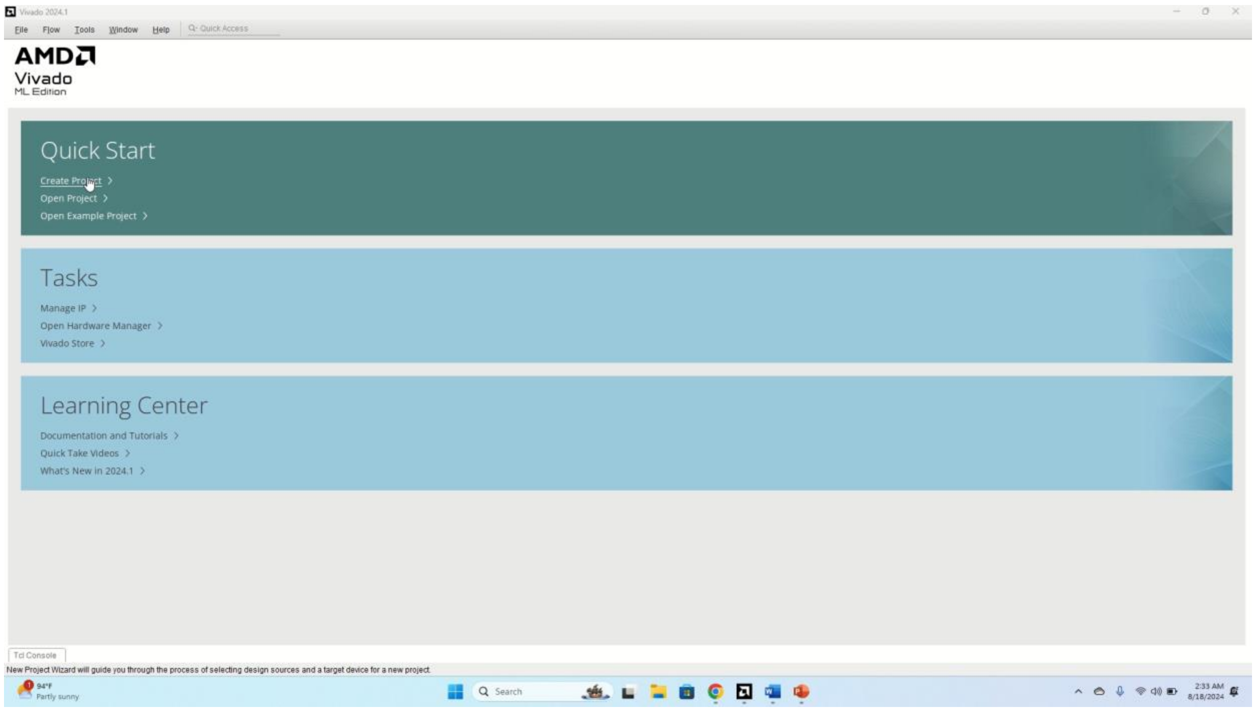
FPGA CODE

ARITMETIC OPERATONS IN VERILOG

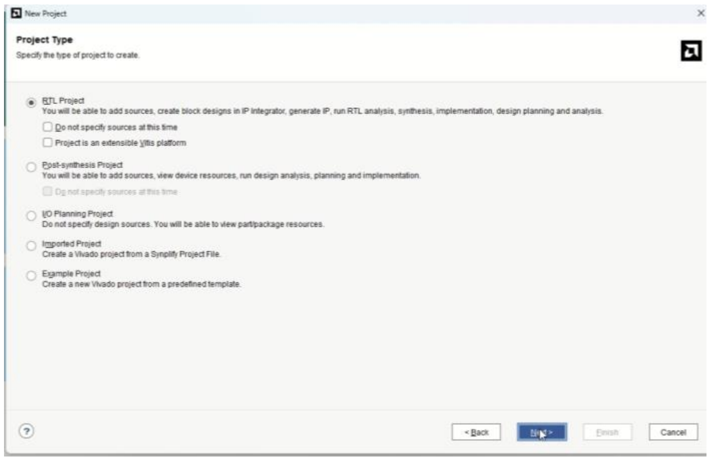
STEPS:

1. Create Project



New Project Wizard will guide you through the process of selecting design sources and a target device for a new project.

2. Click on Next
3. Give a Project name and give the path to store that project where ever you want.
4. Choose ➔RTL Project click on next



Specify the type of project to create.

(●) RTL Project
You will be able to add sources, create block designs in IP Integrator, generate IP, run RTL analysis, synthesis, implementation, design planning and analysis.
[ ] Do not specify sources at this time
[ ] Project is an extensible Vitis platform

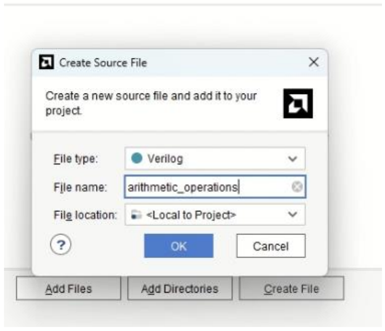
( ) Post-synthesis Project
You will be able to add sources, view device resources, run design analysis, planning and implementation.
[ ] Do not specify sources at this time

( ) I/O Planning Project
Do not specify design sources. You will be able to view part/package resources.

