**PIPLINE PROCESSOR DESIGN SIMULATION REPORT**

1. **TITLE:**

4-Stage Pipeline Processor

1. **Objective:**

To design a 4-stage pipelined processor with basic instructions like add, sub and load.

This report represents the design, a Verilog code, testbench code, waveform analysis, simulation report showing each stage’s operation.

**3. Pipeline Architecture**

The processor is designed with the following four stages:

1. **Instruction Fetch (IF):** Fetches the instruction from memory.
2. **Instruction Decode (ID):** Decodes the fetched instruction and reads source registers.
3. **Execution (EX):** Performs arithmetic operations (ADD, SUB) or calculates memory addresses for LOAD instructions.
4. **Write Back (WB):** Writes the results back to the destination register.

**4. Instruction Set Architecture (ISA)**

* **ADD Rd, Rs, Rt:** Adds contents of Rs and Rt, stores in Rd.
* **SUB Rd, Rs, Rt:** Subtracts Rt from Rs, stores in Rd.
* **LOAD Rd, [address]:** Loads data from memory address into Rd.

**5. Functional Design**

* **Data Path:** Includes ALU, register file, instruction memory, and data memory.
* **Control Unit:** Generates control signals for each instruction type.
* **Pipeline Registers:** Store intermediate data between stages (IF/ID, ID/EX, EX/WB).

**6. Simulation Results**

* **Instruction Fetch Stage:** Shows correct instruction retrieval from memory.
* **Instruction Decode Stage:** Verifies proper decoding and operand fetching.
* **Execution Stage:** Demonstrates accurate arithmetic operations and address calculations.
* **Write Back Stage:** Confirms the correct update of registers with computed results.

Compilation Report

Flow Status Successful -Tue Feb 04 23:53:50 2025

Quartus Prime Version 20.1.1 Build 720 11/11/2020 SJ Lite Edition

Revision Name pipeline\_processor

Top-level Entity Name pipeline\_processor

Family Cyclone V

Device 5CGXFC7C7F23C8

Timing Models Final

Logic utilization (in ALMs) 112 / 56,480 (< 1 %)

Total registers 238

Total pins 34 / 268 (13 %)

Total virtual pins 0

Total block memory bits 512 / 7,024,640 (< 1 % )

Total DSP Blocks 0 / 156 ( 0 % )

Total HSSI RX PCSs 0 / 6 ( 0 % )

Total HSSI PMA RX Deserializers 0 / 6 ( 0 % )

Total HSSI TX PCSs 0 / 6 ( 0 % )

Total HSSI PMA TX Serializers 0 / 6 ( 0 % )

Total PLLs 0 / 13 ( 0 % )

Total DLLs 0 / 4 ( 0 % )

**Verilog code:**

module pipeline\_processor(

input clk,

input reset,

output [31:0] result

);

// Instruction Memory (4 instructions)

reg [31:0] instr\_mem [0:3];

// Register file (32 registers, 32 bits each)

reg [31:0] reg\_file [0:31];

// Pipeline registers for each stage

reg [31:0] IF\_instr, ID\_instr, EX\_instr, WB\_instr;

reg [31:0] IF\_PC, ID\_PC, EX\_PC, WB\_PC;

reg [31:0] ALU\_result, WB\_data;

// Initialize Instruction Memory

initial begin

instr\_mem[0] = 32'b000000\_00001\_00010\_00011\_00000\_100000; // ADD R3, R1, R2

instr\_mem[1] = 32'b000000\_00001\_00010\_00100\_00000\_100010; // SUB R4, R1, R2

instr\_mem[2] = 32'b100011\_00000\_00001\_00000\_00000\_000000; // LOAD R1, 0(R0)

instr\_mem[3] = 32'b000000\_00001\_00010\_00101\_00000\_100000; // ADD R5, R1, R2

end

// Instruction Fetch (IF) Stage

always @(posedge clk or posedge reset) begin

if (reset) begin

IF\_PC <= 0;

IF\_instr <= 32'b0;

end else begin

IF\_instr <= instr\_mem[IF\_PC];

IF\_PC <= IF\_PC + 1;

end

end

// Instruction Decode (ID) Stage

always @(posedge clk or posedge reset) begin

if (reset) begin

ID\_instr <= 32'b0;

ID\_PC <= 0;

end else begin

ID\_instr <= IF\_instr;

ID\_PC <= IF\_PC;

end

end

// Execute (EX) Stage

always @(posedge clk or posedge reset) begin

if (reset) begin

EX\_instr <= 32'b0;

EX\_PC <= 0;

ALU\_result <= 0;

end else begin

EX\_instr <= ID\_instr;

EX\_PC <= ID\_PC;

case (EX\_instr[31:26]) // Opcode

6'b000000: // R-type (ADD, SUB)

case (EX\_instr[5:0])

6'b100000: ALU\_result <= reg\_file[EX\_instr[25:21]] + reg\_file[EX\_instr[20:16]]; // ADD

