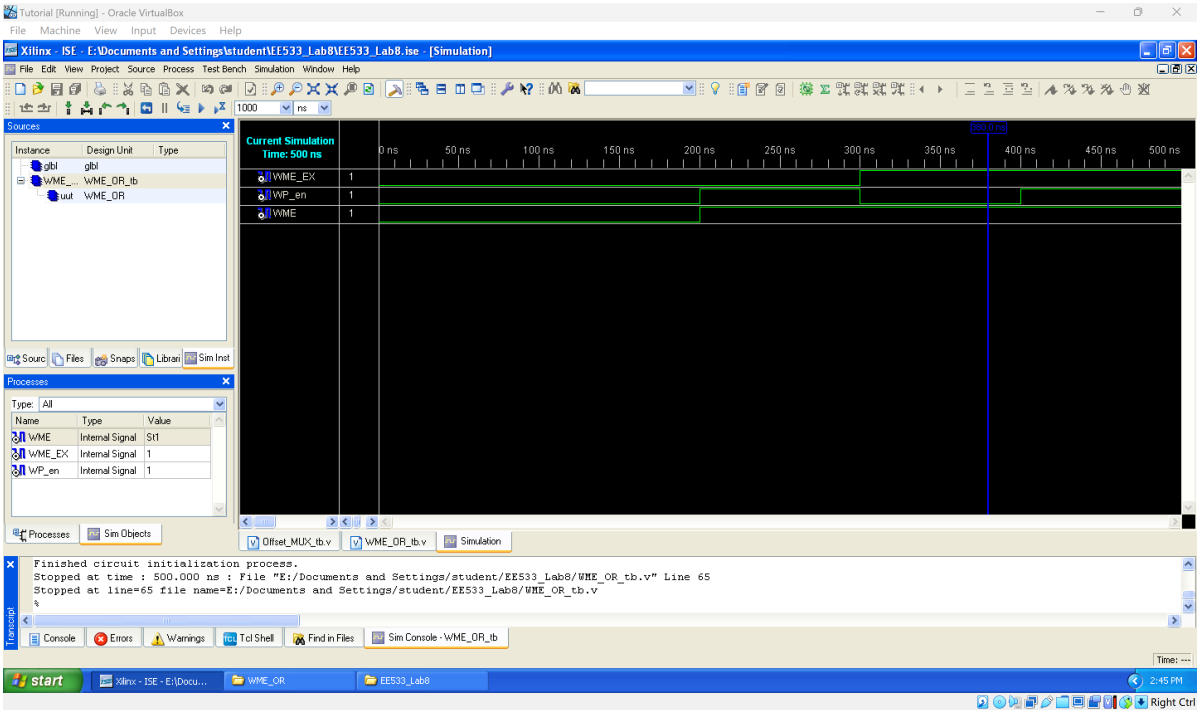
```
`timescale 1ns / 1ps

module WME_OR
(
    input WME_EX,
    input WP_en,

    output WME
);

    assign WME = WME_EX || WP_en;

endmodule
```

- Testbench

```
`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   14:43:27 03/08/2025
// Design Name:   WME_OR
// Module Name:   E:/Documents and Settings/student/EE533_Lab8/WME_OR_tb.v
// Project Name:  EE533_Lab8
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: WME_OR
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module WME_OR_tb;

    // Inputs
    reg WME_EX;
    reg WP_en;

    // Outputs
    wire WME;

    // Instantiate the Unit Under Test (UUT)
```

```
    WME_OR uut (
        .WME_EX(WME_EX),
        .WP_en(WP_en),
        .WME(WME)
    );

    initial begin
        // Initialize Inputs
        WME_EX = 0;
        WP_en = 0;

        // Wait 100 ns for global reset to finish
        #100;
        WME_EX = 0;
        WP_en = 0;

        // Add stimulus here
        #100;
        WME_EX = 0;
        WP_en = 1;

        #100;
        WME_EX = 1;
        WP_en = 0;

        #100;
        WME_EX = 1;
        WP_en = 1;

        #100;
        $stop;

    end

endmodule
```

- Waveform



3.7 FIFO_state_controller

- Verilog

```
`timescale 1ns / 1ps

module FIFO_state_controller #
(
    parameter FIFO_SIZE = 256,
    parameter FIFO_FULL_THREAD = 240,
    parameter FIFO_EMPTY_THREAD = 16
)
(
    input [7:0] WP,
    input [7:0] RP,
    input clk,
    input rst_FIFO,

    output [7:0] depth,
    output FIFO_EMPTY,
    output FIFO_FULL,
    output reg FIFO_almost_full,
    output reg FIFO_almost_empty
);

    assign FIFO_EMPTY = (WP == RP) && FIFO_almost_empty;
    assign FIFO_FULL = (WP == RP) && FIFO_almost_full;
    assign depth = (WP >= RP) ? (WP - RP) : (WP - RP + FIFO_SIZE);

    always @(posedge clk) begin
        if (rst_FIFO) begin
            FIFO_almost_empty <= 0;
            FIFO_almost_full <= 0;
        end
        else begin
```

```
            FIFO_almost_empty <= (depth < FIFO_EMPTY_THREAD) ? 1 : 0;
            FIFO_almost_full <= (depth > FIFO_FULL_THREAD) ? 1 : 0;
        end
    end

endmodule
```