( ) Imported Project
Create a Vivado project from a Synplify Project File.

( ) Example Project
Create a new Vivado project from a predefined template.

5. Click on **create file** add a name ➔arithmetic_operations


Create Source File

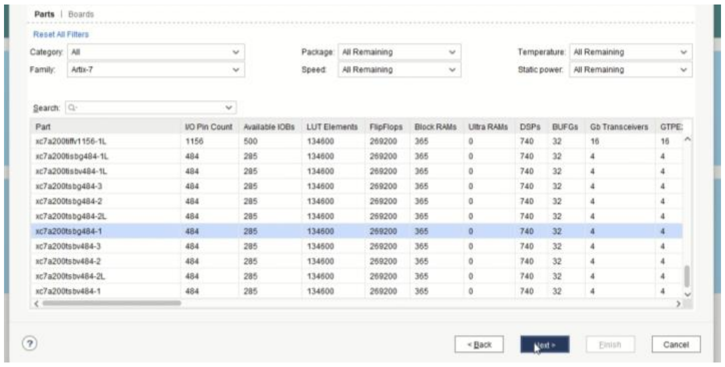
Create a new source file and add it to your project.

File type: Verilog

File name: arithmetic_operations

File location: <Local to Project>

6. Choose remaining same ➔click on **ok** then **next**
7. Choose the board type ➔Artix7 and part of the board click on next

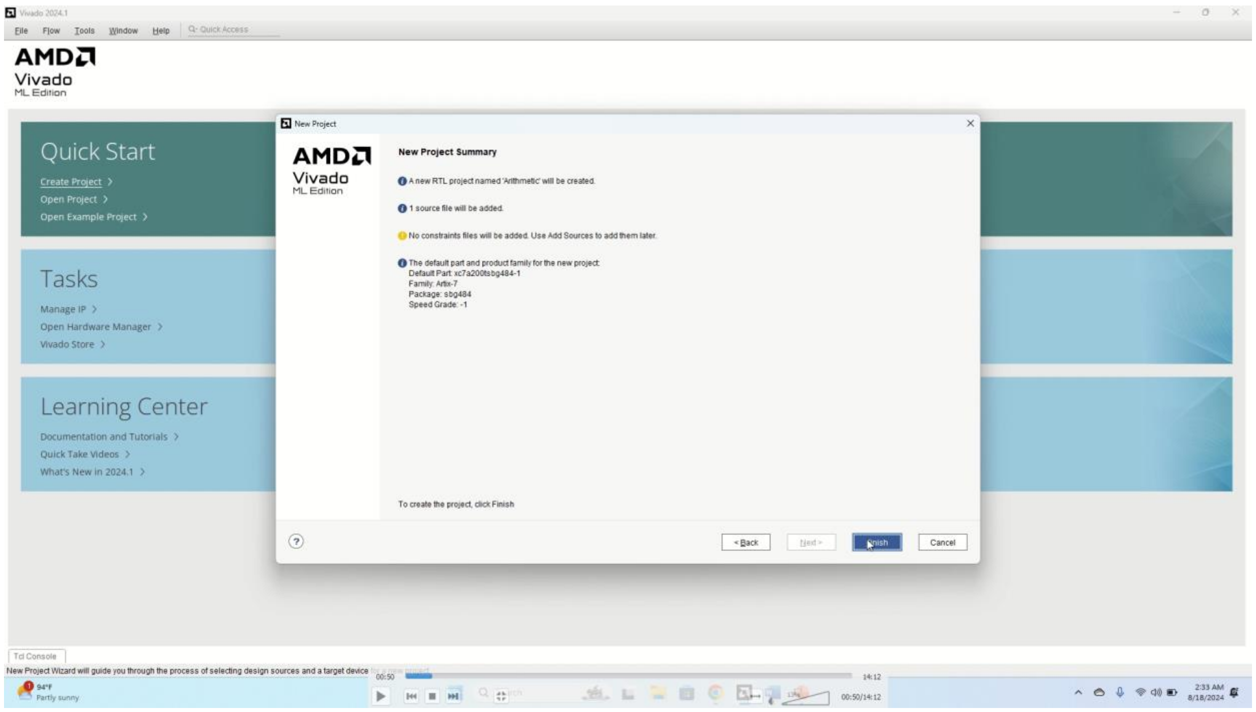


Category: All | Family: Artix-7 | Package: All Remaining | Speed: All Remaining | Temperature: All Remaining | Static power: All Remaining

Search:

Part	I/O Pin Count	Available IOBs	LUT Elements	FlipFlops	Block RAMs	Ultra RAMs	DSPs	BUFGs	Gb Transceivers	GTPE
xc7a200tffv1156-1L	1156	500	134600	269200	365	0	740	32	16	16
xc7a200tisbg484-1L	484	285	134600	269200	365	0	740	32	4	4
xc7a200tisbv484-1L	484	285	134600	269200	365	0	740	32	4	4
xc7a200tsbg484-3	484	285	134600	269200	365	0	740	32	4	4
xc7a200tsbg484-2	484	285	134600	269200	365	0	740	32	4	4
xc7a200tsbg484-2L	484	285	134600	269200	365	0	740	32	4	4
xc7a200tsbg484-1	484	285	134600	269200	365	0	740	32	4	4
xc7a200tsbv484-3	484	285	134600	269200	365	0	740	32	4	4
xc7a200tsbv484-2	484	285	134600	269200	365	0	740	32	4	4
xc7a200tsbv484-2L	484	285	134600	269200	365	0	740	32	4	4
xc7a200tsbv484-1	484	285	134600	269200	365	0	740	32	4	4

8. Click on finish



New Project

New Project Summary

A new RTL project named 'Arithmetic' will be created.