6'b100010: ALU\_result <= reg\_file[EX\_instr[25:21]] - reg\_file[EX\_instr[20:16]]; // SUB

default: ALU\_result <= 0;

endcase

6'b100011: // LOAD

ALU\_result <= reg\_file[EX\_instr[25:21]] + EX\_instr[15:0]; // Address calculation

default: ALU\_result <= 0;

endcase

end

end

// Write Back (WB) Stage

always @(posedge clk or posedge reset) begin

if (reset) begin

WB\_instr <= 32'b0;

WB\_PC <= 0;

WB\_data <= 0;

end else begin

WB\_instr <= EX\_instr;

WB\_PC <= EX\_PC;

case (WB\_instr[31:26])

6'b000000: reg\_file[WB\_instr[15:11]] <= ALU\_result; // R-type

6'b100011: reg\_file[WB\_instr[20:16]] <= ALU\_result; // LOAD

default: ;

endcase

WB\_data <= ALU\_result;

end

end

// Output result

assign result = WB\_data;

endmodule

**Testbench code:**

module tb\_pipeline\_processor:

reg clk;

reg reset;

wire [31:0] result;

pipeline\_processor uut (

.clk(clk),

.reset(reset),

.result(result)

);

// Clock generation

always #5 clk = ~clk;

initial begin

clk = 0;

reset = 1;

#10 reset = 0;

#100;

$stop;

