Week 6

1. Write a SV program for Asynchronous Reset Flip Flop

module async\_reset\_flip\_flop (

input logic clk, // Clock input

input logic reset, // Asynchronous reset input

input logic d, // Data input

output logic q // Output of the flip-flop

);

// Always block with sensitivity list including reset and clock

always\_ff @(posedge clk or posedge reset) begin

if (reset) begin

q <= 0; // On reset, output q is set to 0

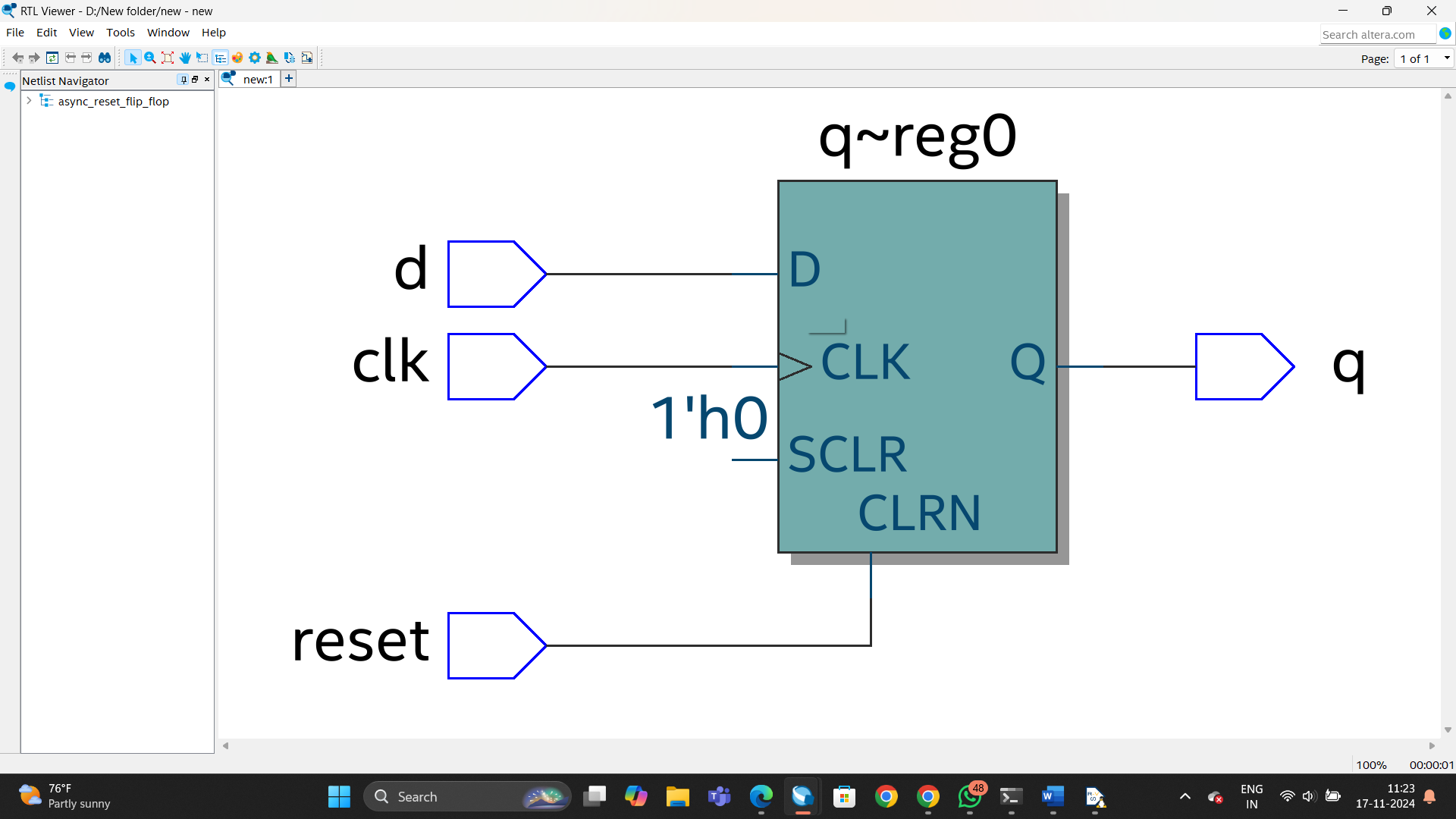
end else begin

q <= d; // On the rising edge of the clock, output follows input d

end

end

endmodule



1. Write a SV program for Synchronous Reset Flip Flop

module sync\_reset\_flip\_flop (

input logic clk, // Clock input

input logic reset, // Synchronous reset input

input logic d, // Data input

output logic q // Output of the flip-flop

);