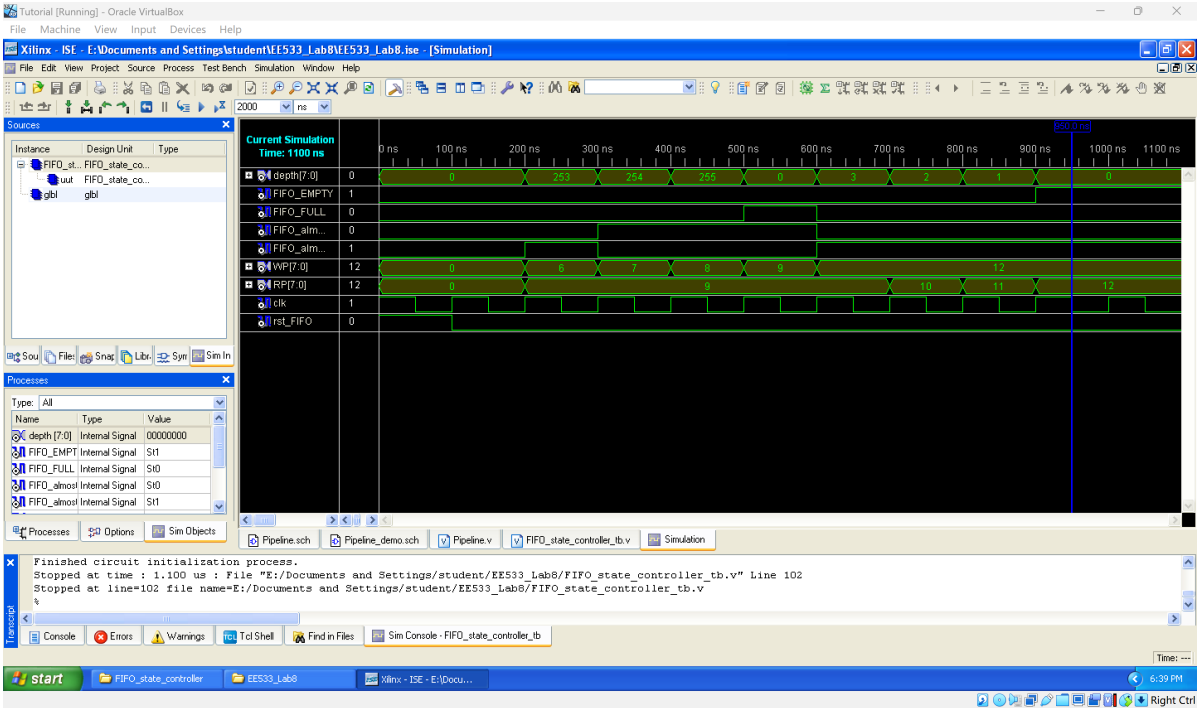
- Testbench

```
`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   18:31:47 03/08/2025
// Design Name:   FIFO_state_controller
// Module Name:   E:/Documents and
Settings/student/EE533_Lab8/FIFO_state_controller_tb.v
// Project Name:  EE533_Lab8
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: FIFO_state_controller
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module FIFO_state_controller_tb;

    // Inputs
    reg [7:0] WP;
    reg [7:0] RP;
    reg clk;
    reg rst_FIFO;

    // Outputs
    wire [7:0] depth;
    wire FIFO_EMPTY;
    wire FIFO_FULL;
    wire FIFO_almost_full;
    wire FIFO_almost_empty;

    // Instantiate the Unit Under Test (UUT)
    FIFO_state_controller uut (
        .WP(WP),
        .RP(RP),
        .clk(clk),
        .rst_FIFO(rst_FIFO),
```

```
        .depth(depth),
        .FIFO_EMPTY(FIFO_EMPTY),
        .FIFO_FULL(FIFO_FULL),
        .FIFO_almost_full(FIFO_almost_full),
        .FIFO_almost_empty(FIFO_almost_empty)
    );

    always #50 clk = ~clk;

    initial begin
        // Initialize Inputs
        WP = 0;
        RP = 0;
        clk = 1;
        rst_FIFO = 1;

        // Wait 100 ns for global reset to finish
        @(posedge clk);
        rst_FIFO = 0;

        // Add stimulus here
        @(posedge clk);
        WP = 8'd6;
        RP = 8'd9;

        @(posedge clk);
        WP = 8'd7;
        RP = 8'd9;

        @(posedge clk);
        WP = 8'd8;
        RP = 8'd9;

        @(posedge clk);
        WP = 8'd9;
        RP = 8'd9;

        @(posedge clk);
        WP = 8'd12;
        RP = 8'd9;

        @(posedge clk);
        WP = 8'd12;
        RP = 8'd10;

        @(posedge clk);
        WP = 8'd12;
        RP = 8'd11;

        @(posedge clk);
        WP = 8'd12;
        RP = 8'd12;

        @(posedge clk);

        @(posedge clk);
```

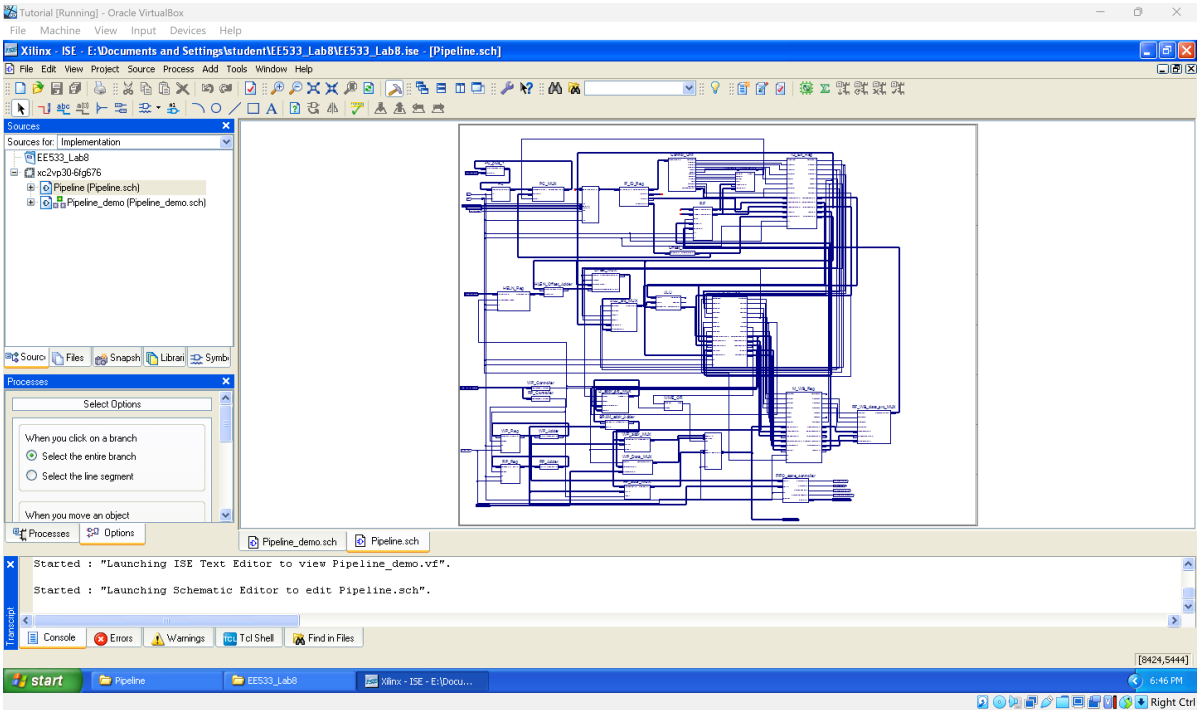
```
        $stop;

    end

endmodule
```

