**Shri Ramdeobaba College of Engineering and Management, Nagpur-13.   
Department of Electronics Engineering  
ENP360 (Computer Architecture and Organization lab.)  
Even Semester – 2023-24**

**Lab 01**

|  |  |
| --- | --- |
| **Name:** | Pawan Dilip Sorte |
| **Batch / Roll No.** | A1 / 12 |
| **Semester/Section:** | VI / A |
| **Date of Performance:** | 08/01/2024 |
| **Date of Submission:** | 15/04/2024 |
| **Name &**  **Signature of Faculty** | Prof. Sandeep Pandey |

**Lab-01**

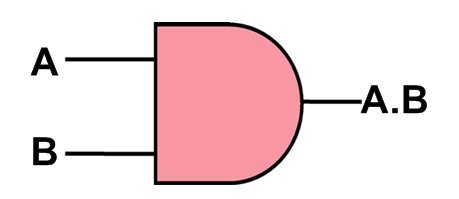
**Aim:**

Implementation of AND Gate and D Flip

**Software and Hardware Used:**

Xilinx Vivaldo 2016.2

**Diagram:**



**Theory:**

* Create a new project and specify the name, location, and target device. Right-click on the "Design Sources" section in the "Sources" tab.
* Select "Add Sources" -> "Add or create design sources" -> "Create File".
* Choose VHDL as the file type and specify the name. Follow the same steps as above to create a new VHDL file. Right-click on the "Design Sources" section in the "Sources" tab. Select "Add Sources" -> "Add or create design sources" -> "Add Files".
* Choose the files you created
* In the Flow Navigator panel, select "Synthesis".
* Click on "Run Synthesis".
* After synthesis is successful, select "Implementation" in the Flow Navigator.
* Click on "Run Implementation".
* Once the implementation is successful, select "Generate Bitstream" in the Flow Navigator.
* Click on "Generate Bitstream".
* After generating the bitstream, program the FPGA with the generated bitstream file.
* Verify the functionality of the AND gate and by applying appropriate inputs and observing the outputs.

**Design Source:**

**Source Code (For AND gate):**

`timescale 1ns / 1ps

module and\_gate(a,b,y);

input a;

input b;

output y;

and(y,a,b);

endmodule

**Test-bench Code (For AND gate):**

`timescale 1ns / 1ps

module and\_gate\_tb;

reg a;

reg b;

wire y;

and\_gate UUT (.a(a), .b(b), .y(y));

initial begin

$display("Testing AND gate");

a = 0; b=0;

#10;

$display("Input\_A = %b, Input\_B = %b, Output = %b", a,b,y);

a = 0; b=1;

#10;

$display("Input\_A = %b, Input\_B = %b, Output = %b", a,b,y);

a = 1; b=0;

#10;

$display("Input\_A = %b, Input\_B = %b, Output = %b", a,b,y);

a = 1; b=1;

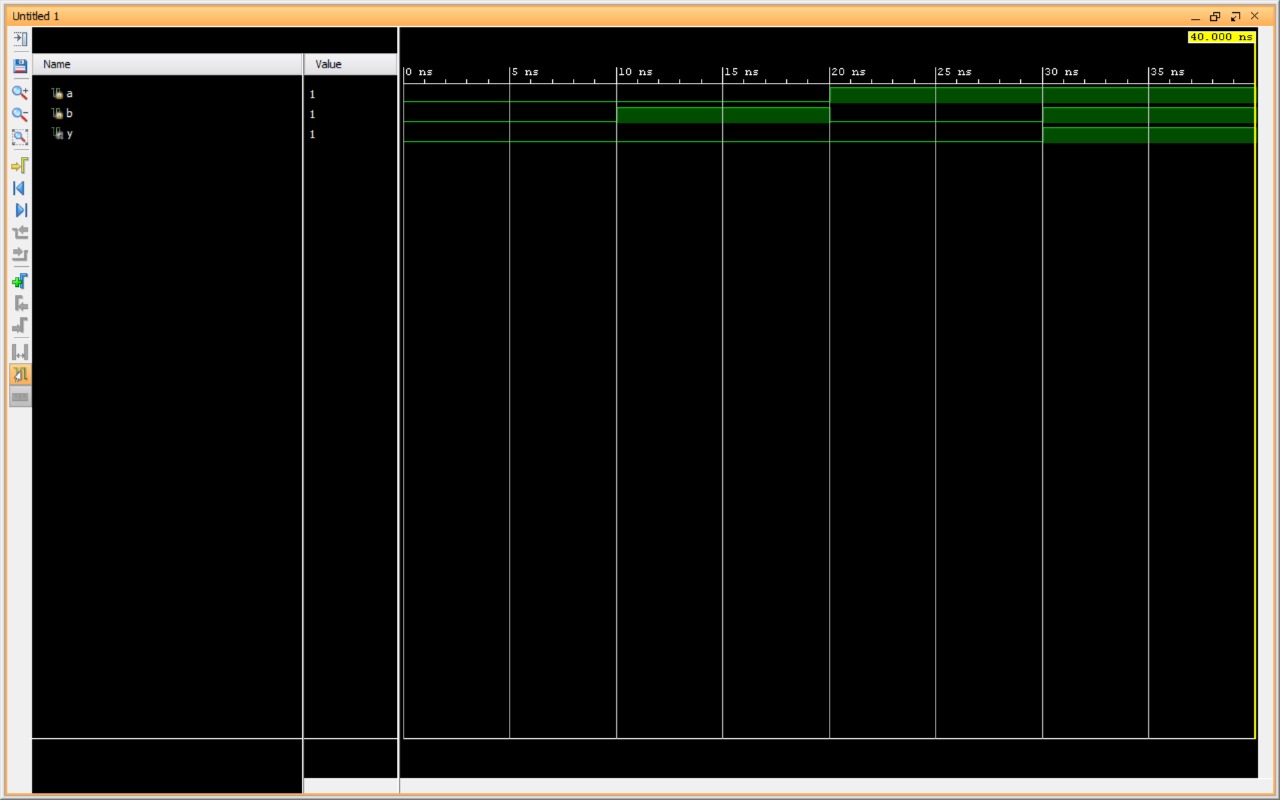
#10;

$display("Input\_A = %b, Input\_B = %b, Output = %b", a,b,y);

$finish;

end

endmodule

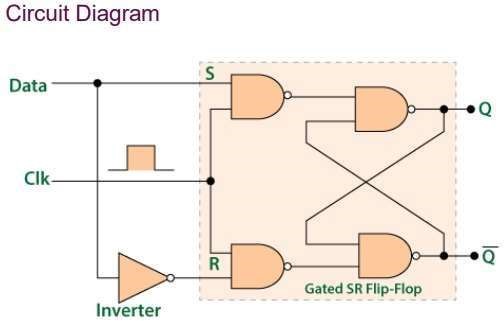
**Simulation Source:** 

**Result:**

**AND Gate:**

In VHDL, an AND gate implements the logical AND operation on its input signals. The output (y) of an AND gate is '1' only when both of its input signals (a and b) are '1'. If any of the inputs are '0', the output is '0'.

**Diagram (For D Flip-Flop):**



**Design Source:**

**Source Code (For D Flip-Flop):**

module D\_ff(Q,D,CLK,reset); input Q; input D; input CLK; output reg Q;

always@(posedge CLK) begin Q<=D; end endmodule

**Test-bench Code (For D Flip-Flop):**

module D\_ff\_TB; reg D; reg CLK; reg reset; wire Q;

D\_ff D1(Q,D,CLK,reset); initial begin

CLK=1'b0;

forever #20 CLK=~CLK; end reset=1'b1; #40;

reset=1'b0; #40; D=1'b0; #40; D=1'b1; #40;

$finish; end endmodule.

**Simulation Source:**



**Result:**

**D Flip-Flop:**

In VHDL, a D flip-flop stores its input signal (d) on the rising edge of the clock (clk) signal. The output (q) changes to the value of its input (d) when a rising clock edge occurs. If the reset signal is asserted ('1'), the output (q) is set to '0' regardless of the input (d). Otherwise, the output (q) retains its value.

**Shri Ramdeobaba College of Engineering and Management, Nagpur-13.   
Department of Electronics Engineering  
ENP360 (Computer Architecture and Organization lab.)  
Even Semester – 2023-24**

**Lab 02**

|  |  |
| --- | --- |
| **Name:** | Pawan Dilip Sorte |
| **Batch / Roll No.** | A1 / 12 |
| **Semester/Section:** | VI / A |
| **Date of Performance:** | 29/01/2024 |
| **Date of Submission:** | 15/04/2024 |
| **Name &**  **Signature of Faculty** | Prof. Sandeep Pandey |

**Lab-02**

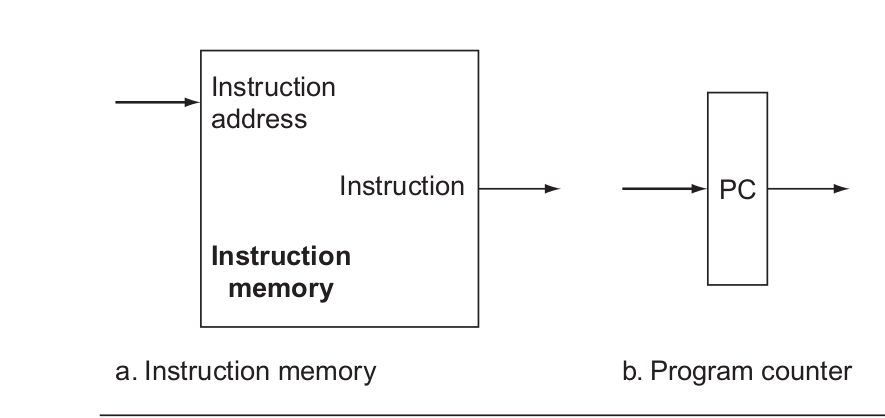
**Aim:**

Memory Design: Implementation of Instruction memory. Verify the functionality and test of FPGA board.