// Always block triggered by the positive edge of the clock

always\_ff @(posedge clk) begin

if (reset) begin

q <= 0; // When reset is active, set output q to 0

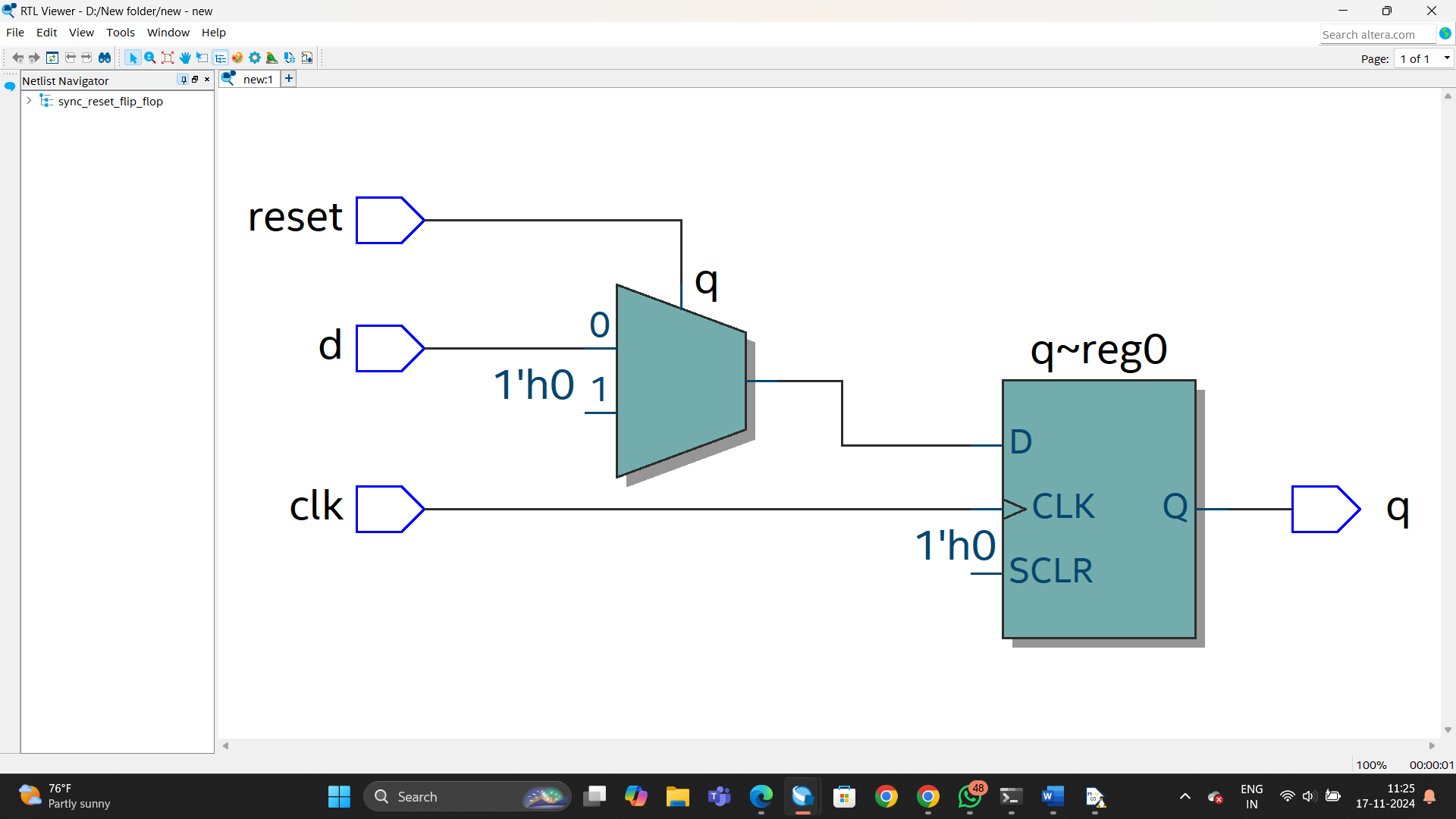
end else begin

q <= d; // On clock rising edge, output follows input d

end

end

endmodule



1. Write a SV program for 3:1 MUX

module mux3to1 (

input logic d0, // Data input 0

input logic d1, // Data input 1

input logic d2, // Data input 2

input logic s1, // Select input 1

input logic s0, // Select input 0

output logic y // Output

);