end

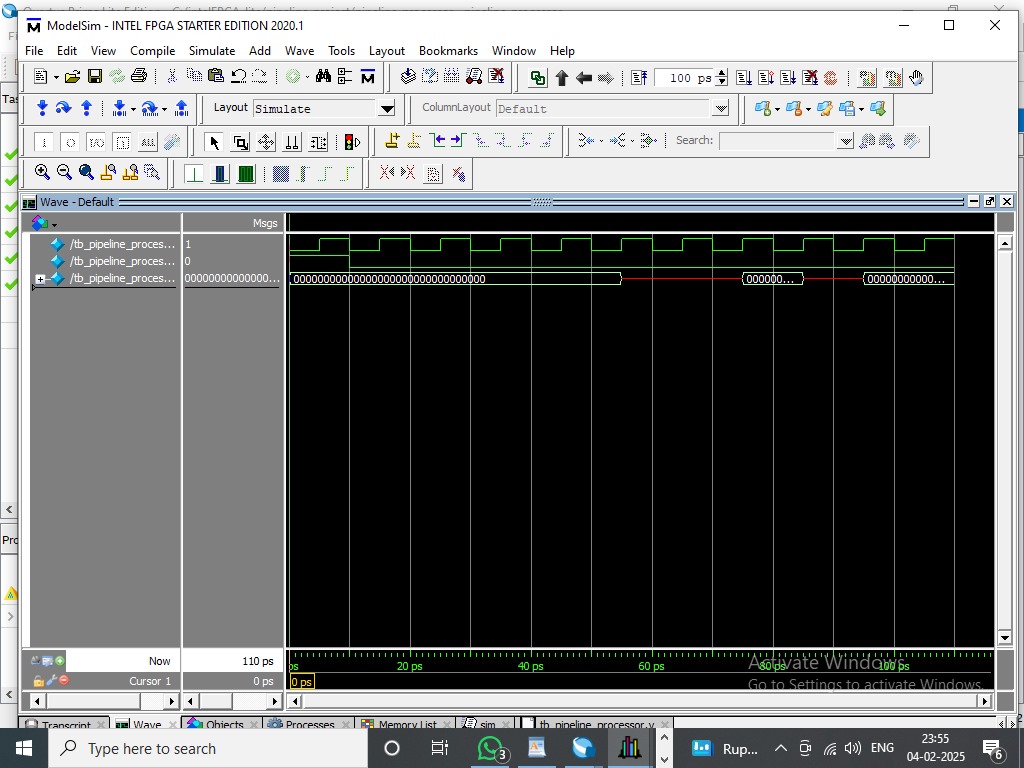
initial begin

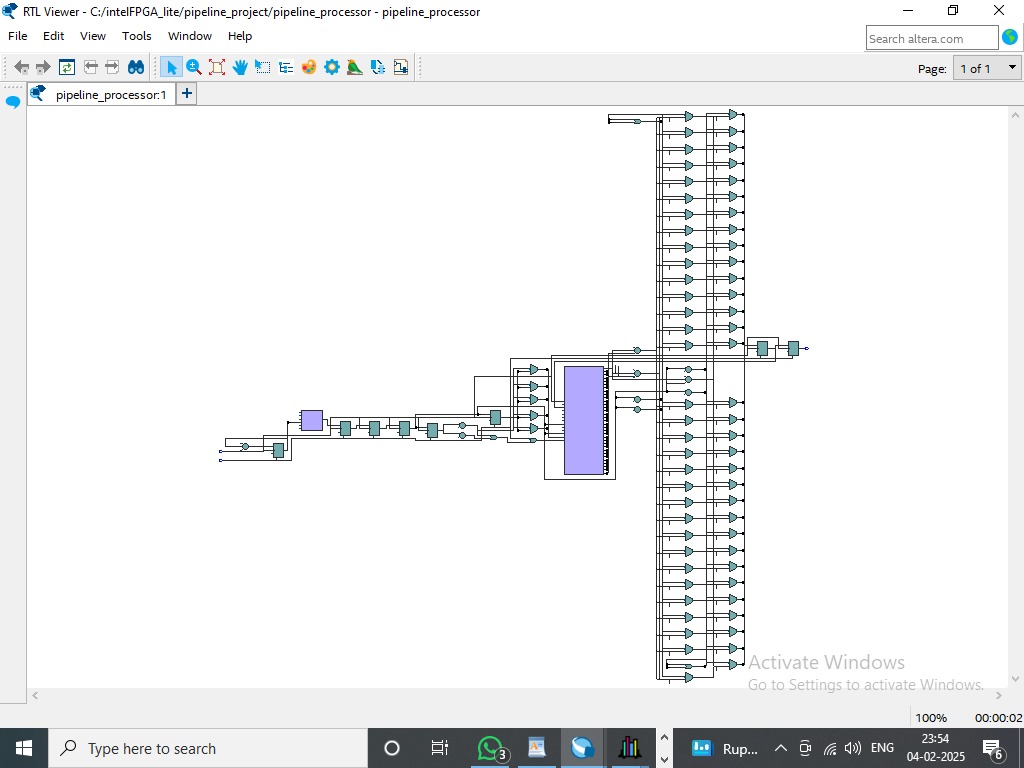
$monitor("Time: %t | Result: %d", $time, result);

end

endmodule

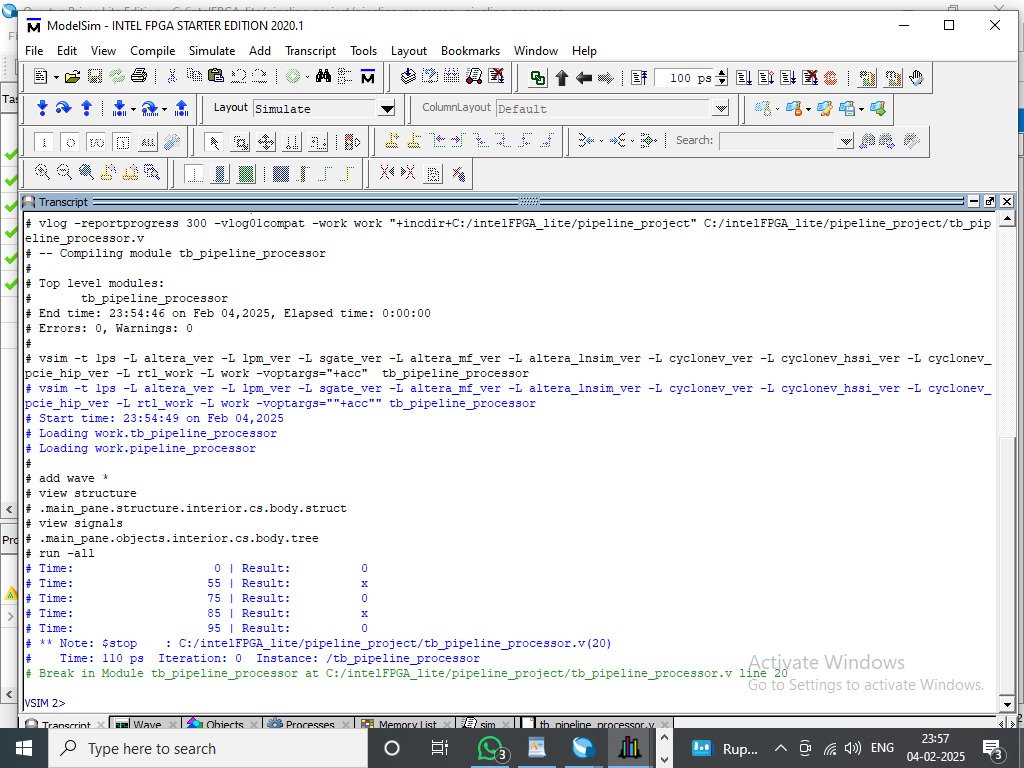
**Waveforms:**





**7. Performance Analysis**

* **Pipeline Efficiency:** Evaluated by measuring instruction throughput and identifying hazards.
* **Hazard Handling:** Discusses data hazards and control hazards, with implemented solutions (e.g., forwarding, stalling).



**8. Conclusion**

The 4-stage pipelined processor successfully executes basic instructions with correct stage-wise operations. The simulation results validate the functional design, showcasing efficient instruction processing.