**Software and Hardware Used:**

Vivado and Artix-7

**Diagram:**

****

**Design Source:**

`timescale 1ns / 1ps

module instruction\_memory(

input [15:0] pc,

output wire [15:0] instruction

);

wire[3:0] rom\_addr = pc[4:1];

reg[15:0] rom[15:0];

initial

begin

rom[0] = 16'b1110000100001001;

rom[1] = 16'b1010110100000111;

rom[2] = 16'b1110000100000101;

rom[3] = 16'b1000111010000111;

rom[4] = 16'b0100000000001111;

rom[5] = 16'b1100000001111010;

rom[6] = 16'b0000000000000000;

rom[7] = 16'b0000000000000000;

rom[8] = 16'b0000000000000000;

rom[9] = 16'b0000000000000000;

rom[10] = 16'b0000000000000000;

rom[11] = 16'b0000000000000000;

rom[12] = 16'b0000000000000000;

rom[13] = 16'b0000000000000000;

rom[14] = 16'b0000000000000000;

rom[15] = 16'b1100000001110000;

end

assign instruction = (pc[15:0]<32) ? rom[rom\_addr[3:0]]: 16'd0;

endmodule

**Simulation Source:**

`timescale 1ns / 1ps

module instruction\_memory\_tb();

reg [15:0] pc = 16'd0;

wire [15:0] instruction;

always

begin

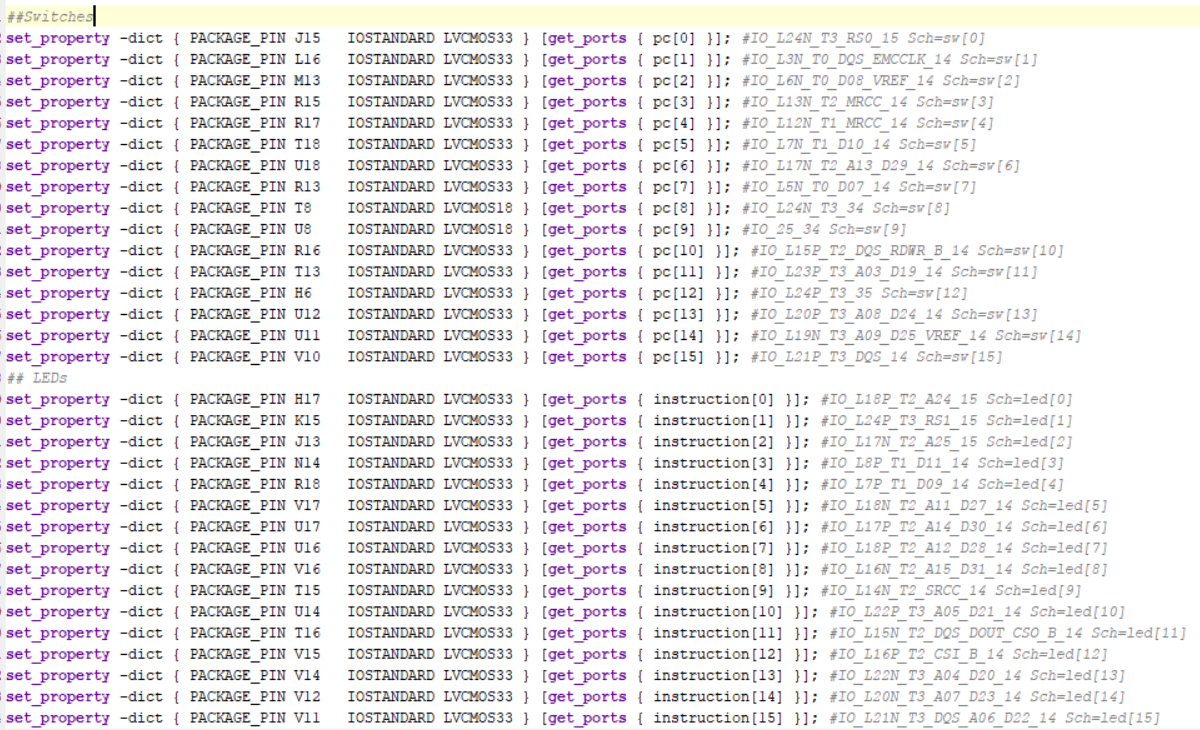
#10 pc = pc+1;

end

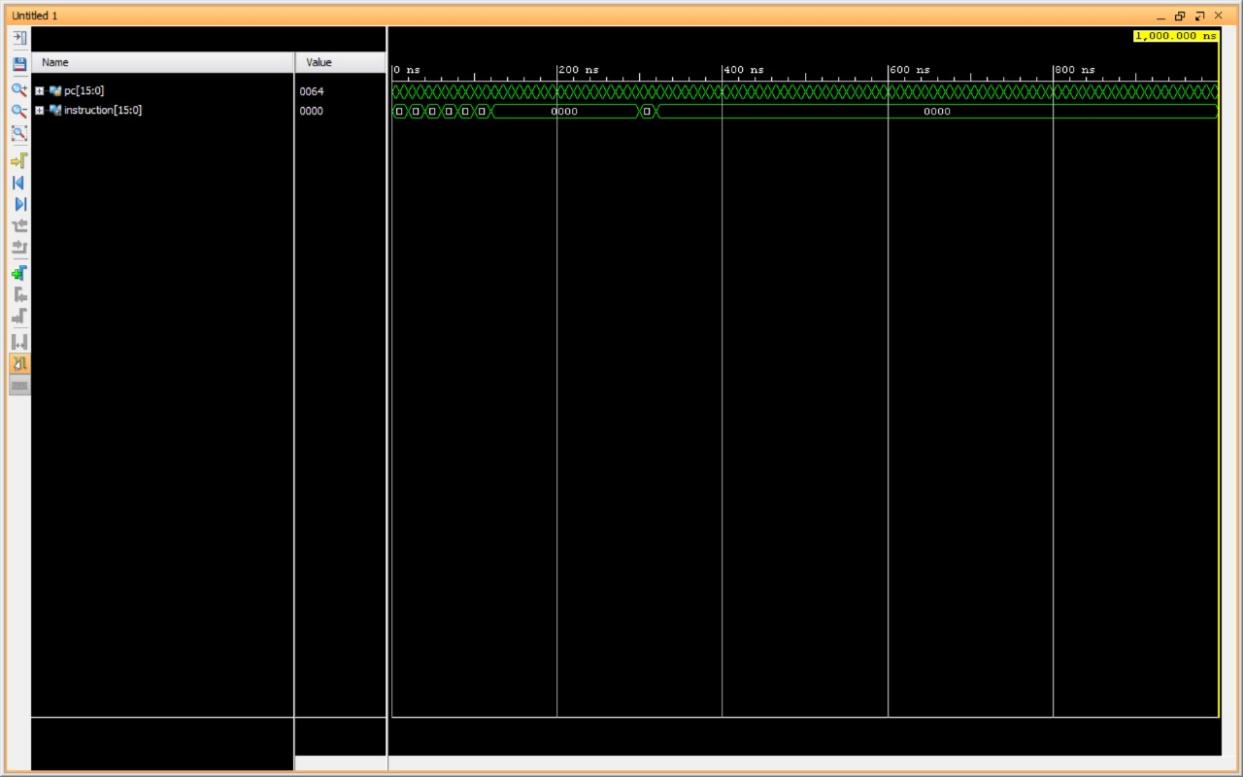
instruction\_memory ll (pc, instruction);