- Waveform



4. Pipeline

- Schematic



- Verilog

```
//////////////////////////////////////////////////////////////////////////////////
// Copyright (c) 1995-2008 Xilinx, Inc.  All rights reserved.
```

```
//////////////////////////////////////////////////////////////////////////////
//   ____  ____
//  /   /\/   /
// /___/  \  /    Vendor: Xilinx
// \   \   \/     Version : 10.1
//  \   \         Application : sch2verilog
//  /   /         Filename : Pipeline_demo.vf
// /___/   /\     Timestamp : 03/09/2025 03:21:43
// \   \  /  \
//  \___\/\___\
//
//Command: C:\Xilinx\10.1\ISE\bin\nt\unwrapped\sch2verilog.exe -intstyle ise -
family virtex2p -w "E:/Documents and
Settings/student/EE533_Lab8/Pipeline_demo.sch" Pipeline_demo.vf
//Design Name: Pipeline_demo
//Device: virtex2p
//Purpose:
//    This verilog netlist is translated from an ECS schematic.It can be
//    synthesized and simulated, but it should not be modified.
//
`timescale 1ns / 1ps

module Pipeline_demo(clk,
                     HLEN,
                     Instr_IN,
                     Instr_IN_addr,
                     Instr_IN_en,
                     mode_code,
                     ONE,
                     pkt_in,
                     rst,
                     rst_FIFO,
                     ADDI_EX,
                     ADDI_ID,
                     ADDI_M,
                     ADDI_WB,
                     ALU_B_in,
                     ALU_OP_EX,
                     ALU_result_EX,
                     ALU_result_M,
                     ALU_result_WB,
                     BEQ_ID,
                     BGT_ID,
                     BLT_ID,
                     Dout_WB,
                     D_addr_src_MUX_out,
                     D_raddr,
                     D_waddr,
                     D_wdata,
                     FIFO_almost_empty,
                     FIFO_almost_full,
                     FIFO_depth,
                     FIFO_EMPTY,
                     FIFO_FULL,
                     HLEN_Offset_sum,
                     HLEN_out,
```

```
                    Instruction_IF,
                    J_ID,
                    LW_EX,
                    LW_ID,
                    LW_M,
                    LW_WB,
                    MOVI_EX,
                    MOVI_ID,
                    MOVI_M,
                    MOVI_WB,
                    NOOP_EX,
                    NOOP_ID,
                    NOOP_M,
                    NOOP_WB,
                    Offset_EX,
                    Offset_ID,
                    Offset_M,
                    Offset_MUX_out,
                    Offset_WB,
                    OP_CODE_ID,
                    PC,
                    PC_ctrl,
                    PC_next,
                    PC_plus_1,
                    pkt_out,
                    RF_Din,
                    RP_en,
                    rs_data_EX,
                    rs_data_ID,
                    rs_data_M,
                    rs_data_WB,
                    rs_ID,
                    rt_data_EX,
                    rt_data_ID,
                    rt_data_M,
                    rt_data_WB,
                    rt_EX,
                    rt_ID,
                    rt_M,
                    rt_WB,
                    SRAM_addr,
                    SUBI_EX,
                    SUBI_ID,
                    SUBI_M,
                    SUBI_WB,
                    SW_EX,
                    SW_ID,
                    SW_M,
                    SW_WB,
                    WME_EX,
                    WME_M,
                    WME_M_out,
                    WP_en,
                    WRE_EX,
                    WRE_M,
                    WRE_WB);
```

```
 input clk;
 input [63:0] HLEN;
 input [31:0] Instr_IN;
 input [8:0] Instr_IN_addr;
 input Instr_IN_en;
 input [1:0] mode_code;
 input [63:0] ONE;
 input [63:0] pkt_in;
 input rst;
 input rst_FIFO;
output ADDI_EX;
output ADDI_ID;
output ADDI_M;
output ADDI_WB;
output [63:0] ALU_B_in;
output [3:0] ALU_OP_EX;
output [63:0] ALU_result_EX;
output [63:0] ALU_result_M;
output [63:0] ALU_result_WB;
output BEQ_ID;
output BGT_ID;
output BLT_ID;
output [63:0] Dout_WB;
output [7:0] D_addr_src_MUX_out;
output [7:0] D_raddr;
output [7:0] D_waddr;
output [63:0] D_wdata;
output FIFO_almost_empty;
output FIFO_almost_full;
output [7:0] FIFO_depth;
output FIFO_EMPTY;
output FIFO_FULL;
output [63:0] HLEN_Offset_sum;
output [63:0] HLEN_out;
output [31:0] Instruction_IF;
output J_ID;
output LW_EX;
output LW_ID;
output LW_M;
output LW_WB;
output MOVI_EX;
output MOVI_ID;
output MOVI_M;
output MOVI_WB;
output NOOP_EX;
output NOOP_ID;
output NOOP_M;
output NOOP_WB;
output [63:0] Offset_EX;
output [63:0] Offset_ID;
output [63:0] Offset_M;
output [63:0] Offset_MUX_out;
output [63:0] Offset_WB;
output [5:0] OP_CODE_ID;
output [63:0] PC;
```

```
    output PC_ctrl;
    output [63:0] PC_next;
    output [63:0] PC_plus_1;
    output [63:0] pkt_out;
    output [63:0] RF_Din;
    output RP_en;
    output [63:0] rs_data_EX;
    output [63:0] rs_data_ID;
    output [63:0] rs_data_M;
    output [63:0] rs_data_WB;
    output [4:0] rs_ID;
    output [63:0] rt_data_EX;
    output [63:0] rt_data_ID;
    output [63:0] rt_data_M;
    output [63:0] rt_data_WB;
    output [4:0] rt_EX;
    output [4:0] rt_ID;
    output [4:0] rt_M;
    output [2:0] rt_WB;
    output [7:0] SRAM_addr;
    output SUBI_EX;
    output SUBI_ID;
    output SUBI_M;
    output SUBI_WB;
    output SW_EX;
    output SW_ID;
    output SW_M;
    output SW_WB;
    output WME_EX;
    output WME_M;
    output WME_M_out;