// Always block to implement the MUX functionality

always\_comb begin

case ({s1, s0})

2'b00: y = d0; // When s1s0 = 00, output is d0

2'b01: y = d1; // When s1s0 = 01, output is d1

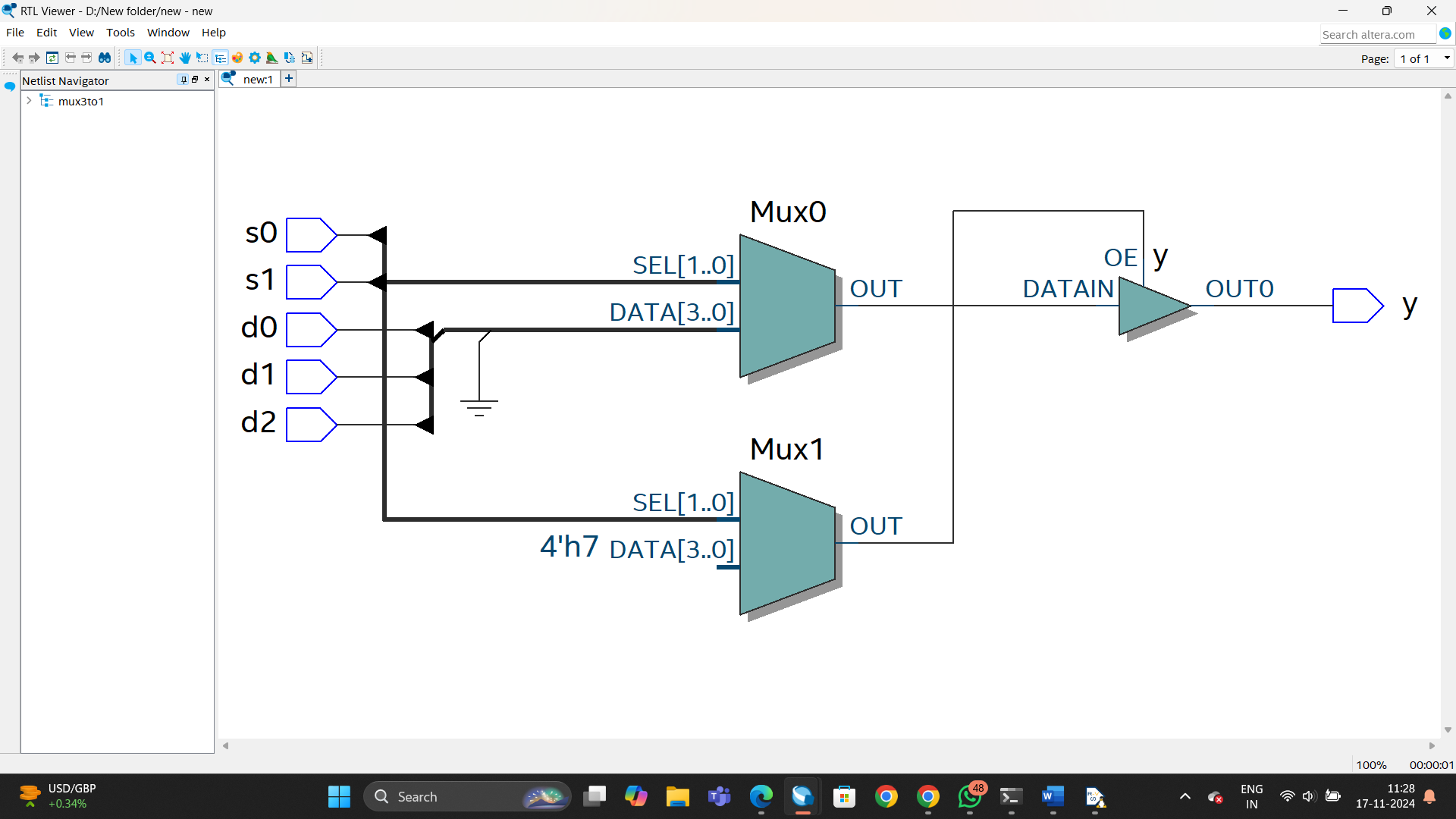
2'b10: y = d2; // When s1s0 = 10, output is d2

default: y = 1'bz; // Undefined condition (s1s0 = 11), high impedance state

endcase

end

endmodule



1. Write a SV program for 2:1 MUX

module mux\_1 #(parameter N=32)(a,b,s,y);