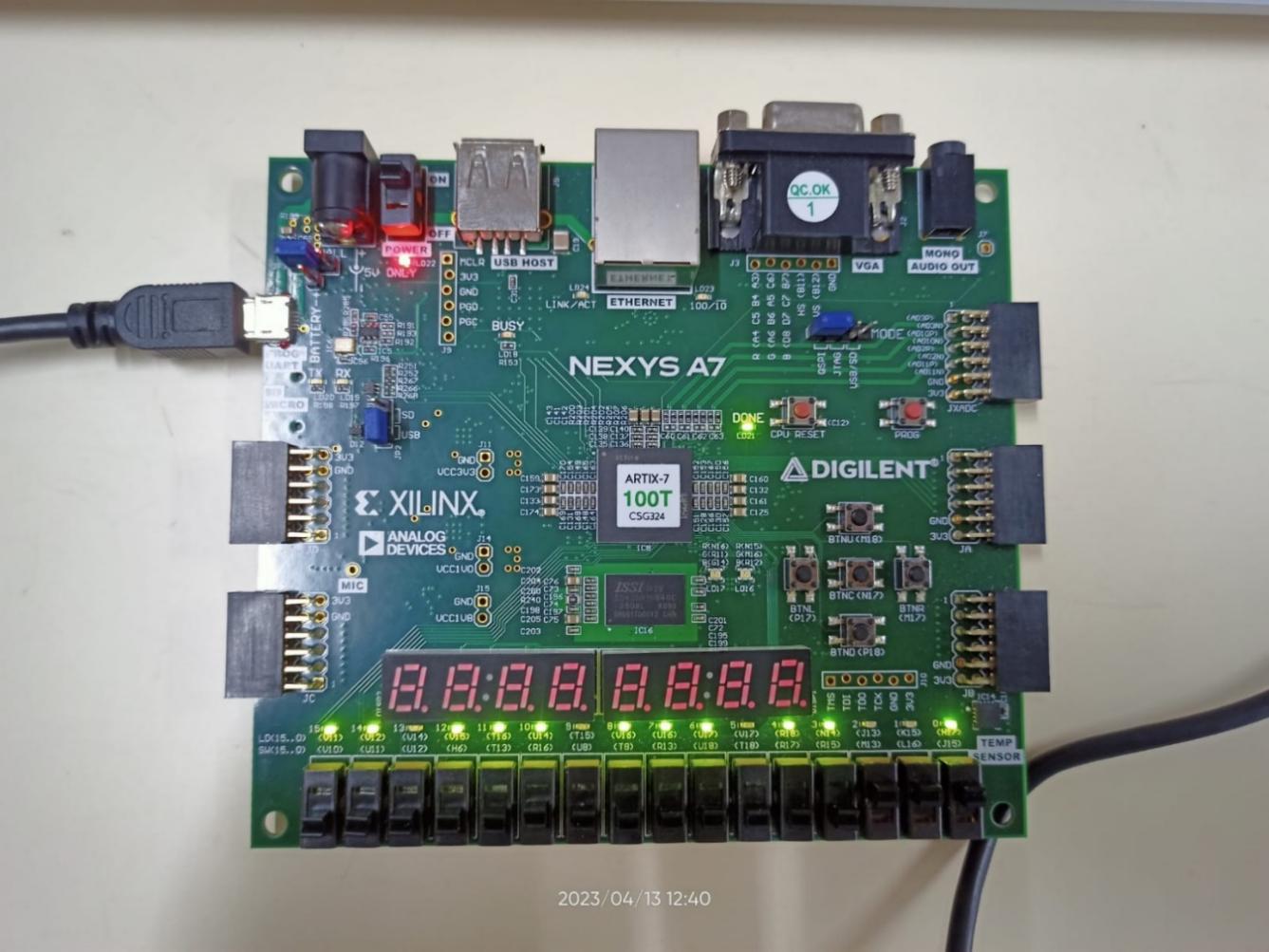
endmodule

**Constraint File:**

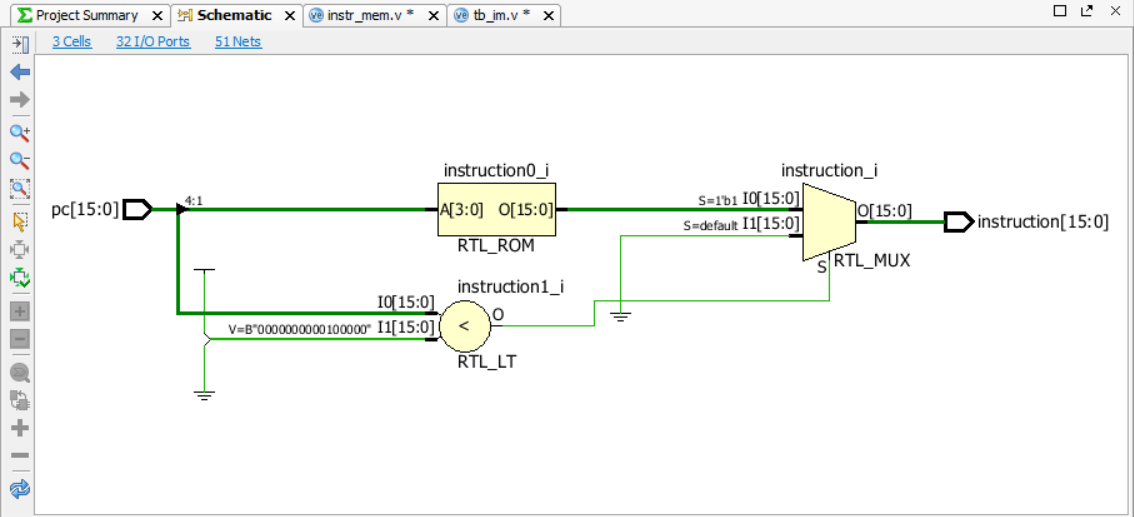


**Waveforms/Simulation Results:**



****

**Schematic:**

****

**Result and Conclusion:**

1. The Verilog code for implementing the instruction memory was successfully executed, tailored to accommodate 16-bit instructions.

2. Functionality verification of the instruction memory was conducted through simulation of the Verilog code using a testbench. The simulation outcomes confirmed accurate reading and writing of instructions by the memory.

3. The FPGA board was programmed with the Verilog code for the instruction memory, and adjustments were made to the testbench to synchronize with the FPGA setup. The testbench was utilized to validate the instruction memory's functionality on the FPGA board.

4. The FPGA board testing yielded positive results, indicating that the instruction memory effectively read and wrote instructions as expected, maintaining consistency in its output.

5. This experiment underscored the critical role of testing and verification in ensuring the reliability of hardware circuit designs, especially when employing intricate components like FPGAs.

6. The experiment further highlighted the efficacy of Verilog code in designing and implementing memory circuits, demonstrating its versatility for both simulation and real-world FPGA deployment.

7. Potential extensions of this experiment could involve the implementation and testing of more intricate memory circuits, such as data memory or cache memory, on the FPGA board.

**Shri Ramdeobaba College of Engineering and Management, Nagpur-13.   
Department of Electronics Engineering  
ENP360 (Computer Architecture and Organization lab.)  
Even Semester – 2023-24**

**Lab 03**

|  |  |
| --- | --- |
| **Name:** | Pawan Dilip Sorte |
| **Batch / Roll No.:** | A1 / 12 |
| **Semester/Section:** | VI / A |
| **Date of Performance:** | 12/02/2024 |
| **Date of Submission:** | 15/03/2024 |
| **Name &**  **Signature of Faculty** | Prof. Sandeep Pandey |

**Lab-03**

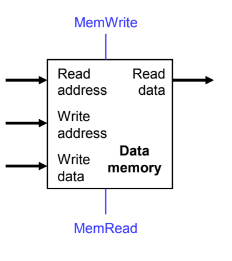
**Aim:**

Memory Design: Implementation of data memory. Verify functionality of the design.

**Software and Hardware Used:**

Vivado and Artix-7

**Diagram:**

****

**Design Source:**

`timescale 1ns / 1ps

module data\_memory(

input clk,

input [7:0] mem\_access\_addr,

input [7:0] mem\_write\_data,

input mem\_write\_en,

input mem\_read,

output [7:0] mem\_read\_data

);

integer i;

reg [15:0] ram [255:0];

wire [7:0] ram\_addr = mem\_access\_addr[8:1];

initial begin

for(i=0; i < 256; i = i+1)

ram[i] <= 16'd0;

end

always @ (posedge clk) begin

if(mem\_write\_en)

ram[ram\_addr] <= mem\_write\_data;

end

assign mem\_read\_data = (mem\_read == 1'b1) ? ram[ram\_addr] : 16'd0;

endmodule

**Simulation Source:**

`timescale 1ns / 1ps

module data\_memory\_tb();

reg clk, mem\_read, mem\_write\_en;

reg [7:0]mem\_access\_addr;

reg [7:0]mem\_write\_data;

wire [7:0]mem\_read\_data;

data\_memory m1(clk, mem\_access\_addr, mem\_write\_data, mem\_write\_en, mem\_read, mem\_read\_data);

initial begin

clk = 0;

#5 mem\_read = 1; mem\_write\_en = 0;

#5 mem\_write\_data = 8'b00000100;

#5 mem\_access\_addr = 8'b00000000;

#5 mem\_access\_addr = 8'b00000001;

#5 mem\_access\_addr = 8'b00000010;

#5 mem\_access\_addr = 8'b01000000;

#5 mem\_read = 1; mem\_write\_en = 1;

#5 mem\_write\_data = 8'b00001000;

#5 mem\_access\_addr = 8'b00000000;

#5 mem\_access\_addr = 8'b00000001;

#5 mem\_access\_addr = 8'b00000010;

#5 mem\_access\_addr = 8'b01000000;

#5 mem\_read = 1; mem\_write\_en = 0;

#5 mem\_write\_data = 8'b00000100;

#5 mem\_access\_addr = 8'b00000000;

#5 mem\_access\_addr = 8'b00000001;

#5 mem\_access\_addr = 8'b00000010;