1 source file will be added.

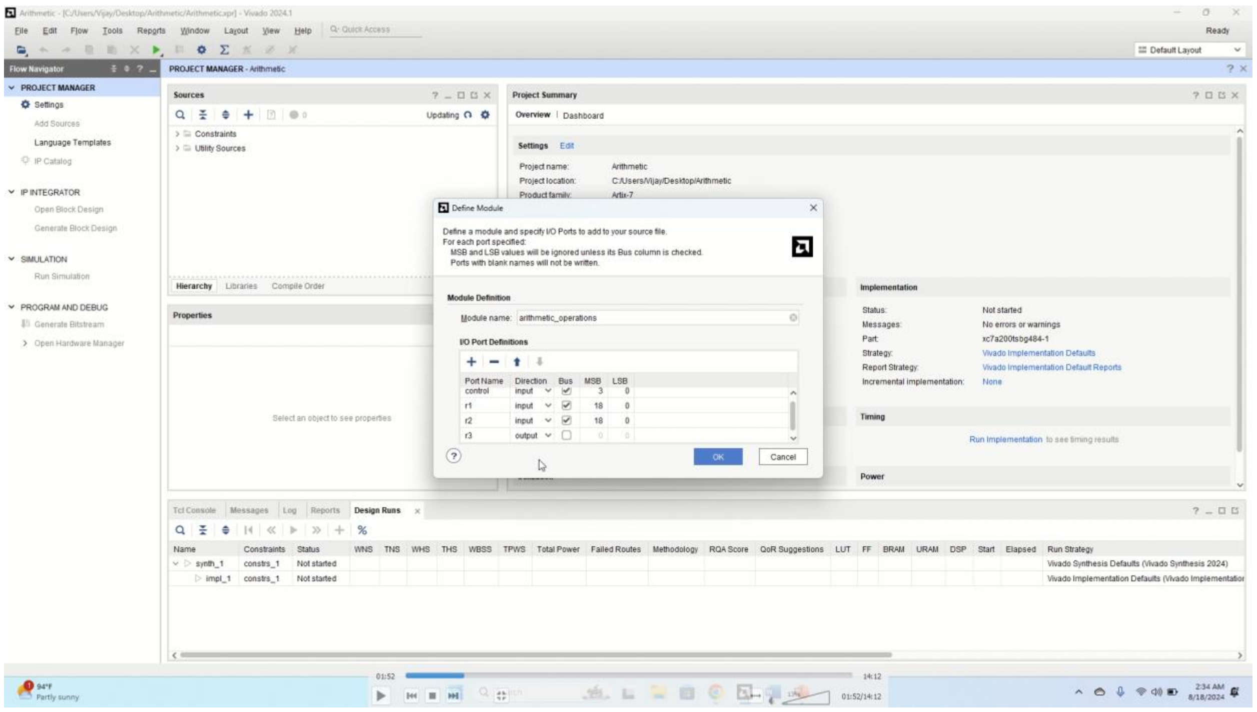
No constraints files will be added. Use Add Sources to add them later.

The default part and product family for the new project:
Default Part: xc7a200tsbg484-1
Family: Artix-7
Package: sbg484
Speed Grade: -1

To create the project, click Finish

New Project Wizard will guide you through the process of selecting design sources and a target device

9. Choose the input and output value or you can click on ok to edit in code part


Define Module

Define a module and specify I/O Ports to add to your source file.
For each port specified:
MSB and LSB values will be ignored unless its Bus column is checked.
Ports with blank names will not be written.