    output WP_en;
    output WRE_EX;
    output WRE_M;
    output WRE_WB;

    wire [3:0] ALU_OP_ID;
    wire [15:0] Offset_ID_before_Extend;
    wire [7:0] RP;
    wire [7:0] RP_next;
    wire WME_ID;
    wire [7:0] WP;
    wire [7:0] WP_next;
    wire WRE_ID;
    wire NOOP_ID_DUMMY;
    wire [7:0] D_waddr_DUMMY;
    wire [63:0] ALU_result_WB_DUMMY;
    wire [63:0] ALU_result_EX_DUMMY;
    wire SUBI_ID_DUMMY;
    wire [63:0] rs_data_ID_DUMMY;
    wire [63:0] Offset_WB_DUMMY;
    wire RP_en_DUMMY;
    wire [63:0] Offset_EX_DUMMY;
    wire [2:0] rt_WB_DUMMY;
    wire [7:0] SRAM_addr_DUMMY;
    wire [63:0] rt_data_EX_DUMMY;
```

```
    wire SW_ID_DUMMY;
    wire FIFO_FULL_DUMMY;
    wire [63:0] HLEN_Offset_sum_DUMMY;
    wire [4:0] rt_EX_DUMMY;
    wire [63:0] RF_Din_DUMMY;
    wire [63:0] PC_DUMMY;
    wire FIFO_EMPTY_DUMMY;
    wire [4:0] rt_M_DUMMY;
    wire WRE_WB_DUMMY;
    wire [31:0] Instruction_IF_DUMMY;
    wire [63:0] Offset_MUX_out_DUMMY;
    wire [63:0] rs_data_M_DUMMY;
    wire PC_ctrl_DUMMY;
    wire [63:0] ALU_result_M_DUMMY;
    wire MOVI_WB_DUMMY;
    wire WRE_EX_DUMMY;
    wire [63:0] D_wdata_DUMMY;
    wire MOVI_EX_DUMMY;
    wire SW_M_DUMMY;
    wire BGT_ID_DUMMY;
    wire ADDI_ID_DUMMY;
    wire [7:0] D_addr_src_MUX_out_DUMMY;
    wire NOOP_EX_DUMMY;
    wire MOVI_M_DUMMY;
    wire BEQ_ID_DUMMY;
    wire [63:0] Offset_M_DUMMY;
    wire WP_en_DUMMY;
    wire SUBI_WB_DUMMY;
    wire [3:0] ALU_OP_EX_DUMMY;
    wire SUBI_EX_DUMMY;
    wire [63:0] rs_data_EX_DUMMY;
    wire LW_ID_DUMMY;
    wire [4:0] rs_ID_DUMMY;
    wire NOOP_M_DUMMY;
    wire SW_EX_DUMMY;
    wire WME_M_out_DUMMY;
    wire [63:0] HLEN_out_DUMMY;
    wire [7:0] D_raddr_DUMMY;
    wire [63:0] Offset_ID_DUMMY;
    wire BLT_ID_DUMMY;
    wire [63:0] Dout_WB_DUMMY;
    wire SUBI_M_DUMMY;
    wire [63:0] pkt_out_DUMMY;
    wire [63:0] rt_data_ID_DUMMY;
    wire ADDI_M_DUMMY;
    wire [4:0] rt_ID_DUMMY;
    wire ADDI_WB_DUMMY;
    wire [63:0] rt_data_M_DUMMY;
    wire ADDI_EX_DUMMY;
    wire [63:0] ALU_B_in_DUMMY;
    wire J_ID_DUMMY;
    wire LW_WB_DUMMY;
    wire LW_EX_DUMMY;
    wire MOVI_ID_DUMMY;
    wire [63:0] PC_plus_1_DUMMY;
    wire WME_M_DUMMY;
```

```
    wire [5:0] OP_CODE_ID_DUMMY;
    wire WRE_M_DUMMY;
    wire LW_M_DUMMY;
    wire [63:0] PC_next_DUMMY;
    wire WME_EX_DUMMY;

    assign ADDI_EX = ADDI_EX_DUMMY;
    assign ADDI_ID = ADDI_ID_DUMMY;
    assign ADDI_M = ADDI_M_DUMMY;
    assign ADDI_WB = ADDI_WB_DUMMY;
    assign ALU_B_in[63:0] = ALU_B_in_DUMMY[63:0];
    assign ALU_OP_EX[3:0] = ALU_OP_EX_DUMMY[3:0];
    assign ALU_result_EX[63:0] = ALU_result_EX_DUMMY[63:0];
    assign ALU_result_M[63:0] = ALU_result_M_DUMMY[63:0];
    assign ALU_result_WB[63:0] = ALU_result_WB_DUMMY[63:0];
    assign BEQ_ID = BEQ_ID_DUMMY;
    assign BGT_ID = BGT_ID_DUMMY;
    assign BLT_ID = BLT_ID_DUMMY;
    assign Dout_WB[63:0] = Dout_WB_DUMMY[63:0];
    assign D_addr_src_MUX_out[7:0] = D_addr_src_MUX_out_DUMMY[7:0];
    assign D_raddr[7:0] = D_raddr_DUMMY[7:0];
    assign D_waddr[7:0] = D_waddr_DUMMY[7:0];
    assign D_wdata[63:0] = D_wdata_DUMMY[63:0];
    assign FIFO_EMPTY = FIFO_EMPTY_DUMMY;
    assign FIFO_FULL = FIFO_FULL_DUMMY;
    assign HLEN_Offset_sum[63:0] = HLEN_Offset_sum_DUMMY[63:0];
    assign HLEN_out[63:0] = HLEN_out_DUMMY[63:0];
    assign Instruction_IF[31:0] = Instruction_IF_DUMMY[31:0];
    assign J_ID = J_ID_DUMMY;
    assign LW_EX = LW_EX_DUMMY;
    assign LW_ID = LW_ID_DUMMY;
    assign LW_M = LW_M_DUMMY;
    assign LW_WB = LW_WB_DUMMY;
    assign MOVI_EX = MOVI_EX_DUMMY;
    assign MOVI_ID = MOVI_ID_DUMMY;
    assign MOVI_M = MOVI_M_DUMMY;
    assign MOVI_WB = MOVI_WB_DUMMY;
    assign NOOP_EX = NOOP_EX_DUMMY;
    assign NOOP_ID = NOOP_ID_DUMMY;
    assign NOOP_M = NOOP_M_DUMMY;
    assign Offset_EX[63:0] = Offset_EX_DUMMY[63:0];
    assign Offset_ID[63:0] = Offset_ID_DUMMY[63:0];
    assign Offset_M[63:0] = Offset_M_DUMMY[63:0];
    assign Offset_MUX_out[63:0] = Offset_MUX_out_DUMMY[63:0];
    assign Offset_WB[63:0] = Offset_WB_DUMMY[63:0];
    assign OP_CODE_ID[5:0] = OP_CODE_ID_DUMMY[5:0];
    assign PC[63:0] = PC_DUMMY[63:0];
    assign PC_ctrl = PC_ctrl_DUMMY;
    assign PC_next[63:0] = PC_next_DUMMY[63:0];
    assign PC_plus_1[63:0] = PC_plus_1_DUMMY[63:0];
    assign pkt_out[63:0] = pkt_out_DUMMY[63:0];
    assign RF_Din[63:0] = RF_Din_DUMMY[63:0];
    assign RP_en = RP_en_DUMMY;
    assign rs_data_EX[63:0] = rs_data_EX_DUMMY[63:0];
    assign rs_data_ID[63:0] = rs_data_ID_DUMMY[63:0];
    assign rs_data_M[63:0] = rs_data_M_DUMMY[63:0];
```

```
    assign rs_ID[4:0] = rs_ID_DUMMY[4:0];
    assign rt_data_EX[63:0] = rt_data_EX_DUMMY[63:0];
    assign rt_data_ID[63:0] = rt_data_ID_DUMMY[63:0];
    assign rt_data_M[63:0] = rt_data_M_DUMMY[63:0];
    assign rt_EX[4:0] = rt_EX_DUMMY[4:0];
    assign rt_ID[4:0] = rt_ID_DUMMY[4:0];
    assign rt_M[4:0] = rt_M_DUMMY[4:0];
    assign rt_WB[2:0] = rt_WB_DUMMY[2:0];
    assign SRAM_addr[7:0] = SRAM_addr_DUMMY[7:0];
    assign SUBI_EX = SUBI_EX_DUMMY;
    assign SUBI_ID = SUBI_ID_DUMMY;
    assign SUBI_M = SUBI_M_DUMMY;
    assign SUBI_WB = SUBI_WB_DUMMY;
    assign SW_EX = SW_EX_DUMMY;
    assign SW_ID = SW_ID_DUMMY;
    assign SW_M = SW_M_DUMMY;
    assign WME_EX = WME_EX_DUMMY;
    assign WME_M = WME_M_DUMMY;
    assign WME_M_out = WME_M_out_DUMMY;
    assign WP_en = WP_en_DUMMY;
    assign WRE_EX = WRE_EX_DUMMY;
    assign WRE_M = WRE_M_DUMMY;
    assign WRE_WB = WRE_WB_DUMMY;
    PC XLXI_1 (.clk(clk),
               .PC_next(PC_next_DUMMY[63:0]),
               .rst(rst),