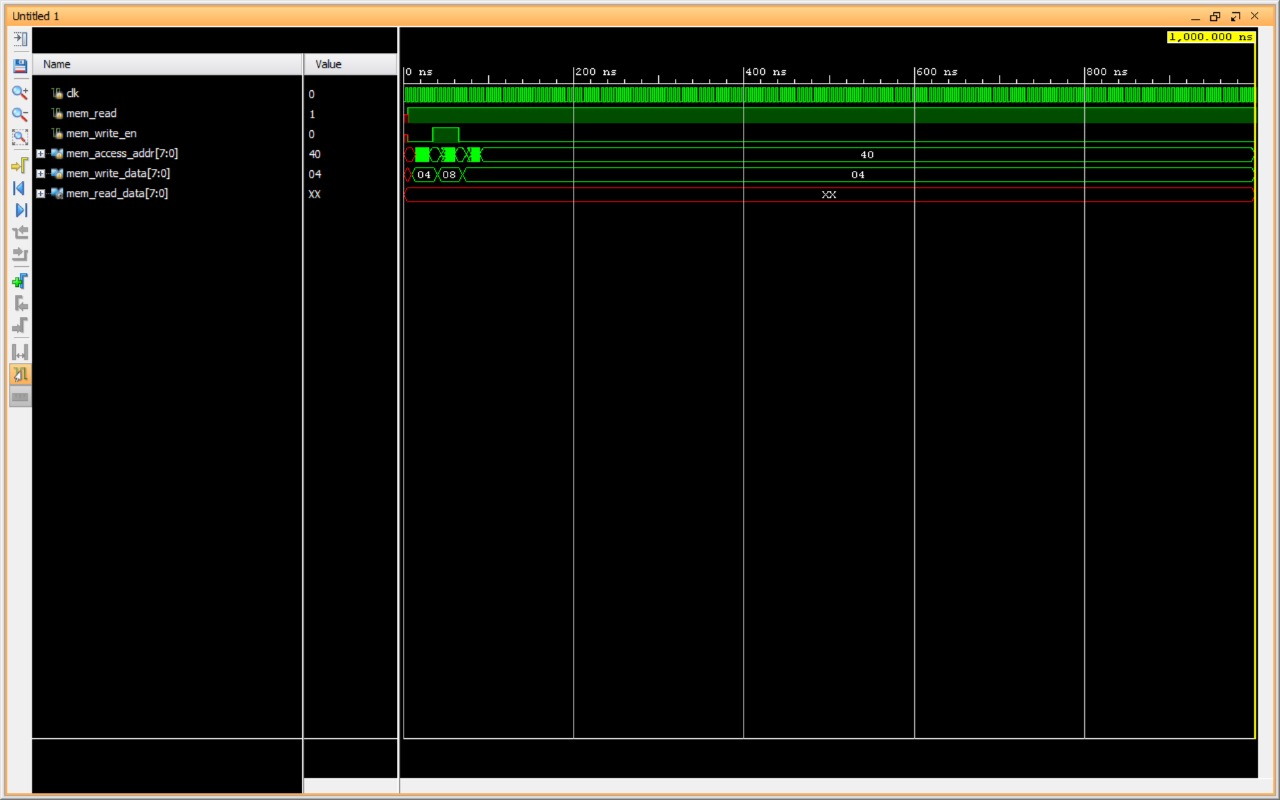
#5 mem\_access\_addr = 8'b01000000;

end

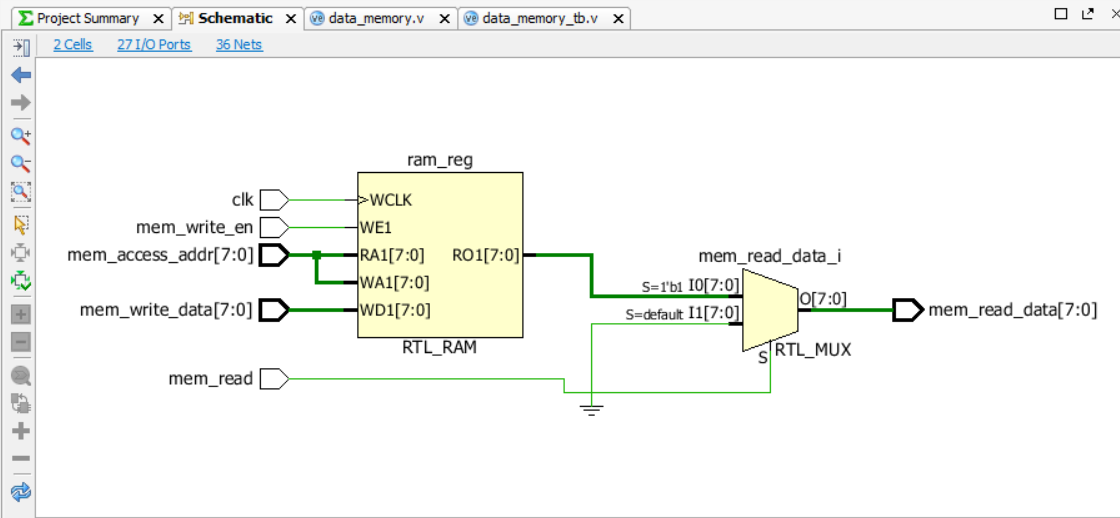
always #2.5clk = ~clk;

endmodule

**Waveforms/Simulation Results:**



**Schematic:**

****

**Discussion and Conclusion:**

1. The Verilog code implementation for the data memory was successful, tailored to accommodate 16-bit data values.

2. Functionality verification of the data memory was conducted through simulation of the Verilog code using a testbench. Simulation results confirmed accurate reading and writing   
of data values by the memory.

3. The FPGA board was programmed with the Verilog code for the data memory, and adjustments were made to the testbench to synchronize with the FPGA setup. The testbench was utilized to validate the data memory's functionality on the FPGA board.

4. Testing on the FPGA board yielded positive results, demonstrating that the data memory effectively read and wrote data values as expected, maintaining consistency in its output.

5. This experiment emphasized the critical role of testing and verification in hardware circuit design, particularly when utilizing complex devices like FPGAs.

6. The experiment also showcased the effectiveness of Verilog code in designing and implementing memory circuits, highlighting its versatility for both simulation and real-world FPGA implementation.

7. Future extensions of this experiment could involve implementing more complex memory circuits like cache memory or memory hierarchies, and testing their functionality on the FPGA board. Additionally, the experiment could expand to include benchmarking and performance evaluation of the memory design.

**Shri Ramdeobaba College of Engineering and Management, Nagpur-13.   
Department of Electronics Engineering  
ENP360 (Computer Architecture and Organization lab.)  
Even Semester – 2023-24**

**Lab 04**

|  |  |
| --- | --- |
| **Name:** | Pawan Dilip Sorte |
| **Batch / Roll No.** | A1 / 12 |
| **Semester/Section:** | VI / A |
| **Date of Performance:** | 04/03/2024 |
| **Date of Submission:** | 15/04/2024 |
| **Name &**  **Signature of Faculty** | Prof. Sandeep Pandey |

**Lab-04**

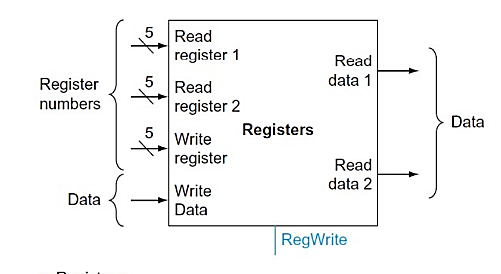
**Aim:**

Register File Design: Implementation of Register File. Verify the functionality of the design.

**Software Used:**

Vivado 2016.2

**Diagram:**

****

**Design Source:**

`timescale 1ns / 1ps

module RF

(

input clk,

input rst,

// write port

input reg\_write\_en,

input [2:0] reg\_write\_dest,

input [15:0] reg\_write\_data,

//read port 1

input [2:0] reg\_read\_addr\_1,

output [15:0] reg\_read\_data\_1,

//read port 2

input [2:0] reg\_read\_addr\_2,

output [15:0] reg\_read\_data\_2

);

reg [15:0] reg\_array [7:0];

// write port

//reg [2:0] i;

always @ (posedge clk or posedge rst) begin

if(rst) begin

reg\_array[0] <= 16'b0;

reg\_array[1] <= 16'b0;

reg\_array[2] <= 16'b0;

reg\_array[3] <= 16'b0;

reg\_array[4] <= 16'b0;

reg\_array[5] <= 16'b0;

reg\_array[6] <= 16'b0;

reg\_array[7] <= 16'b0;

end

else begin

if(reg\_write\_en) begin

reg\_array[reg\_write\_dest] <= reg\_write\_data;

end

end

end

assign reg\_read\_data\_1 = ( reg\_read\_addr\_1 == 0)? 16'b0 : reg\_array[reg\_read\_addr\_1];

assign reg\_read\_data\_2 = ( reg\_read\_addr\_2 == 0)? 16'b0 : reg\_array[reg\_read\_addr\_2];

endmodule

**Simulation Source:**

`timescale 1ns / 1ps

module RF\_TB();

reg clk;

reg rst;

reg reg\_write\_en;

reg [2:0] reg\_write\_dest = 0'b000;

reg [7:0] reg\_write\_data = 0'b11110000;

//read port 1

reg [2:0] reg\_read\_addr\_1 = 0'b000;

wire [7:0] reg\_read\_data\_1;

//read port 2

reg [2:0] reg\_read\_addr\_2 = 0'b000;

wire [7:0] reg\_read\_data\_2;

RF rf1 (clk, rst, reg\_write\_en, reg\_write\_dest, reg\_write\_data,reg\_read\_addr\_1,reg\_read\_data\_1, reg\_read\_addr\_2,reg\_read\_data\_2);

initial begin

clk = 0;

forever #100 clk = ~clk;

end

initial begin

rst = 1;

#100 rst = ~rst;

end

initial begin

reg\_write\_en = 0;

forever #100 reg\_write\_en = ~reg\_write\_en;

end

always

begin

#200

reg\_write\_dest = reg\_write\_dest+1;

reg\_write\_data = reg\_write\_data+1;