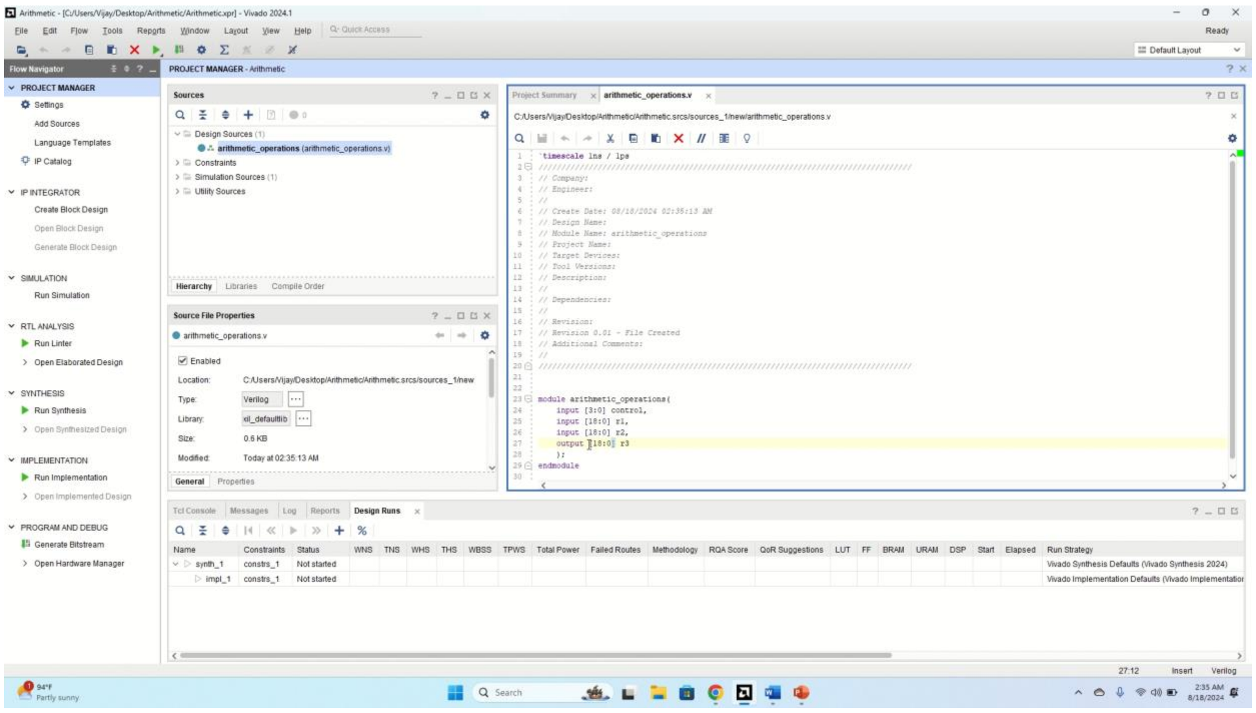
Module Definition

Module name: arithmetic_operations

I/O Port Definitions

Port Name	Direction	Bus	MSB	LSB
control	input	✓	3	0
r1	input	✓	18	0
r2	input	✓	18	0
r3	output	☐	0	0

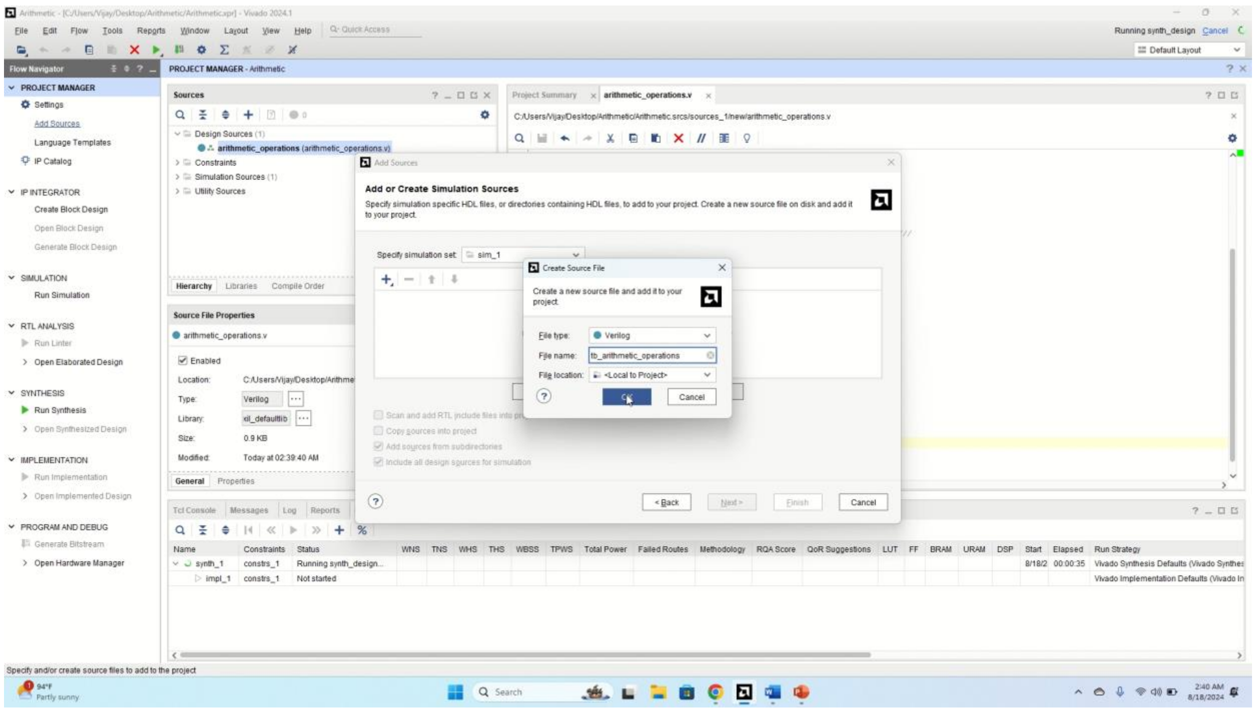
10. Source choose the file arithmetic_operations.v


C:/Users/Vijay/Desktop/Arithmetic/Arithmetic.srcs/sources_1/new/arithmetic_operations.v