               .PC(PC_DUMMY[63:0]));
    PC_plus_1 XLXI_2 (.ONE(ONE[63:0]),
                      .PC(PC_DUMMY[63:0]),
                      .PC_next(PC_plus_1_DUMMY[63:0]));
    PC_MUX XLXI_3 (.BTA(Offset_ID_DUMMY[63:0]),
                   .PC_ctrl(PC_ctrl_DUMMY),
                   .PC_next_in(PC_plus_1_DUMMY[63:0]),
                   .PC_next_out(PC_next_DUMMY[63:0]));
    I_MEM XLXI_7 (.addra(PC_DUMMY[8:0]),
                  .addrb(Instr_IN_addr[8:0]),
                  .clka(clk),
                  .clkb(clk),
                  .dinb(Instr_IN[31:0]),
                  .web(Instr_IN_en),
                  .douta(Instruction_IF_DUMMY[31:0]));
    IF_ID_Reg XLXI_8 (.Instruction(Instruction_IF_DUMMY[31:0]),
                      .Offset_ID(Offset_ID_before_Extend[15:0]),
                      .OP_CODE_ID(OP_CODE_ID_DUMMY[5:0]),
                      .rs_ID(rs_ID_DUMMY[4:0]),
                      .rt_ID(rt_ID_DUMMY[4:0]));
    Offset_Extend XLXI_9 (.Offset(Offset_ID_before_Extend[15:0]),
                          .Offset_ID(Offset_ID_DUMMY[63:0]));
    Control_Unit XLXI_10 (.OP_CODE(OP_CODE_ID_DUMMY[5:0]),
                          .ADDI_ID(ADDI_ID_DUMMY),
                          .ALU_OP_ID(ALU_OP_ID[3:0]),
                          .BEQ_ID(BEQ_ID_DUMMY),
                          .BGT_ID(BGT_ID_DUMMY),
                          .BLT_ID(BLT_ID_DUMMY),
                          .J_ID(J_ID_DUMMY),
                          .LW_ID(LW_ID_DUMMY),
```

```
                        .MOVI_ID(MOVI_ID_DUMMY),
                        .NOOP_ID(NOOP_ID_DUMMY),
                        .SUBI_ID(SUBI_ID_DUMMY),
                        .SW_ID(SW_ID_DUMMY),
                        .WME_ID(WME_ID),
                        .WRE_ID(WRE_ID));
    Branch_Detection_Unit XLXI_11 (.BEQ_ID(BEQ_ID_DUMMY),
                                   .BGT_ID(BGT_ID_DUMMY),
                                   .BLT_ID(BLT_ID_DUMMY),
                                   .J_ID(J_ID_DUMMY),
                                   .rs_data(rs_data_ID_DUMMY[63:0]),
                                   .rt_data(rt_data_ID_DUMMY[63:0]),
                                   .PC_ctrl(PC_ctrl_DUMMY));
    ID_EX_Reg XLXI_12 (.ADDI_ID(ADDI_ID_DUMMY),
                       .ALU_OP_ID(ALU_OP_ID[3:0]),
                       .clk(clk),
                       .LW_ID(LW_ID_DUMMY),
                       .MOVI_ID(MOVI_ID_DUMMY),
                       .NOOP_ID(NOOP_ID_DUMMY),
                       .Offset_ID(Offset_ID_DUMMY[63:0]),
                       .rst(rst),
                       .rs_data_ID(rs_data_ID_DUMMY[63:0]),
                       .rt_data_ID(rt_data_ID_DUMMY[63:0]),
                       .rt_ID(rt_ID_DUMMY[4:0]),
                       .SUBI_ID(SUBI_ID_DUMMY),
                       .SW_ID(SW_ID_DUMMY),
                       .WME_ID(WME_ID),
                       .WRE_ID(WRE_ID),
                       .ADDI_EX(ADDI_EX_DUMMY),
                       .ALU_OP_EX(ALU_OP_EX_DUMMY[3:0]),
                       .LW_EX(LW_EX_DUMMY),
                       .MOVI_EX(MOVI_EX_DUMMY),
                       .NOOP_EX(NOOP_EX_DUMMY),
                       .Offset_EX(Offset_EX_DUMMY[63:0]),
                       .rs_data_EX(rs_data_EX_DUMMY[63:0]),
                       .rt_data_EX(rt_data_EX_DUMMY[63:0]),
                       .rt_EX(rt_EX_DUMMY[4:0]),
                       .SUBI_EX(SUBI_EX_DUMMY),
                       .SW_EX(SW_EX_DUMMY),
                       .WME_EX(WME_EX_DUMMY),
                       .WRE_EX(WRE_EX_DUMMY));
    RF XLXI_15 (.clk(clk),
                .rst(rst),
                .r0addr(rs_ID_DUMMY[2:0]),
                .r1addr(rt_ID_DUMMY[2:0]),
                .waddr(rt_WB_DUMMY[2:0]),
                .wdata(RF_Din_DUMMY[63:0]),
                .wena(WRE_WB_DUMMY),
                .r0data(rs_data_ID_DUMMY[63:0]),
                .r1data(rt_data_ID_DUMMY[63:0]));
    ALU_src_MUX XLXI_16 (.ADDI_EX(ADDI_EX_DUMMY),
                         .LW_EX(LW_EX_DUMMY),
                         .Offset_EX(Offset_MUX_out_DUMMY[63:0]),
                         .rt_data(rt_data_EX_DUMMY[63:0]),
                         .SUBI_EX(SUBI_EX_DUMMY),
                         .SW_EX(SW_EX_DUMMY),
```

```
                        .ALU_B(ALU_B_in_DUMMY[63:0]));
    ALU XLXI_17 (.A(rs_data_EX_DUMMY[63:0]),
                 .ALU_OP(ALU_OP_EX_DUMMY[3:0]),
                 .B(ALU_B_in_DUMMY[63:0]),
                 .ALU_Out(ALU_result_EX_DUMMY[63:0]),
                 .Overflow(),
                 .Zero_Flag());
    EX_M_Reg XLXI_18 (.ADDI_EX(ADDI_EX_DUMMY),
                      .ALU_result_EX(ALU_result_EX_DUMMY[63:0]),
                      .clk(clk),
                      .LW_EX(LW_EX_DUMMY),
                      .MOVI_EX(MOVI_EX_DUMMY),
                      .NOOP_EX(NOOP_EX_DUMMY),
                      .Offset_EX(Offset_EX_DUMMY[63:0]),
                      .rst(rst),
                      .rs_data_EX(rs_data_EX_DUMMY[63:0]),
                      .rt_data_EX(rt_data_EX_DUMMY[63:0]),
                      .rt_EX(rt_EX_DUMMY[4:0]),
                      .SUBI_EX(SUBI_EX_DUMMY),
                      .SW_EX(SW_EX_DUMMY),
                      .WME_EX(WME_EX_DUMMY),
                      .WRE_EX(WRE_EX_DUMMY),
                      .ADDI_M(ADDI_M_DUMMY),
                      .ALU_result_M(ALU_result_M_DUMMY[63:0]),
                      .LW_M(LW_M_DUMMY),
                      .MOVI_M(MOVI_M_DUMMY),
                      .NOOP_M(NOOP_M_DUMMY),
                      .Offset_M(Offset_M_DUMMY[63:0]),