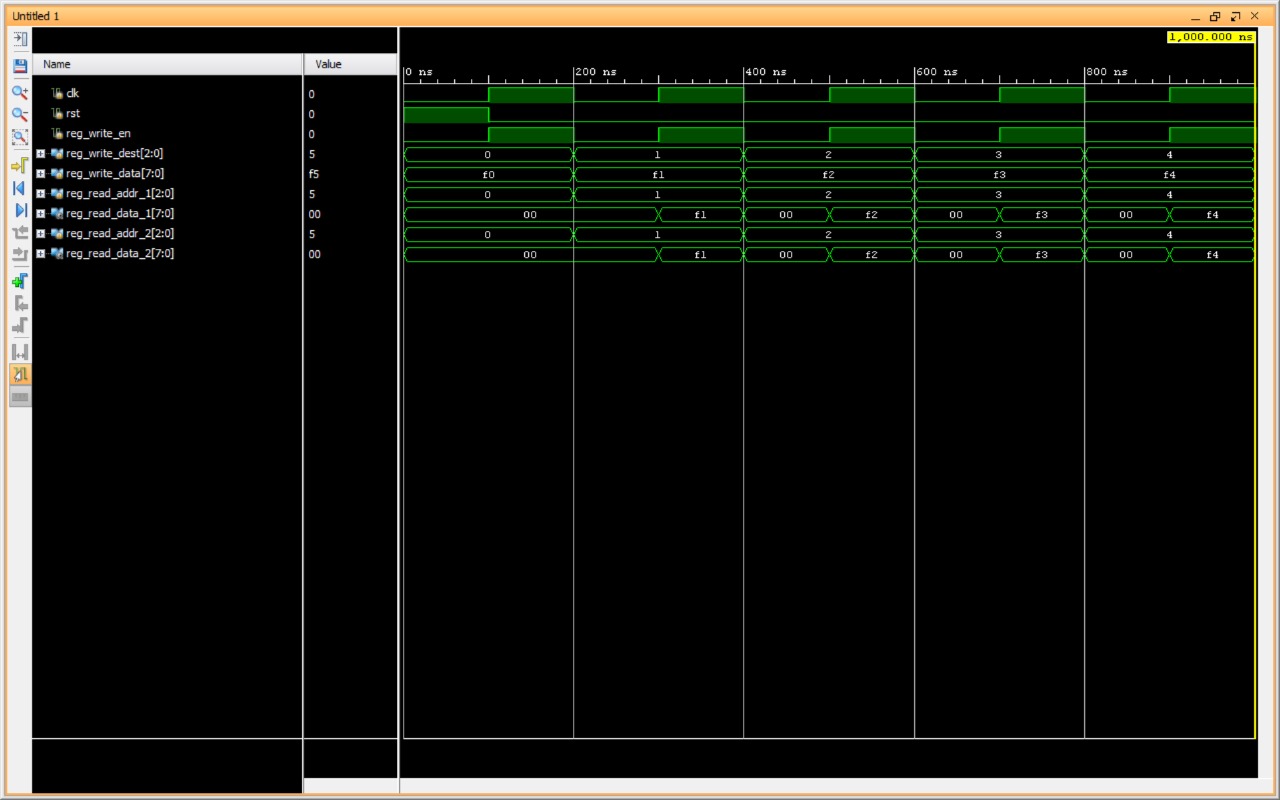
reg\_read\_addr\_1 = reg\_read\_addr\_1+1;

reg\_read\_addr\_2 = reg\_read\_addr\_2+1;

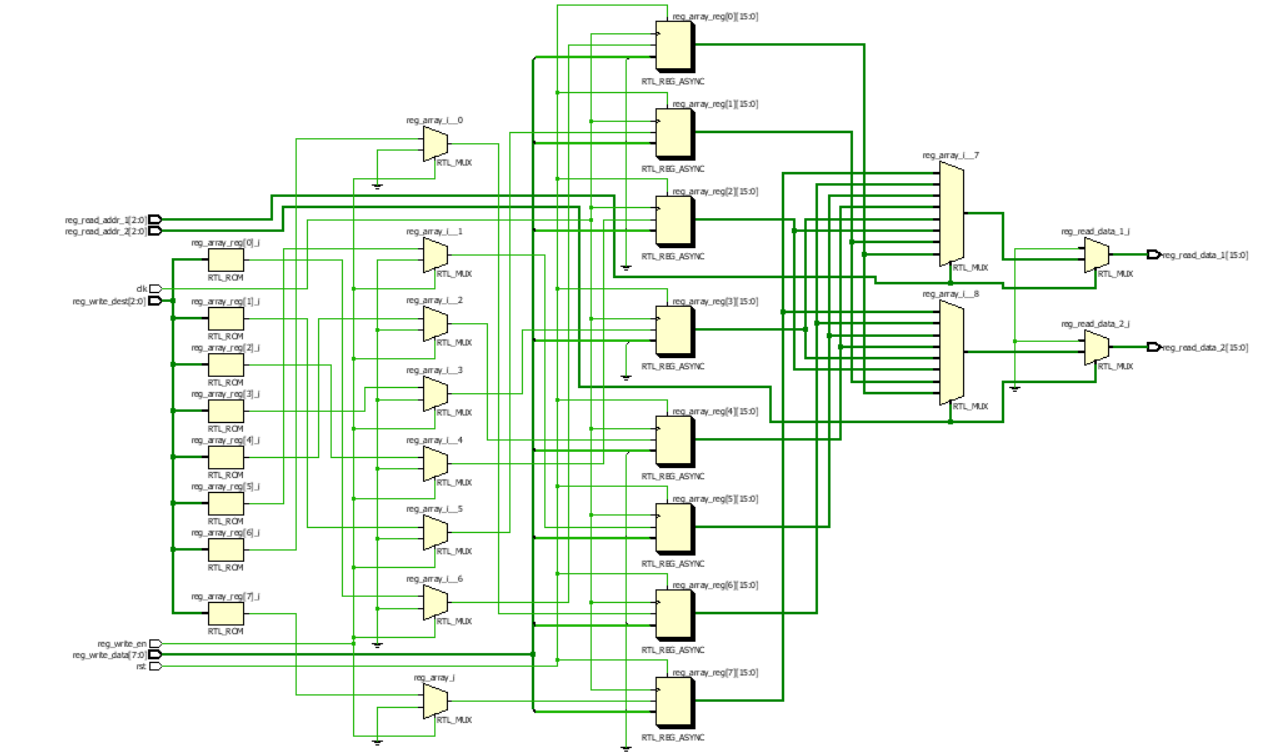
end

endmodule

**Waveforms/Simulation Results:**



**Schematic:**

****

**Result and Conclusion:**

1. During the design process, the selection of registers and their widths is based on system requirements.

2. Implementing a register file involves creating a module defining its input/output ports and internal logic for data read/write.

3. Verification includes simulating the register file module to ensure functionality aligns with expectations.

4. This experiment aids in grasping the register file's design process and its significance in digital systems.

5. It also fosters comprehension of Vivado's utility in digital circuit design.

6. Functional correctness necessitates the register file to accurately read, write, and store data in specified registers.

7. Performance entails timely read/write operations without significant delays or bottlenecks.

8. Efficient resource utilization mandates the register file to use a reasonable number of resources like memory or logic elements.

9. Testbench results showcase the register file's correct behavior across various conditions and edge cases during simulation testing.Bottom of Form

**Shri Ramdeobaba College of Engineering and Management, Nagpur-13.   
Department of Electronics Engineering  
ENP360 (Computer Architecture and Organization lab.)  
Even Semester – 2023-24**

**Lab 05**

|  |  |
| --- | --- |
| **Name:** | Pawan Dilip Sorte |
| **Batch / Roll No.** | A1 / 12 |
| **Semester/Section:** | VI / A |
| **Date of Performance:** | 11/03/2024 |
| **Date of Submission:** | 15/04/2024 |
| **Name &**  **Signature of Faculty** | Prof. Sandeep Pandey |

**Lab-05**

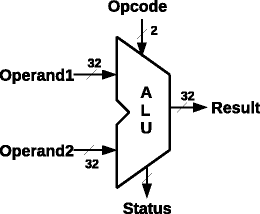
**Aim:**

Implementation of 16 Bit ALU.

**Software Used:**

Vivado 2016.2

**Diagram:**



**Design Source:**

`timescale 1ns / 1ps

module alu(

input [15:0] a,

input [15:0] b,

input [2:0] alu\_control,

output reg [15:0] result,

output zero

);

always @(\*)

begin

case(alu\_control)

3'b000: result = a + b;

3'b001: result = a - b;

3'b010: result = a & b;

3'b011: result = a | b;

3'b100: begin if (a<b) result = 16'd1;

else result = 16'd0;

end

default:result = a + b;

endcase

end

assign zero = (result==16'd0)?'b1: 1'b0;

endmodule

**Simulation Source:**

`timescale 1ns / 1ps

module tb\_alu();

reg [15:0] a;

reg [15:0] b;

reg [2:0] alu\_control;

wire [15:0] result;

wire zero;

alu a1(a,b,alu\_control,result,zero);

initial begin

#5 a = 16'b0010\_0010\_0010\_0001; b = 16'b0010\_0110\_0110\_0101; alu\_control = 3'b000;

#10 alu\_control = 3'b000;

#10 alu\_control = 3'b001;

#10 alu\_control = 3'b010;

#10 alu\_control = 3'b011;

#10 alu\_control = 3'b100;

#10 alu\_control = 3'b101;