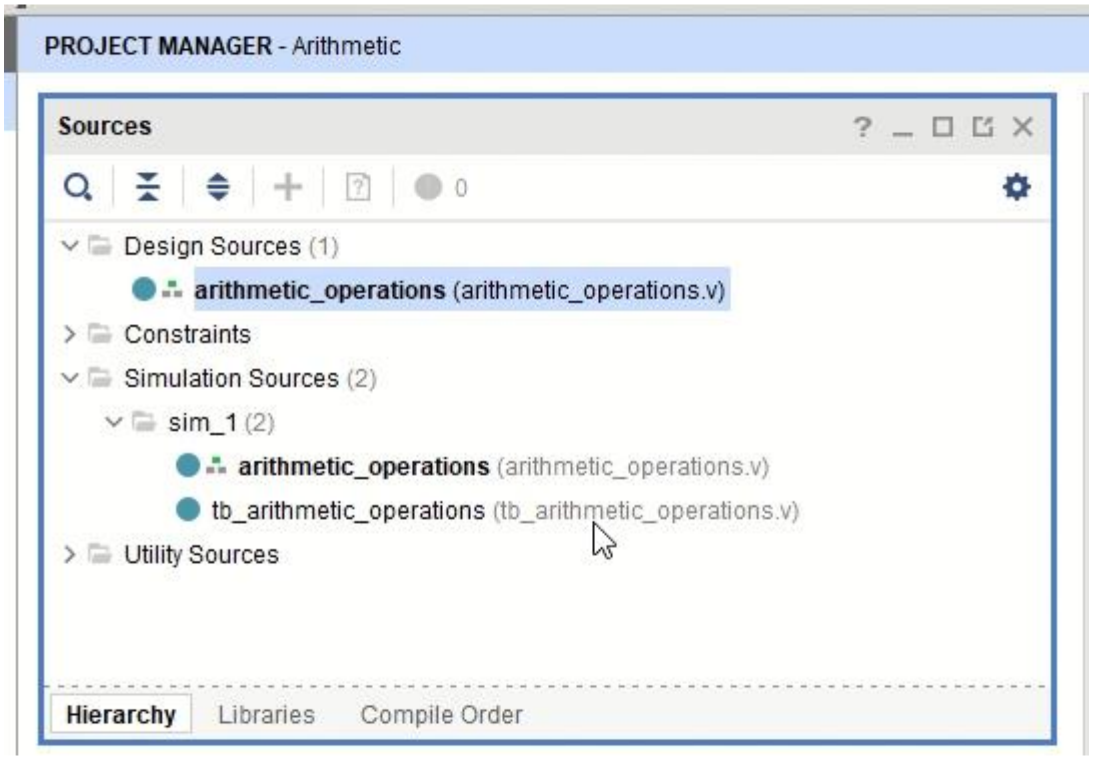
```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 08/18/2024 02:35:13 AM
// Design Name:
// Module Name: arithmetic_operations
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module arithmetic_operations(
    input [3:0] control,
    input [18:0] r1,
    input [18:0] r2,
    output [18:0] r3
    );
endmodule
```

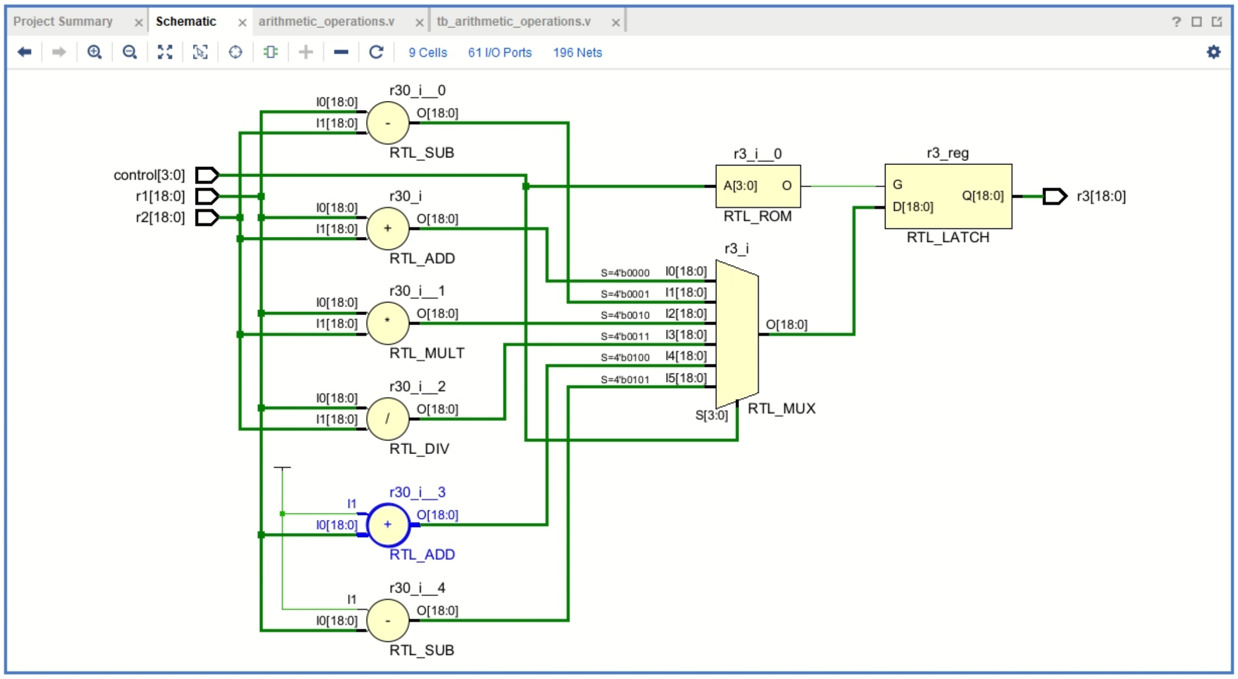
11. Edit the code
12. On the left side click on →SYNTHESIS for checking the syntax errors
13. On the Top Left side →ADD SOURCES give the name tb_arthimetic_operations.v then click on OK