                      .rs_data_M(rs_data_M_DUMMY[63:0]),
                      .rt_data_M(rt_data_M_DUMMY[63:0]),
                      .rt_M(rt_M_DUMMY[4:0]),
                      .SUBI_M(SUBI_M_DUMMY),
                      .SW_M(SW_M_DUMMY),
                      .WME_M(WME_M_DUMMY),
                      .WRE_M(WRE_M_DUMMY));
    HELN_Reg XLXI_19 (.clk(clk),
                      .HLEN_in(HLEN[63:0]),
                      .HLEN_Reg_write_en(WP_en_DUMMY),
                      .rst_FIFO(rst_FIFO),
                      .HLEN_out(HLEN_out_DUMMY[63:0]));
    HLEN_Offset_Adder XLXI_21 (.HLEN(HLEN_out_DUMMY[63:0]),
                               .Offset(Offset_EX_DUMMY[63:0]),
                               .result(HLEN_Offset_sum_DUMMY[63:0]));
    Offset_MUX XLXI_22 (.HLEN_Offset_Adder_result(HLEN_Offset_sum_DUMMY[63:0]),
                        .LW_EX(LW_EX_DUMMY),
                        .Offset_EX(Offset_EX_DUMMY[63:0]),
                        .SW_EX(SW_EX_DUMMY),
                        .ALU_src_MUX_in(Offset_MUX_out_DUMMY[63:0]));
    D_MEM XLXI_24 (.addra(D_waddr_DUMMY[7:0]),
                   .addrb(D_raddr_DUMMY[7:0]),
                   .clka(clk),
                   .clkb(clk),
                   .dina(D_wdata_DUMMY[63:0]),
                   .wea(WME_M_out_DUMMY),
                   .doutb(pkt_out_DUMMY[63:0]));
    D_addr_src_MUX XLXI_25 (.ALU_result_M(ALU_result_M_DUMMY[63:0]),
```

```
                        .LW_M(LW_M_DUMMY),
                        .rt_M(rt_M_DUMMY[4:0]),
                        .SW_M(SW_M_DUMMY),
                        .D_addr(D_addr_src_MUX_out_DUMMY[7:0]));
WP_Reg XLXI_26 (.clk(clk),
                .FIFO_FULL(FIFO_FULL_DUMMY),
                .rst(rst_FIFO),
                .WP_en(WP_en_DUMMY),
                .WP_next(WP_next[7:0]),
                .WP(WP[7:0]));
WP_Adder XLXI_27 (.WP(WP[7:0]),
                  .WP_next(WP_next[7:0]));
WP_addr_MUX XLXI_28 (.SRAM_addr(SRAM_addr_DUMMY[7:0]),
                     .WP(WP[7:0]),
                     .WP_ctrl(WP_en_DUMMY),
                     .D_waddr(D_waddr_DUMMY[7:0]));
WP_Controller XLXI_29 (.mode_code(mode_code[1:0]),
                       .WP_en(WP_en_DUMMY));
RP_Reg XLXI_30 (.clk(clk),
                .FIFO_EMPTY(FIFO_EMPTY_DUMMY),
                .RP_en(RP_en_DUMMY),
                .RP_next(RP_next[7:0]),
                .rst(rst_FIFO),
                .RP(RP[7:0]));
RP_addr_MUX XLXI_31 (.RP(RP[7:0]),
                     .RP_ctrl(RP_en_DUMMY),
                     .SRAM_addr(SRAM_addr_DUMMY[7:0]),
                     .D_raddr(D_raddr_DUMMY[7:0]));
RP_Controller XLXI_32 (.mode_code(mode_code[1:0]),
                       .RP_en(RP_en_DUMMY));
RP_Adder XLXI_33 (.RP(RP[7:0]),
                  .RP_next(RP_next[7:0]));
WME_OR XLXI_34 (.WME_M(WME_M_DUMMY),
                .WP_en(WP_en_DUMMY),
                .WME(WME_M_out_DUMMY));
M_WB_Reg XLXI_37 (.ADDI_M(ADDI_M_DUMMY),
                  .ALU_result_M(ALU_result_M_DUMMY[63:0]),
                  .clk(clk),
                  .D_out_M(pkt_out_DUMMY[63:0]),
                  .LW_M(LW_M_DUMMY),
                  .MOVI_M(MOVI_M_DUMMY),
                  .NOOP_M(NOOP_M_DUMMY),
                  .Offset_M(Offset_M_DUMMY[63:0]),
                  .rst(rst),
                  .rs_data_M(rs_data_M_DUMMY[63:0]),
                  .rt_data_M(rt_data_M_DUMMY[63:0]),
                  .rt_M(rt_M_DUMMY[4:0]),
                  .SUBI_M(SUBI_M_DUMMY),
                  .SW_M(SW_M_DUMMY),
                  .WRE_M(WRE_M_DUMMY),
                  .ADDI_WB(ADDI_WB_DUMMY),
                  .ALU_result_WB(ALU_result_WB_DUMMY[63:0]),
                  .D_out_WB(Dout_WB_DUMMY[63:0]),
                  .LW_WB(LW_WB_DUMMY),
                  .MOVI_WB(MOVI_WB_DUMMY),
                  .NOOP_WB(NOOP_WB),
```

```
                    .Offset_WB(Offset_WB_DUMMY[63:0]),
                    .rs_data_WB(rs_data_WB[63:0]),
                    .rt_data_WB(rt_data_WB[63:0]),
                    .rt_WB(rt_WB_DUMMY[2:0]),
                    .SUBI_WB(SUBI_WB_DUMMY),
                    .SW_WB(SW_WB),
                    .WRE_WB(WRE_WB_DUMMY));
    SRAM_addr_Adder XLXI_38 (.D_addr(D_addr_src_MUX_out_DUMMY[7:0]),
                             .RP(RP[7:0]),
                             .D_addr_in(SRAM_addr_DUMMY[7:0]));
    WP_Data_MUX XLXI_39 (.FIFO_Din(pkt_in[63:0]),
                         .SRAM_Din(rt_data_M_DUMMY[63:0]),
                         .WP_ctrl(WP_en_DUMMY),
                         .D_MEM_Din(D_wdata_DUMMY[63:0]));
    RF_WB_data_src_MUX XLXI_40 (.ADDI_WB(ADDI_WB_DUMMY),
                                .ALU_out_WB(ALU_result_WB_DUMMY[63:0]),
                                .D_out_WB(Dout_WB_DUMMY[63:0]),
                                .LW_WB(LW_WB_DUMMY),
                                .MOVI_WB(MOVI_WB_DUMMY),
                                .Offset_WB(Offset_WB_DUMMY[63:0]),
                                .SUBI_WB(SUBI_WB_DUMMY),
                                .RF_WB_Din(RF_Din_DUMMY[63:0]));
    FIFO_state_controller XLXI_41 (.clk(clk),
                                   .RP(RP[7:0]),
                                   .rst_FIFO(rst_FIFO),
                                   .WP(WP[7:0]),
                                   .depth(FIFO_depth[7:0]),
                                   .FIFO_almost_empty(FIFO_almost_empty),
                                   .FIFO_almost_full(FIFO_almost_full),
                                   .FIFO_EMPTY(FIFO_EMPTY_DUMMY),
                                   .FIFO_FULL(FIFO_FULL_DUMMY));
endmodule
```