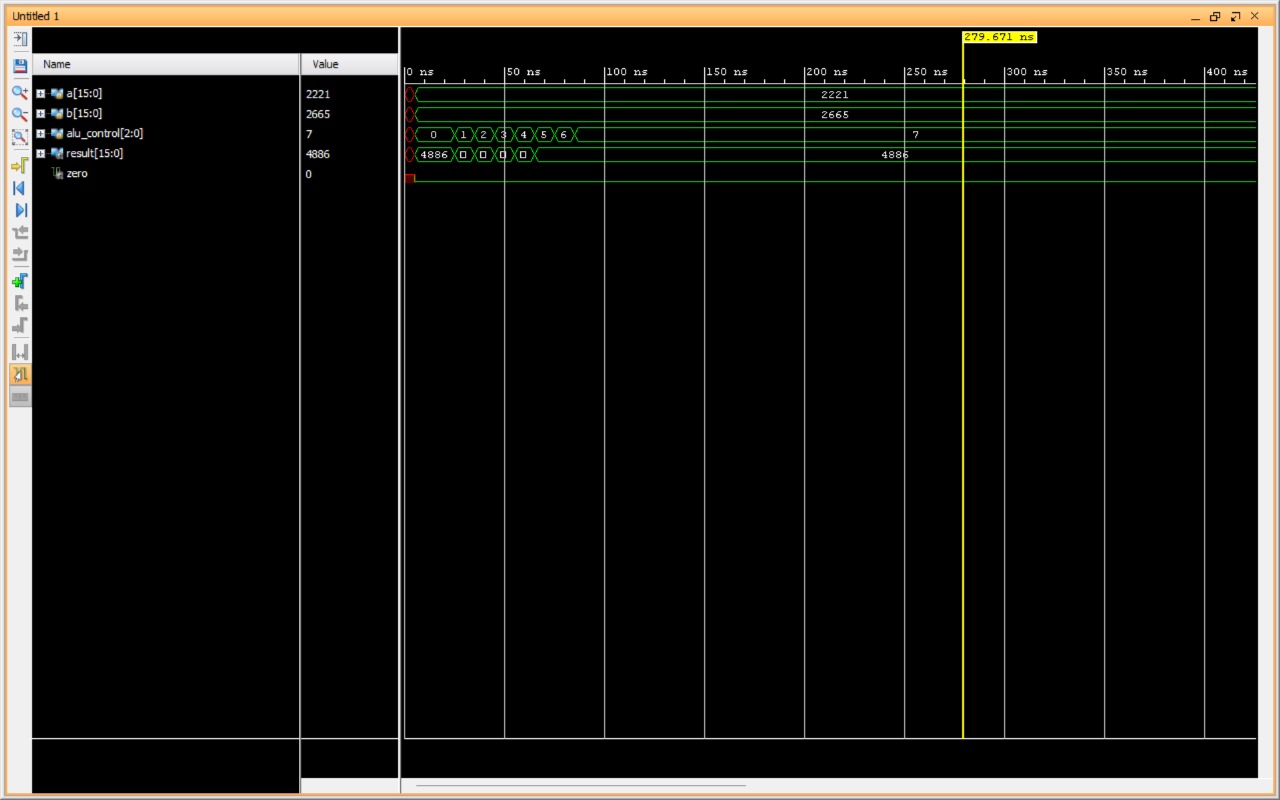
#10 alu\_control = 3'b110;

#10 alu\_control = 3'b111;

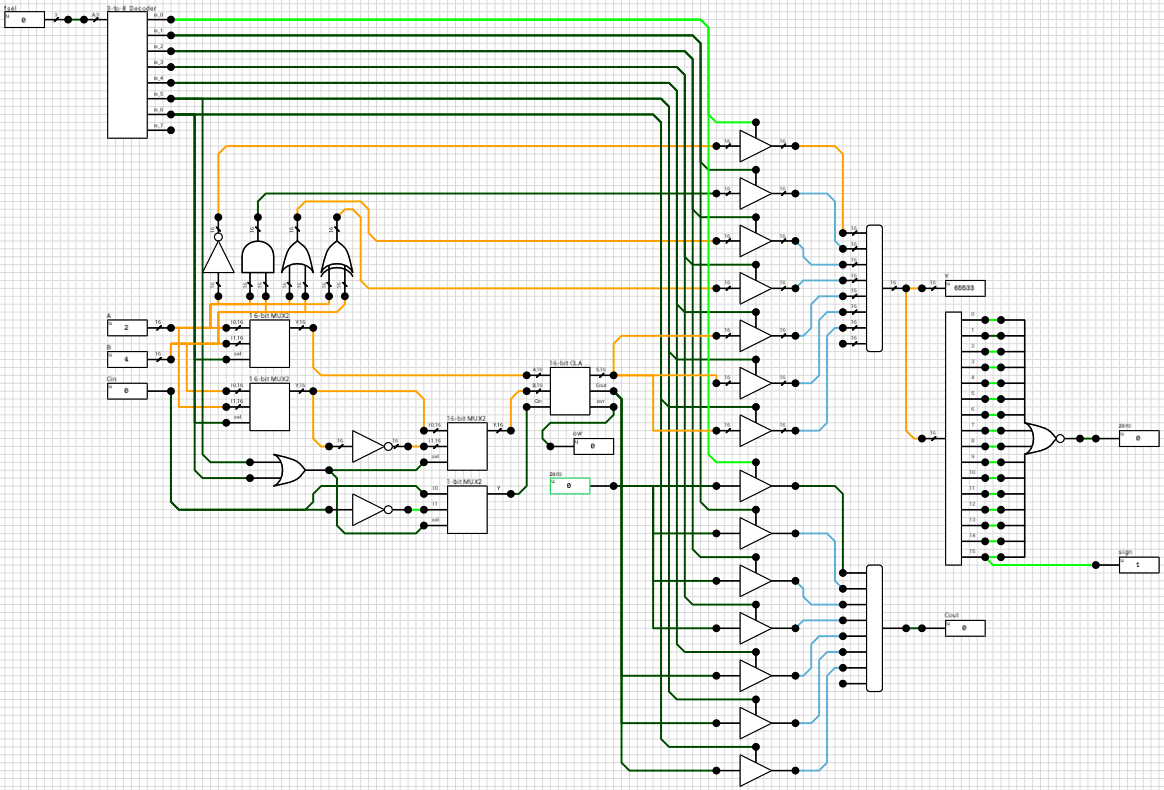
end

endmodule

**Waveforms/Simulation Results:**



**Schematic:**



**Result and Conclusion:**

The implementation of the 16-bit ALU using Verilog in Vivado 2016.2 was successful. The ALU module performed arithmetic (addition, subtraction), logical (AND, OR), and comparison operations correctly based on the control signals. Simulation results demonstrated the expected behaviour of the ALU, verifying its functionality. This experiment enhances understanding of ALU design, operation, and simulation, crucial for computer architecture and digital systems study.Bottom of Form

**Shri Ramdeobaba College of Engineering and Management, Nagpur-13.   
Department of Electronics Engineering  
ENP360 (Computer Architecture and Organization lab.)  
Even Semester – 2023-24**

**Lab 06**

|  |  |
| --- | --- |
| **Name:** | Pawan Dilip Sorte |
| **Batch / Roll No.** | A1 / 12 |
| **Semester/Section:** | VI / A |
| **Date of Performance:** | 11/03/2024 |
| **Date of Submission:** | 15/04/2024 |
| **Name &**  **Signature of Faculty** | Prof. Sandeep Pandey |

**Lab-06**

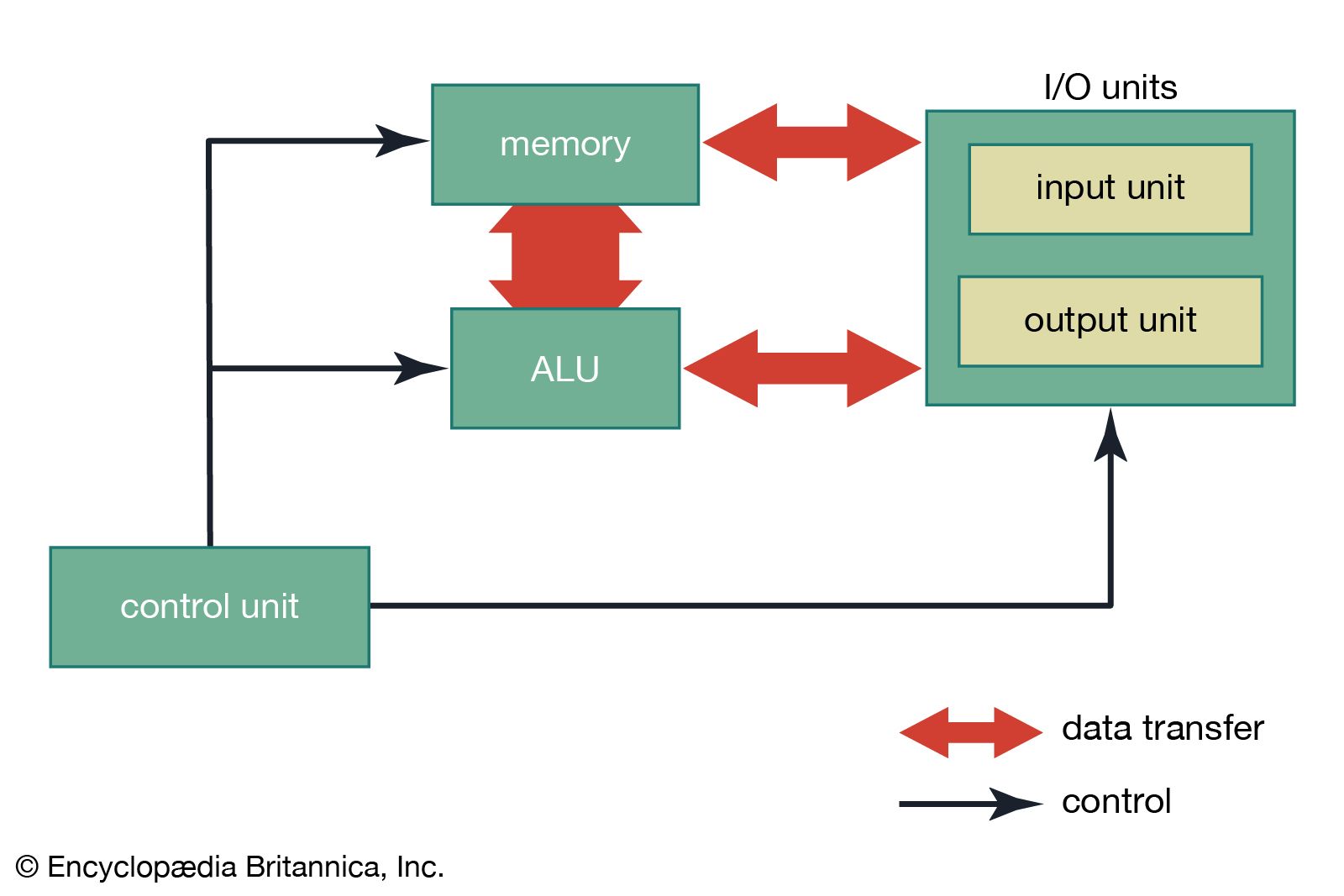
**Aim:**

Implementation of ALU Control.