Add or Create Simulation Sources
Specify simulation specific HDL files, or directories containing HDL files, to add to your project. Create a new source file on disk and add it to your project.
Create Source File
Create a new source file and add it to your project.
File type: Verilog
File name: tb_arithmetic_operations
File location: <Local to Project>
OK
Cancel

14. Edit the input and output values or click on ok to edit on inside the file.

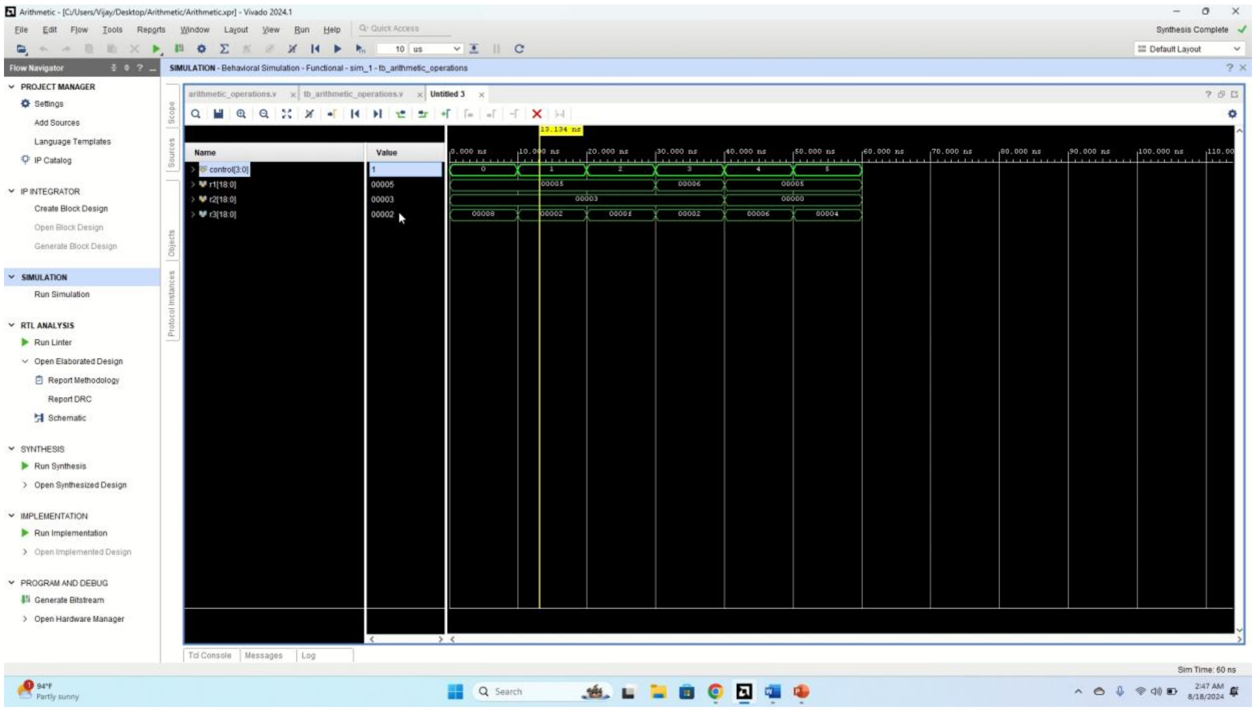

PROJECT MANAGER - Arithmetic
Sources
Design Sources (1)
arithmetic_operations (arithmetic_operations.v)
Constraints
Simulation Sources (2)
sim_1 (2)
arithmetic_operations (arithmetic_operations.v)
tb_arithmetic_operations (tb_arithmetic_operations.v)
Utility Sources
Hierarchy
Libraries
Compile Order

15. Click on tb_arithmetic_operations.v
16. Edit the code and click on → OPEN SYNTHESIZED DESIGN
    →For the Symantic



17. Click on → RUN SIMULATION

→RUN BEHAVIOURAL



Name	Value	0.000 ns	10.000 ns	20.000 ns	30.000 ns	40.000 ns	50.000 ns
control[3:0]	1	0	1	2	3	4	5
r1[18:0]	00005	00005			00006	00005	
r2[18:0]	00003	00003				00000	
r3[18:0]	00002	00008	00002	0000f	00002	00006	00004

HERE IS THE CODE

1. **ARITHMETIC OPERATIONS:**

➔arithmetic_operations.v

```
module arithmetic_operations (
   input [3:0] control,
   input [18:0] r1,
   input [18:0] r2,
   output reg [18:0] r3
);

always @(*) begin
   case (control)
      4'b0000: r3 = r1 + r2;
      4'b0001: r3 = r1 - r2;
      4'b0010: r3 = r1 * r2;
      4'b0011: r3 = r1 / r2;
      4'b0100: r3 = r1 + 1;
      4'b0101: r3 = r1 - 1;
      default: r3 = 0;
   endcase
end
endmodule
```