- Testbench

```
`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:   19:29:13 03/08/2025
// Design Name:   Pipeline_demo
// Module Name:   E:/Documents and
Settings/student/EE533_Lab8/EE533_Lab_8/Pipeline_demo_tb.v
// Project Name:  EE533_Lab_8
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: Pipeline_demo
//
// Dependencies:
//
// Revision:
```

```
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

module Pipeline_demo_tb;

    // Inputs
    reg clk;
    reg [63:0] HLEN;
    reg [31:0] Instr_IN;
    reg [8:0] Instr_IN_addr;
    reg Instr_IN_en;
    reg [1:0] mode_code;
    reg [63:0] ONE;
    reg [63:0] pkt_in;
    reg rst;
    reg rst_FIFO;

    // Outputs
    wire ADDI_EX;
    wire ADDI_ID;
    wire ADDI_M;
    wire ADDI_WB;
    wire [63:0] ALU_B_in;
    wire [3:0] ALU_OP_EX;
    wire [63:0] ALU_result_EX;
    wire [63:0] ALU_result_M;
    wire [63:0] ALU_result_WB;
    wire BEQ_ID;
    wire BGT_ID;
    wire BLT_ID;
    wire [63:0] Dout_WB;
    wire [7:0] D_addr_src_MUX_out;
    wire [7:0] D_raddr;
    wire [7:0] D_waddr;
    wire [63:0] D_wdata;
    wire FIFO_almost_empty;
    wire FIFO_almost_full;
    wire [7:0] FIFO_depth;
    wire FIFO_EMPTY;
    wire FIFO_FULL;
    wire [63:0] HLEN_Offset_sum;
    wire [63:0] HLEN_out;
    wire [31:0] Instruction_IF;
    wire J_ID;
    wire LW_EX;
    wire LW_ID;
    wire LW_M;
    wire LW_WB;
    wire MOVI_EX;
    wire MOVI_ID;
    wire MOVI_M;
    wire MOVI_WB;
    wire NOOP_EX;
    wire NOOP_ID;
```

```
    wire NOOP_M;
    wire NOOP_WB;
    wire [63:0] Offset_EX;
    wire [63:0] Offset_ID;
    wire [63:0] Offset_M;
    wire [63:0] Offset_MUX_out;
    wire [63:0] Offset_WB;
    wire [5:0] OP_CODE_ID;
    wire [63:0] PC;
    wire [63:0] PC_next;
    wire [63:0] pkt_out;
    wire [63:0] RF_Din;
    wire RP_en;
    wire [63:0] rs_data_EX;
    wire [63:0] rs_data_ID;
    wire [63:0] rs_data_M;
    wire [63:0] rs_data_WB;
    wire [4:0] rs_ID;
    wire [63:0] rt_data_EX;
    wire [63:0] rt_data_ID;
    wire [63:0] rt_data_M;
    wire [63:0] rt_data_WB;
    wire [4:0] rt_EX;
    wire [4:0] rt_ID;
    wire [4:0] rt_M;
    wire [2:0] rt_WB;
    wire [7:0] SRAM_addr;
    wire SUBI_EX;
    wire SUBI_ID;
    wire SUBI_M;
    wire SUBI_WB;
    wire SW_EX;
    wire SW_ID;
    wire SW_M;
    wire SW_WB;
    wire WME_EX;
    wire WME_M;
    wire WME_M_out;
    wire WP_en;
    wire WRE_EX;
    wire WRE_M;
    wire WRE_WB;

    // Instantiate the Unit Under Test (UUT)
    Pipeline_demo uut (
        .clk(clk),
        .HLEN(HLEN),
        .Instr_IN(Instr_IN),
        .Instr_IN_addr(Instr_IN_addr),
        .Instr_IN_en(Instr_IN_en),
        .mode_code(mode_code),
        .ONE(ONE),
        .pkt_in(pkt_in),
        .rst(rst),
        .rst_FIFO(rst_FIFO),
        .ADDI_EX(ADDI_EX),
```

```
        .ADDI_ID(ADDI_ID),
        .ADDI_M(ADDI_M),
        .ADDI_WB(ADDI_WB),
        .ALU_B_in(ALU_B_in),
        .ALU_OP_EX(ALU_OP_EX),
        .ALU_result_EX(ALU_result_EX),
        .ALU_result_M(ALU_result_M),
        .ALU_result_WB(ALU_result_WB),
        .BEQ_ID(BEQ_ID),
        .BGT_ID(BGT_ID),
        .BLT_ID(BLT_ID),
        .Dout_WB(Dout_WB),
        .D_addr_src_MUX_out(D_addr_src_MUX_out),
        .D_raddr(D_raddr),
        .D_waddr(D_waddr),
        .D_wdata(D_wdata),
        .FIFO_almost_empty(FIFO_almost_empty),
        .FIFO_almost_full(FIFO_almost_full),
        .FIFO_depth(FIFO_depth),
        .FIFO_EMPTY(FIFO_EMPTY),
        .FIFO_FULL(FIFO_FULL),
        .HLEN_Offset_sum(HLEN_Offset_sum),
        .HLEN_out(HLEN_out),
        .Instruction_IF(Instruction_IF),
        .J_ID(J_ID),
        .LW_EX(LW_EX),
        .LW_ID(LW_ID),
        .LW_M(LW_M),
        .LW_WB(LW_WB),
        .MOVI_EX(MOVI_EX),
        .MOVI_ID(MOVI_ID),
        .MOVI_M(MOVI_M),
        .MOVI_WB(MOVI_WB),
        .NOOP_EX(NOOP_EX),
        .NOOP_ID(NOOP_ID),
        .NOOP_M(NOOP_M),
        .NOOP_WB(NOOP_WB),
        .Offset_EX(Offset_EX),
        .Offset_ID(Offset_ID),
        .Offset_M(Offset_M),
        .Offset_MUX_out(Offset_MUX_out),
        .Offset_WB(Offset_WB),
        .OP_CODE_ID(OP_CODE_ID),
        .PC(PC),
        .PC_next(PC_next),
        .pkt_out(pkt_out),
        .RF_Din(RF_Din),
        .RP_en(RP_en),
        .rs_data_EX(rs_data_EX),
        .rs_data_ID(rs_data_ID),
        .rs_data_M(rs_data_M),
        .rs_data_WB(rs_data_WB),
        .rs_ID(rs_ID),
        .rt_data_EX(rt_data_EX),
        .rt_data_ID(rt_data_ID),
        .rt_data_M(rt_data_M),
```

```
        .rt_data_WB(rt_data_WB),
        .rt_EX(rt_EX),
        .rt_ID(rt_ID),
        .rt_M(rt_M),
        .rt_WB(rt_WB),
        .SRAM_addr(SRAM_addr),
        .SUBI_EX(SUBI_EX),
        .SUBI_ID(SUBI_ID),
        .SUBI_M(SUBI_M),
        .SUBI_WB(SUBI_WB),
        .SW_EX(SW_EX),
        .SW_ID(SW_ID),
        .SW_M(SW_M),
        .SW_WB(SW_WB),
        .WME_EX(WME_EX),
        .WME_M(WME_M),
        .WME_M_out(WME_M_out),
        .WP_en(WP_en),
        .WRE_EX(WRE_EX),
        .WRE_M(WRE_M),
        .WRE_WB(WRE_WB)
    );

    always #50 clk = ~clk;

    initial begin
        // Initialize Inputs
        clk = 1;
        HLEN = 64'd3;
        Instr_IN = 0;
        Instr_IN_addr = 0;
        Instr_IN_en = 0;
        mode_code = 2'b11;
        ONE = 64'd1;
        pkt_in = 0;
        rst = 1;
        rst_FIFO = 1;

        // Wait 100 ns for global reset to finish
        @(posedge clk);
        rst_FIFO = 0;

        // Add stimulus here
        @(posedge clk);
        mode_code = 2'b00;
        pkt_in = 64'h4600004000001C46;

        @(posedge clk);
        mode_code = 2'b00;
        pkt_in = 64'h4000040600000A00;

        @(posedge clk);
        mode_code = 2'b00;
        pkt_in = 64'h0D030A000E030000;

        @(posedge clk);
```

```
        mode_code = 2'b00;
        pkt_in = 64'h0000000000000003;

        @(posedge clk);
        mode_code = 2'b00;
        pkt_in = 64'h0000000000000004;

        @(posedge clk);
        mode_code = 2'b00;
        pkt_in = 64'h0000000000000003;

        @(posedge clk);
        mode_code = 2'b00;
        pkt_in = 64'h0000000000000002;

        @(posedge clk);
        mode_code = 2'b00;
        pkt_in = 64'h0000000000000001;

        @(posedge clk);
        mode_code = 2'b10;
        rst = 0;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;
```

```
        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;
```

```
        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
```

```
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;
```

```
        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;
```

```
        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
```

```
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;
```

```
        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;
```

```
        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
```

```
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b10;
```

```
        @(posedge clk);
        mode_code = 2'b10;

        @(posedge clk);
        mode_code = 2'b01;

        @(posedge clk);
        mode_code = 2'b01;

        @(posedge clk);
        mode_code = 2'b01;

        @(posedge clk);
        mode_code = 2'b01;

        @(posedge clk);
        mode_code = 2'b01;

        @(posedge clk);
        mode_code = 2'b01;

        @(posedge clk);
        mode_code = 2'b01;

        @(posedge clk);
        mode_code = 2'b01;

        @(posedge clk);
        mode_code = 2'b01;

        @(posedge clk);

        @(posedge clk);
        $stop;

    end

endmodule
```