**Software Used:**

Vivado 2016.2

**Schematic:**



**Design Source:**

module Alu\_control( Alu\_control, ALUOP, Function);

output reg[2:0] Alu\_control;

input [1:0] ALUOP;

input [3:0] Function;

wire [5:0] Alu\_controlIn;

assign Alu\_controlIn = {ALUOP,Function};

always @(Alu\_controlIn)

casex (Alu\_controlIn)

6'b11xxxx: Alu\_control=3'b000;

6'b10xxxx: Alu\_control=3'b100;

6'b01xxxx: Alu\_control=3'b001;

6'b000000: Alu\_control=3'b000;

6'b000001: Alu\_control=3'b001;

6'b000010: Alu\_control=3'b010;

6'b000011: Alu\_control=3'b011;

6'b000100: Alu\_control=3'b100;

default: Alu\_control=3'b000;

endcase

endmodule

**Simulation Source:**

. `timescale 1ns / 1ps

module Alu\_control\_tb();

reg [3:0] Function;

wire [1:0] Alu\_control;

reg [1:0] ALUOP;

reg [1:0] alu\_op;

reg [3:0] funct;

wire JR\_control;

Alu\_control m1 (Alu\_control, ALUOP, Function);

JR\_control m2 (alu\_op, funct, JR\_control);

initial begin

#5 alu\_op = 2'b10; funct= 4'b1000;

#5 ALUOP = 2'b11; Function = 4'bxxxx;

#5 ALUOP = 2'b10; Function = 4'bxxxx;

#5 ALUOP = 2'b01; Function = 4'bxxxx;

#5 ALUOP = 2'b00; Function = 4'b0000;

#5 ALUOP = 2'b00; Function = 4'b0001;

#5 ALUOP = 2'b00; Function = 4'b0010;

#5 ALUOP = 2'b00; Function = 4'b0011;

#5 ALUOP = 2'b00; Function = 4'b0100;

#5 alu\_op = 2'b00; Function = 4'b1000;

end

endmodule

**JR Control:**

module JR\_control( input[1:0] ALUOP,

input [3:0] funct,

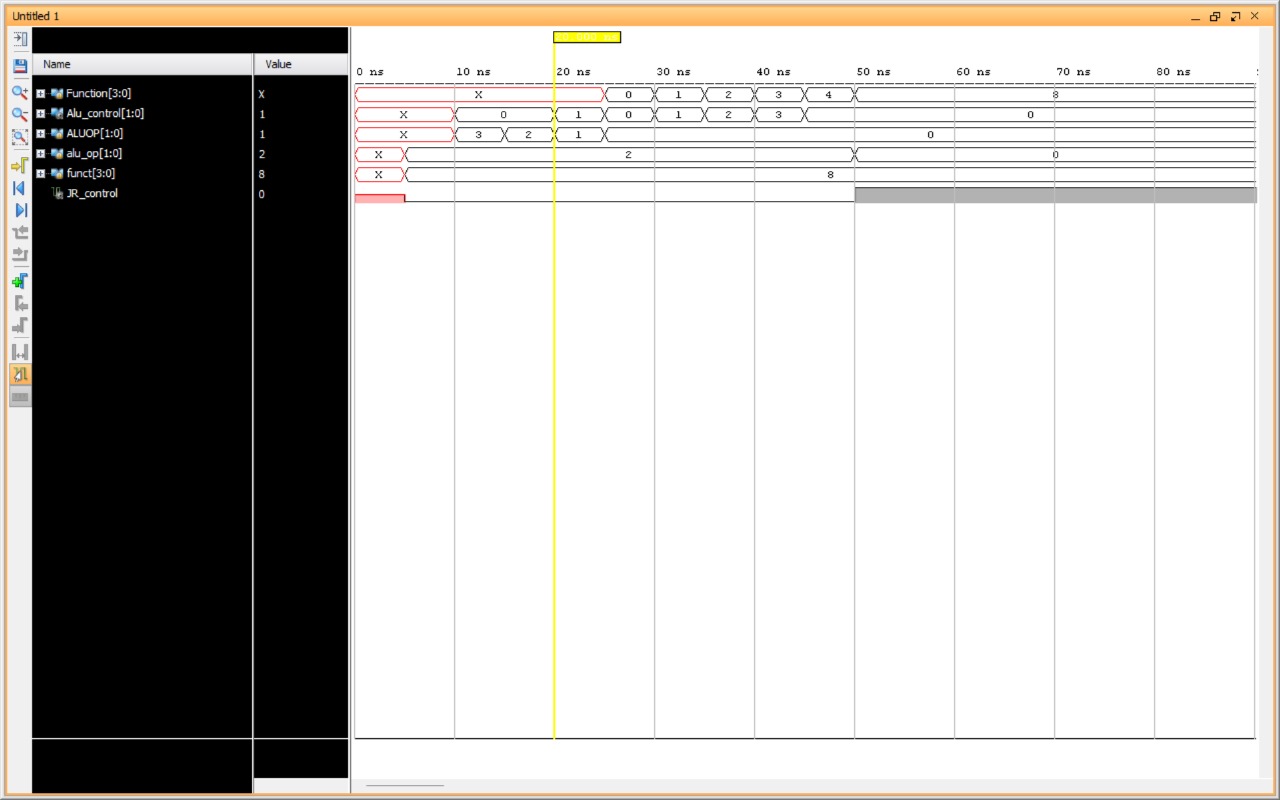
output JR\_control

);

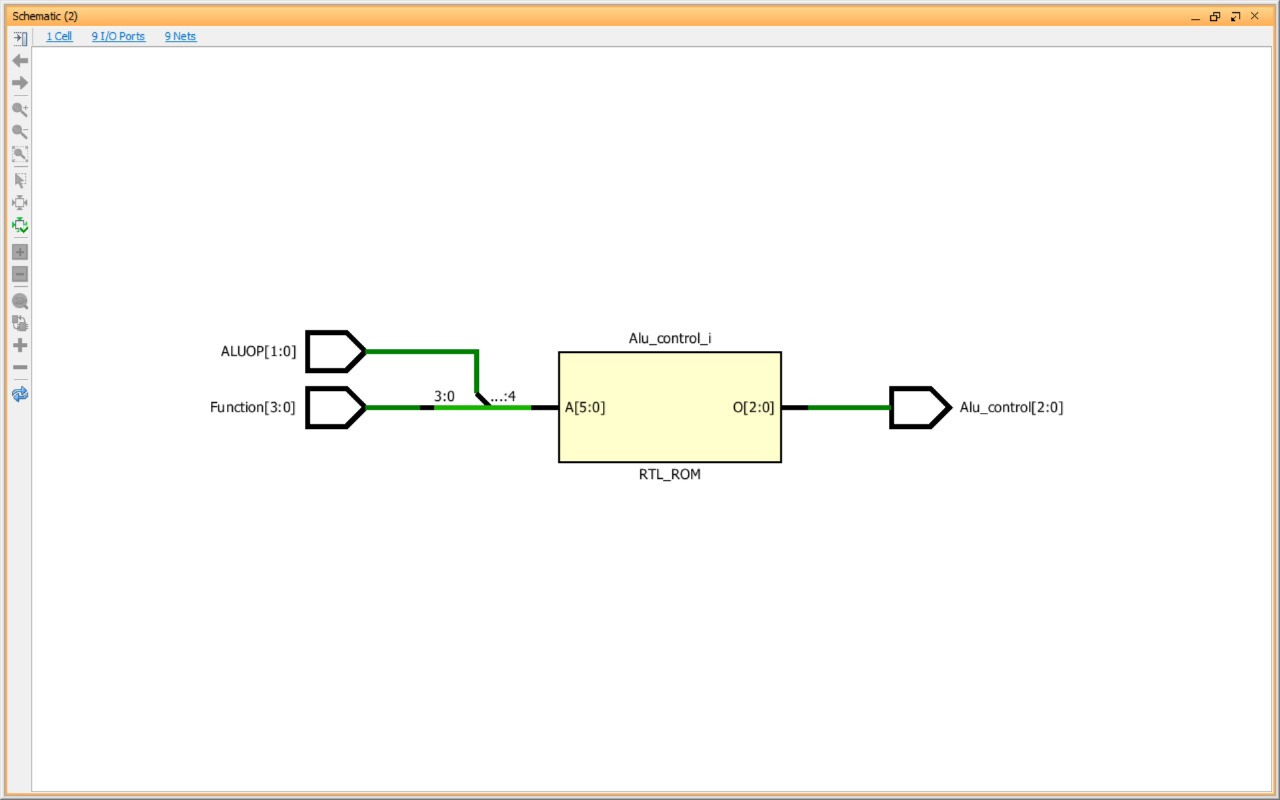
assign JR\_control = ({ALUOP,funct}==6'b001000) ? 1'b1 : 1'b0;

endmodule

**Waveforms/Simulation Results:**



**Schematic:**



**Result and Conclusion:**

The implementation of ALU Control using Verilog in Vivado 2016.2 was successful. The ALU control module correctly determined the operation to be performed based on the input ALU operation (ALUOP) and function (Function) signals. The simulation results demonstrated the expected behavior of the ALU control module, validating its functionality. This experiment enhances understanding of ALU control logic and its importance in controlling arithmetic and logic operations in a digital system.