TEST BENCH CODE FOR ARITHMETIC OPERATIONS

➔tb_ arithmetic_operations.v

```
module tb_arithmetic_operations;
reg [3:0] control;
reg [18:0] r1;
reg [18:0] r2;
wire [18:0] r3;
arithmetic_operations uut (
    .control(control),
    .r1(r1),
    .r2(r2),
    .r3(r3)
);

initial begin
    control = 4'b0000;
    r1 = 19'b0000000000000000101;
    r2 = 19'b0000000000000000011;
    #10;
    $display("ADD: r1 = %d, r2 = %d, r3 = %d", r1, r2, r3);
    control = 4'b0001;
    r1 = 19'b0000000000000000101;
    r2 = 19'b0000000000000000011;
    #10;
    $display("SUB: r1 = %d, r2 = %d, r3 = %d", r1, r2, r3);

    control = 4'b0010;
    r1 = 19'b0000000000000000101;
    r2 = 19'b0000000000000000011;
    #10;
    $display("MUL: r1 = %d, r2 = %d, r3 = %d", r1, r2, r3);
    control = 4'b0011;
    r1 = 19'b0000000000000000110;
    r2 = 19'b0000000000000000011;
    #10;
    $display("DIV: r1 = %d, r2 = %d, r3 = %d", r1, r2, r3);
    control = 4'b0100;
    r1 = 19'b0000000000000000101;
    r2 = 19'b0;
    #10;
    $display("INC: r1 = %d, r3 = %d", r1, r3);
    control = 4'b0101;
    r1 = 19'b0000000000000000101; // 5
    r2 = 19'b0; // Not used
    #10;
    $display("DEC: r1 = %d, r3 = %d", r1, r3);
    $finish;
end
endmodule
```

HERE IS THE CODE

2. LOGIC OPERATIONS

➔logical_operations.v

```
module logical_operations (
    input [3:0] control,
    input [18:0] r1,
    input [18:0] r2,
    output reg [18:0] r3
);
```

```
always @(*) begin
    case (control)
        4'b0000: r3 = r1 & r2;
        4'b0001: r3 = r1 | r2;
        4'b0010: r3 = r1 ^ r2;
        4'b0011: r3 = ~r2;
        default: r3 = 0;
    endcase
end
endmodule
```

TEST BENCH CODE FOR LOGICAL OPERATIONS

➔tb_ logical_operations.v

```
module tb_logical_operations;
reg [3:0] control;
reg [18:0] r1;
reg [18:0] r2;
wire [18:0] r3;
logical_operations uut (
    .control(control),
    .r1(r1),
    .r2(r2),
    .r3(r3)
);

initial begin
    control = 4'b0000;
    r1 = 19'b0000000000000000101;
    r2 = 19'b0000000000000000011;
    #10;
    $display("AND: r1 = %b, r2 = %b, r3 = %b", r1, r2, r3);

    control = 4'b0001;
    r1 = 19'b0000000000000000101;
    r2 = 19'b0000000000000000011;
    #10;
    $display("OR: r1 = %b, r2 = %b, r3 = %b", r1, r2, r3);
    control = 4'b0010;
    r1 = 19'b0000000000000000101;
    r2 = 19'b0000000000000000011;
    #10;
    $display("XOR: r1 = %b, r2 = %b, r3 = %b", r1, r2, r3);
    control = 4'b0011;
    r1 = 19'b0000000000000000101;
    r2 = 19'b0000000000000000101;
    #10;
    $display("NOT: r1 = %b (ignored), r2 = %b, r3 = %b", r1, r2, r3);
    $finish;
end
endmodule
```

HERE IS THE CODE

3. **CONTROL FLOW**

➔Control_flow.v

```
module control_flow (
    input [3:0] control,
    input [18:0] r1,
    input [18:0] r2,
    input [18:0] addr,
    input clk,
    input reset,
    output reg [18:0] PC,
    output reg [18:0] SP,
    inout [18:0] stack
);

reg [18:0] stack_mem [0:255];
reg [18:0] stack_pointer;
initial begin
    stack_pointer = 19'b0;
end

always @(posedge clk or posedge reset) begin
    if (reset) begin
        PC <= 19'b0;
        SP <= 19'b0;
        stack_pointer <= 19'b0;
    end else begin
        case (control)
            4'b0000: PC <= addr;           /
            4'b0001: if (r1 == r2) PC <= addr;
            4'b0010: if (r1 != r2) PC <= addr;
            4'b0011: begin
```

```
                stack_mem[stack_pointer] <= PC + 1;
                stack_pointer <= stack_pointer + 1;
                PC <= addr;
            end
            4'b0100: begin
                stack_pointer <= stack_pointer - 1;
                PC <= stack_mem[stack_pointer];
            end
            default: PC <= PC;
        endcase
    end
end
endmodule
```