5. Script File

5.1 Write Script

```
#!/usr/bin/perl
use strict;
use warnings;
use Time::HiRes qw(usleep);

# Register Addresses
my $ADDR_REG  = "0x2001208";  # PIPLINE_ADDR_REG_REG
my $FLAG_REG = "0x2001204";  # PIPLINE_STATUS_REG
my $CMD_REG   = "0x2001200";  # PIPLINE_CMD_REG_REG


my $WRITE_FLAG = 0x0;
my $POLL_DELAY = 100;
my $WORD_SIZE = 4;
```

```
sub regwrite {
    my ($addr, $value) = @_;
    my $cmd = sprintf("regwrite %s 0x%08x", $addr, $value);
    my $result = `$cmd`;
    # print "CMD: $cmd\nRESULT: $result";
}

sub regread {
    my ($addr) = @_;
    my $cmd = sprintf("regread %s", $addr);
    my @out = `$cmd`;
    return "0x00000000" unless @out;  #error

    my $result = $out[0];
    if ($result =~ m/Reg (0x[0-9a-f]+) \((\d+)\):\s+(0x[0-9a-f]+) \((\d+)\)/) {
        return $3;
    }
    return "0x00000000";
}


die "Usage: $0 <instruction_file>\n" unless @ARGV;
my $filename = $ARGV[0];

open(my $fh, '<', $filename) or die "Can't open $filename: $!";

my $current_addr = 0;
while (my $line = <$fh>) {
    chomp $line;
    next if $line =~ /^\s*(#|$)/;

    # Parse instruction (support both "0x12345678" and "12345678" formats)
    my $instruction = $line;
    $instruction = "0x$instruction" unless $line =~ /^0x/i;
    $instruction = hex($instruction);

    # Write address register
    regwrite($ADDR_REG, $current_addr);

    # Write data register
    regwrite($WDATA_REG, $instruction);

    # Trigger write operation
    regwrite($CMD_REG, $WRITE_FLAG);

    # Poll until operation completes
    while (1) {
        my $status = hex(regread($FLAG_REG_REG));
        last unless ($status & 0x1);  # Exit when start bit cleared
        usleep($POLL_DELAY);
    }

    $current_addr += $WORD_SIZE;  # Increment address
}
```

```
close($fh);
print "Loaded ".($current_addr/$WORD_SIZE)." instructions\n";
```

5.2 Read Script

```
#!/usr/bin/perl
use strict;
use warnings;

my $HI_REG = "0x200020c"; # PIPLINE_RDATA_REG_HI
my $LO_REG = "0x2000210"; # PIPLINE_RDATA_REG_LO

sub regread {
    my ($addr) = @_;
    my $cmd = sprintf("regread %s", $addr);
    my @out = `$cmd`;
    return "0x00000000" unless @out;

    my $result = $out[0];
    if ($result =~ m/Reg (0x[0-9a-f]+) \((\d+)\):\s+(0x[0-9a-f]+) \((\d+)\)/) {
        return $3;
    }
    return "0x00000000";
}

my $hi_value = regread($HI_REG);
my $lo_value = regread($LO_REG);

print "HI_REG ($HI_REG): $hi_value\n";
print "LO_REG ($LO_REG): $lo_value\n";
```

6. Passthrough

- Verilog

```
`timescale 1ns/1ps


module passthrough
   #(
      parameter DATA_WIDTH = 64,
      parameter CTRL_WIDTH = DATA_WIDTH/8,
      parameter UDP_REG_SRC_WIDTH = 2
   )
    (

      input  [DATA_WIDTH-1:0]             in_data,
      input  [CTRL_WIDTH-1:0]             in_ctrl,
      input                               in_wr,
      output                              in_rdy,

      output [DATA_WIDTH-1:0]             out_data,
      output [CTRL_WIDTH-1:0]             out_ctrl,
```

```
      output                                  out_wr,
      input                                   out_rdy,
      // --- Register interface
      input                                   reg_req_in,
      input                                   reg_ack_in,
      input                                   reg_rd_wr_L_in,
      input  [`UDP_REG_ADDR_WIDTH-1:0]        reg_addr_in,
      input  [`CPCI_NF2_DATA_WIDTH-1:0]       reg_data_in,
      input  [UDP_REG_SRC_WIDTH-1:0]          reg_src_in,

      output                                  reg_req_out,
      output                                  reg_ack_out,
      output                                  reg_rd_wr_L_out,
      output  [`UDP_REG_ADDR_WIDTH-1:0]       reg_addr_out,
      output  [`CPCI_NF2_DATA_WIDTH-1:0]      reg_data_out,
      output  [UDP_REG_SRC_WIDTH-1:0]         reg_src_out,

      // misc
      input                                    reset,
      input                                    clk
    );

   reg [31:0]   hw_reg0;         //hardware register
   reg [31:0]   addr_reg;


   reg [1:0]    DRAM_MODE;        //01 for DRAM, 00 for fifo recieve,  10 for
fifo send

   wire [31:0]    CMD_reg;        //software register
   wire [31:0]    flag_reg;

   wire [31:0]    CMD_GET_IN;


   wire [31:0]    inst_out;       //wires
   wire [8:0]     addr_out;
   wire           write_enable;
   wire           pipline_rst;







   generic_regs
   #(
    .UDP_REG_SRC_WIDTH   (UDP_REG_SRC_WIDTH),
    .TAG                 (`PASSTHROUGH_BLOCK_ADDR_WIDTH ),          // Tag -- eg.
MODULE_TAG
    .REG_ADDR_WIDTH      (`PASSTHROUGH_REG_ADDR_WIDTH),
    .NUM_COUNTERS        (0),
    .NUM_SOFTWARE_REGS   (2),
    .NUM_HARDWARE_REGS   (2)
   )module_regs(
```

```
    .reg_req_in       (reg_req_in),
    .reg_ack_in       (reg_ack_in),
    .reg_rd_wr_L_in   (reg_rd_wr_L_in),
    .reg_addr_in      (reg_addr_in),
    .reg_data_in      (reg_data_in),
    .reg_src_in       (reg_src_in),

    .reg_req_out      (reg_req_out),
    .reg_ack_out      (reg_ack_out),
    .reg_rd_wr_L_out  (reg_rd_wr_L_out),
    .reg_addr_out     (reg_addr_out),
    .reg_data_out     (reg_data_out),
    .reg_src_out      (reg_src_out),

      // --- counters interface
      .counter_updates  (),
      .counter_decrement(),

    .software_regs       ({CMD_reg,flag_reg}),
    .hardware_regs       ({hw_reg0,addr_reg}),
    .clk                 (clk),
    .reset               (reset)

  );
  instruction_fsm fsm_inst(

    .clk                (clk),
    .rst_n              (reset),
    .addr_in            (addr_reg[8:0]),
    .inst_in            (CMD_GET_IN),
    .inst_out           (inst_out),
    .addr_out           (addr_out),
    .write_enable       (write_enable),
    .flag_ready         (flag_reg[2]),
    .flag_reg           (flag_reg)
  )


    assign out_data = in_data;
    assign out_ctrl =in_ctrl;
    assign out_wr = in_wr;
    assign in_rdy = out_rdy;


    typedef enum logic[2:0]{
        start = 3'b000
        Packet_Rec = 3'b001,
        Inst_INIT = 3'b010,
        Process = 3'b011,
        Packet_send = 3'b100
    } state_t;

    state_t state, next_state;

    always @(posedge clk) begin
      case (state)
```

```
        start:
            next_state = Packet_Rec;

        Packet_Rec:
          if(DRAM_MODE == 2'b01)
            next_state = Inst_INIT;
          else
            next_state = Packet_Rec;

        Inst_INIT:
          if(flag_reg[3])
            next_state = Process;
          else
            next_state = Inst_INIT;

        Process:
          if(DRAM_MODE == 2'b10)
              next_state = Packet_send;
          else
              next_state = Process;

        Packet_send:
          if(flag_reg[4])
            next_state = Packet_Rec;
          else
            next_state = Packet_send;
      endcase
    end

     always @(posedge clk) begin
        case (state)
          start: begin
          end


          Packet_Rec: begin
            DRAM_MODE = 2'b00;
          end

          Inst_INIT: begin
              DRAM_MODE = 2'B01;
              CMD_GET_IN <= CMD_reg;
              pipline_rst <= 1'b1;
              //pipline_mode_input = DRAM_MODE

          end

          Process: begin
              pipline_rst <= 1'b0;
          end

          Packet_send: begin

          end
        endcase
```

```
    end

endmodule
```

7. GitHub Link

- https://github.com/yuezhenglingluan/USC_EE533_lab8.git