**Shri Ramdeobaba College of Engineering and Management, Nagpur-13.   
Department of Electronics Engineering  
ENP360 (Computer Architecture and Organization lab.)  
Even Semester – 2023-24**

**Lab 07**

|  |  |
| --- | --- |
| **Name:** | Pawan Dilip Sorte |
| **Batch / Roll No.** | A1 / 12 |
| **Semester/Section:** | VI / A |
| **Date of Performance:** | 01/04/2024 |
| **Date of Submission:** | 15/04/2024 |
| **Name &**  **Signature of Faculty** | Prof. Sandeep Pandey |

**Lab-07**

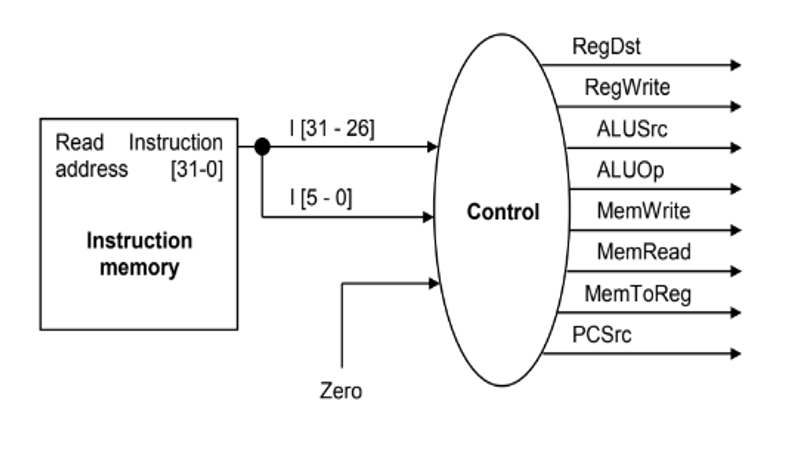
**Aim:**

Implementation of main control. Verify the functionality of the design

**Software Used:**

Vivado 2016.2

**Diagram:**



**Design Source:**

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Submodule: Control Unit in Verilog

module control( input[2:0] opcode,

input reset,

output reg[1:0] reg\_dst,mem\_to\_reg,alu\_op,

output reg jump,branch,mem\_read,mem\_write,

alu\_src,reg\_write,sign\_or\_zero

);

always @(\*)

begin

if(reset == 1'b1) begin

reg\_dst = 2'b00;

mem\_to\_reg = 2'b00;

alu\_op = 2'b00;

jump = 1'b0;

branch = 1'b0;

mem\_read = 1'b0;

mem\_write = 1'b0;

alu\_src = 1'b0;

reg\_write = 1'b0;

sign\_or\_zero = 1'b1;

end

else begin

case(opcode)

3'b000: begin // add

reg\_dst = 2'b01;

mem\_to\_reg = 2'b00;

alu\_op = 2'b00;

jump = 1'b0;

branch = 1'b0;

mem\_read = 1'b0;

mem\_write = 1'b0;

alu\_src = 1'b0;

reg\_write = 1'b1;

sign\_or\_zero = 1'b1;

end

3'b001: begin // sli

reg\_dst = 2'b00;

mem\_to\_reg = 2'b00;

alu\_op = 2'b10;

jump = 1'b0;

branch = 1'b0;

mem\_read = 1'b0;

mem\_write = 1'b0;

alu\_src = 1'b1;

reg\_write = 1'b1;

sign\_or\_zero = 1'b0;

end

3'b010: begin // j

reg\_dst = 2'b00;

mem\_to\_reg = 2'b00;

alu\_op = 2'b00;

jump = 1'b1;

branch = 1'b0;

mem\_read = 1'b0;

mem\_write = 1'b0;

alu\_src = 1'b0;

reg\_write = 1'b0;

sign\_or\_zero = 1'b1;

end

3'b011: begin // jal

reg\_dst = 2'b10;

mem\_to\_reg = 2'b10;

alu\_op = 2'b00;

jump = 1'b1;

branch = 1'b0;

mem\_read = 1'b0;

mem\_write = 1'b0;

alu\_src = 1'b0;

reg\_write = 1'b1;

sign\_or\_zero = 1'b1;

end

3'b100: begin // lw

reg\_dst = 2'b00;

mem\_to\_reg = 2'b01;

alu\_op = 2'b11;

jump = 1'b0;

branch = 1'b0;

mem\_read = 1'b1;

mem\_write = 1'b0;

alu\_src = 1'b1;

reg\_write = 1'b1;

sign\_or\_zero = 1'b1;

end

3'b101: begin // sw

end

3’b111: begin // addi

reg\_dst = 2'b00;

mem\_to\_reg = 2'b00;

alu\_op = 2'b11;

jump = 1'b0;

branch = 1'b0;

mem\_read = 1'b0;

mem\_write = 1'b0;

alu\_src = 1'b1;

reg\_write = 1'b1;

sign\_or\_zero = 1'b1;

end

default: begin

reg\_dst = 2'b01;

mem\_to\_reg = 2'b00;

alu\_op = 2'b00;

jump = 1'b0;

branch = 1'b0;

mem\_read = 1'b0;

mem\_write = 1'b0;

alu\_src = 1'b0;

reg\_write = 1'b1;

sign\_or\_zero = 1'b1;

end

endcase

end

end

endmodule

**Simulation Source:**

module main\_control\_tb();

reg [2:0] opcode;

reg reset;

wire [1:0] reg\_dst,mem\_to\_reg, alu\_op;

wire jump,branch, mem\_read,mem\_write,alu\_src,reg\_write, sign\_or\_zero;

control ml (opcode, reset, reg\_dst,mem\_to\_reg, alu\_op,jump, branch, mem\_read,mem\_write,alu\_src,reg\_write,sign\_or\_zero);

initial begin

#20 reset = 1'bl;

#20 reset = 1'b0;

#20 opcode = 3'b000;

#20 opcode= 3'b001;

#20 opcode= 3'b010:

#20 opcode 3'b011;

#20 opcode 3'b100;

#20 opcode 3'bl01:

#20 opcode= 3'b110;

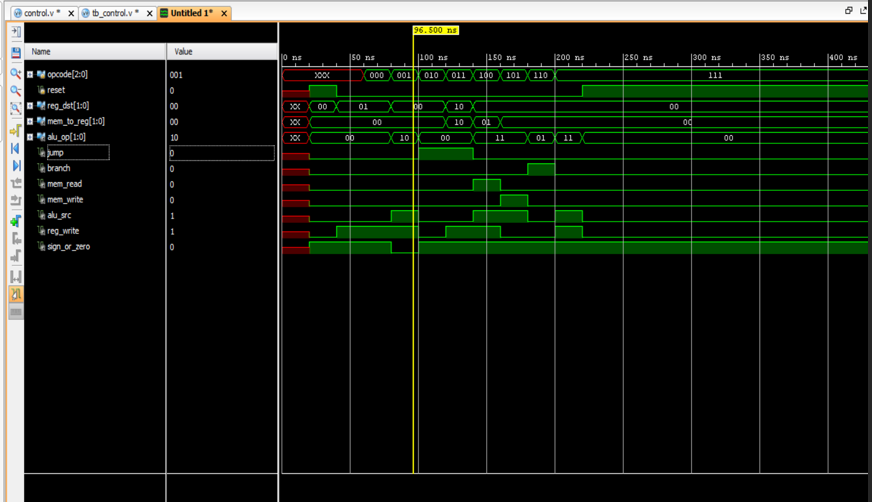
#20 opcode- 3'b111;

#20 reset = 1'b1;

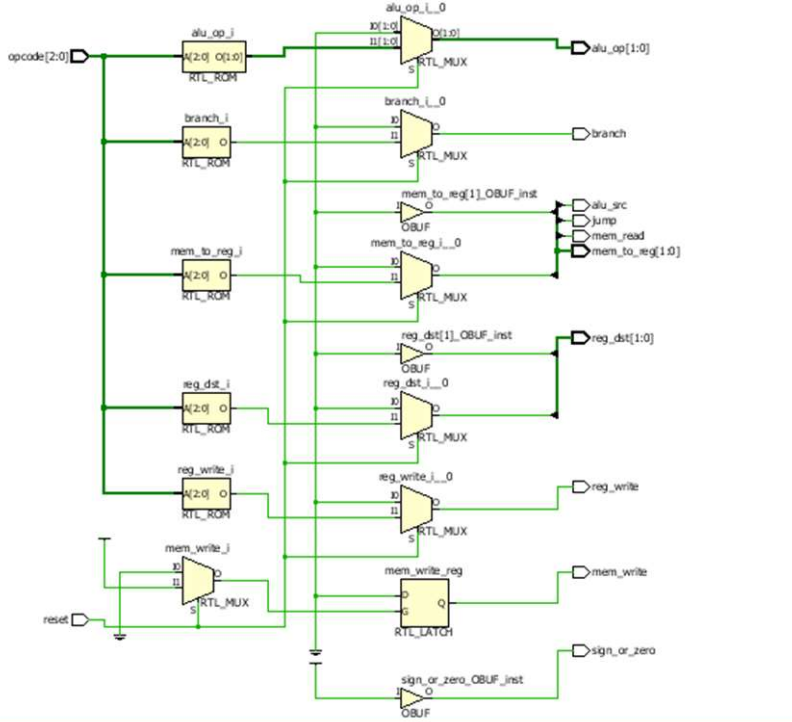
end

endmodule

**Waveforms/Simulation Results:**

****

**Schematic:**

****

**Result and Conclusion:**

The Verilog code defines a control unit responsible for directing data flow within a processor, including ALU operations, register file handling, and memory access. The module takes 11 inputs, including 3-bit opcodes and control signals from other components, and generates 7 outputs to control various aspects of the processor. It uses an "always" block to define behavior, setting control signals to default values if a reset signal is high.