TEST BENCH CODE FOR CONTROL FLOW

➔Tb_control_flow.v

```
module tb_control_flow;


reg [3:0] control;
reg [18:0] r1;
reg [18:0] r2;
reg [18:0] addr;
reg clk;
reg reset;
wire [18:0] PC;
wire [18:0] SP;
reg [18:0] stack [0:255];
control_flow uut (
    .control(control),
    .r1(r1),
    .r2(r2),
    .addr(addr),
    .clk(clk),
    .reset(reset),
    .PC(PC),
    .SP(SP),
    .stack(stack)
);
initial begin
    clk = 0;
    forever #5 clk = ~clk;
end

initial begin
    reset = 1;
    control = 4'b0000;
    r1 = 19'b0;
    r2 = 19'b0;
    addr = 19'b0000000000000000101;
    #10;
    reset = 0;
    control = 4'b0000;
    #10;
    $display("JMP: PC = %d", PC);
    control = 4'b0001;
    r1 = 19'b0000000000000000101;
    r2 = 19'b0000000000000000101;
    #10;
    $display("BEQ: r1 = %d, r2 = %d, PC = %d", r1, r2, PC);
    control = 4'b0010;
    r1 = 19'b0000000000000000101;
    r2 = 19'b0000000000000000110;
    #10;
    $display("BNE: r1 = %d, r2 = %d, PC = %d", r1, r2, PC);
    control = 4'b0011;
    addr = 19'b0000000000000001000;
    #10;
    $display("CALL: PC = %d, SP = %d", PC, SP);
    control = 4'b0100;
    #10;
    $display("RET: PC = %d, SP = %d", PC, SP);
    $finish;
End
```

HERE IS THE CODE

4. **MEMORY ACCESS**

➔Memory_access.v

```
module memory_access (
    input [3:0] control,
    input [18:0] r1,
    input [18:0] addr,
    input clk,
    input reset,
    output reg [18:0] r2,
    inout [18:0] memory
);
reg [18:0] memory_array [0:255];
always @(posedge clk or posedge reset) begin
    if (reset) begin
        r2 <= 19'b0;
    end else begin
        case (control)
            4'b0000: begin
                r2 <= memory_array[addr];
            end
            4'b0001: begin // ST addr, r1
                memory_array[addr] <= r1;
            end
            default: begin
                r2 <= r2;
            end
        endcase
    end
end
endmodule
```

TEST BENCH CODE FOR MEMORY ACCESS

➔Tb_memory_access

```
module tb_memory_access;
reg [3:0] control;
reg [18:0] r1;
reg [18:0] addr;
reg clk;
reg reset;
wire [18:0] r2;
reg [18:0] memory [0:255];

// Instantiate the module
memory_access uut (
    .control(control),
    .r1(r1),
    .addr(addr),
    .clk(clk),
    .reset(reset),
    .r2(r2),
    .memory(memory)
);

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

initial begin
 reset = 1;
    control = 4'b0000;
    r1 = 19'b0;
    addr = 19'b0000000000000000101;
    #10;
    reset = 0;
    control = 4'b0001;
    r1 = 19'b0000000000000000110;
    addr = 19'b0000000000000000101;
    #10;
    $display("ST: addr = %d, r1 = %d, memory[addr] = %d", addr, r1, memory[addr]);
    control = 4'b0000;
    addr = 19'b0000000000000000101;
    #10;
    $display("LD: addr = %d, r2 = %d", addr, r2);
    $finish;
end
endmodule
```

HERE IS THE CODE

5. CUSTOM INSTRUCTION FOR SPECIALIZED OPERATIONS

➔Custom.v

```
module custom (
    input [3:0] control,
    input [18:0] r1,
    input [18:0] r2,
    inout [15:0] memory [0:255]
);

reg [15:0] temp_data;
reg [15:0] result_data;
```

```
function [15:0] FFT(input [15:0] data);
    begin
        FFT = data;
    end
endfunction
function [15:0] Encrypt(input [15:0] data);
    begin
        Encrypt = data;
    end
endfunction
function [15:0] Decrypt(input [15:0] data);
    begin
        Decrypt = data;
    end
endfunction

always @(*) begin
    case (control)
        4'b0000: begin
            temp_data = memory[r2];
            result_data = FFT(temp_data);
            memory[r1] = result_data;
        end

        4'b0001: begin
            temp_data = memory[r2];
            result_data = Encrypt(temp_data);
            memory[r1] = result_data;
        end

        4'b0010: begin
            temp_data = memory[r2];
            result_data = Decrypt(temp_data);
            memory[r1] = result_data;
        end

        default: begin
            // Handle other operations or default case
            result_data = 16'b0;
            memory[r1] = result_data;
        end
    endcase
end

endmodule
```

TEST BENCH CODE FOR CUSTOM INSTRUCTIONS FOR SPECIALIZED OPERATIONS

➔Tb_custom.v

```
module tb_custom;
reg [3:0] control;
reg [18:0] r1;
reg [18:0] r2;
reg [15:0] memory [0:255];
wire [15:0] mem_out;
```

```
data_operations uut (
    .control(control),
    .r1(r1),
    .r2(r2),
    .memory(memory)
);
initial begin
    control = 4'b0000;
    r1 = 19'd1;
    r2 = 19'd0;
    memory[r2] = 16'd123;
    #10;
    $display("FFT: memory[%0d] = %d", r1, memory[r1]);
    control = 4'b0001;
    r1 = 19'd2;
    r2 = 19'd1;
    memory[r2] = 16'd456;
    #10;
    $display("ENC: memory[%0d] = %d", r1, memory[r1]);
    control = 4'b0010;
    r1 = 19'd3;
    r2 = 19'd2;
    memory[r2] = 16'd789;
    #10;
    $display("DEC: memory[%0d] = %d", r1, memory[r1]);
    $finish;
end
```

```
endmodule
```