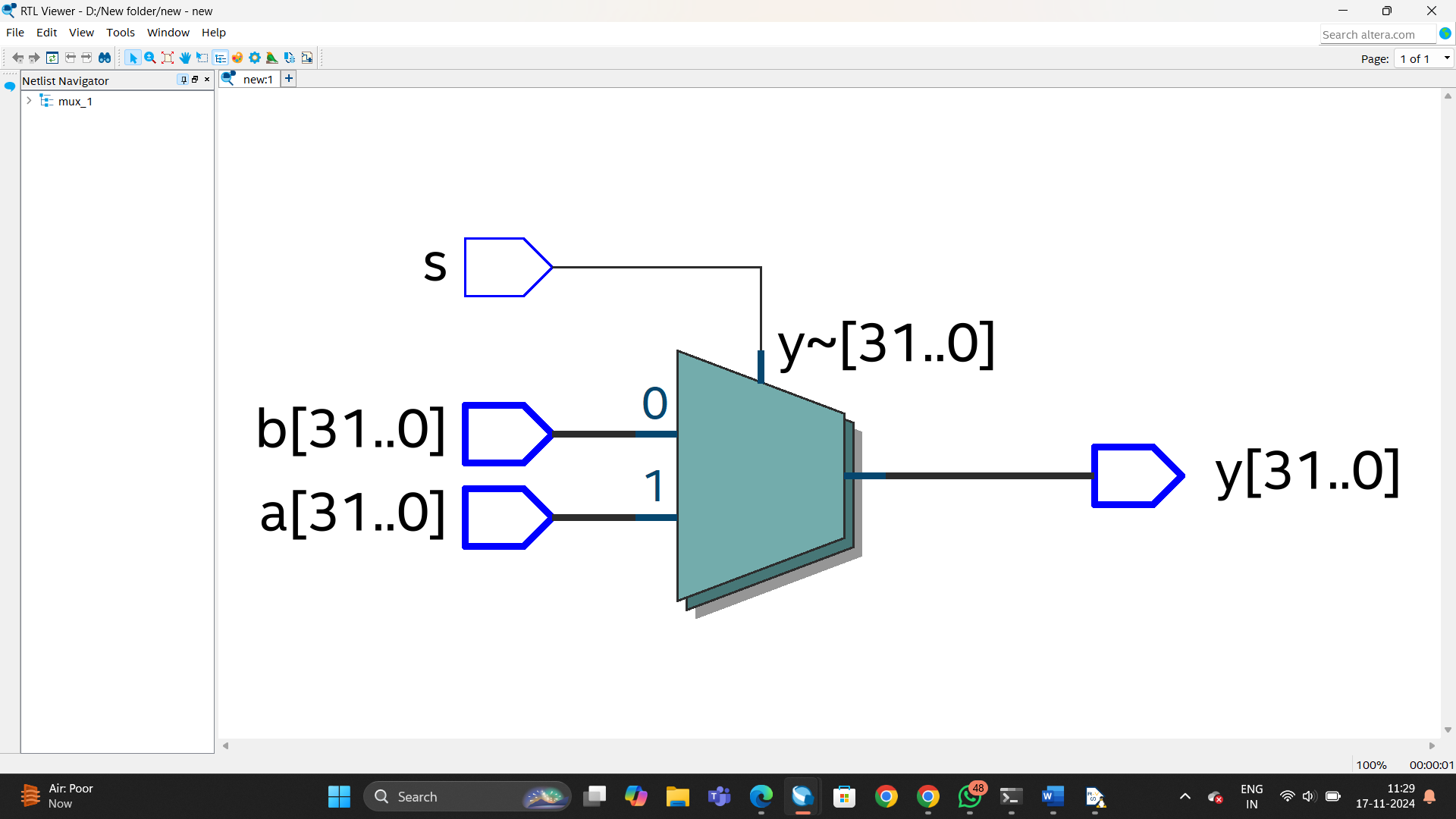
input logic [N-1:0] a,b;

output logic [N-1:0]y;

input logic s;

assign y=s?a:b;

endmodule



1. Write a SV program for Adder

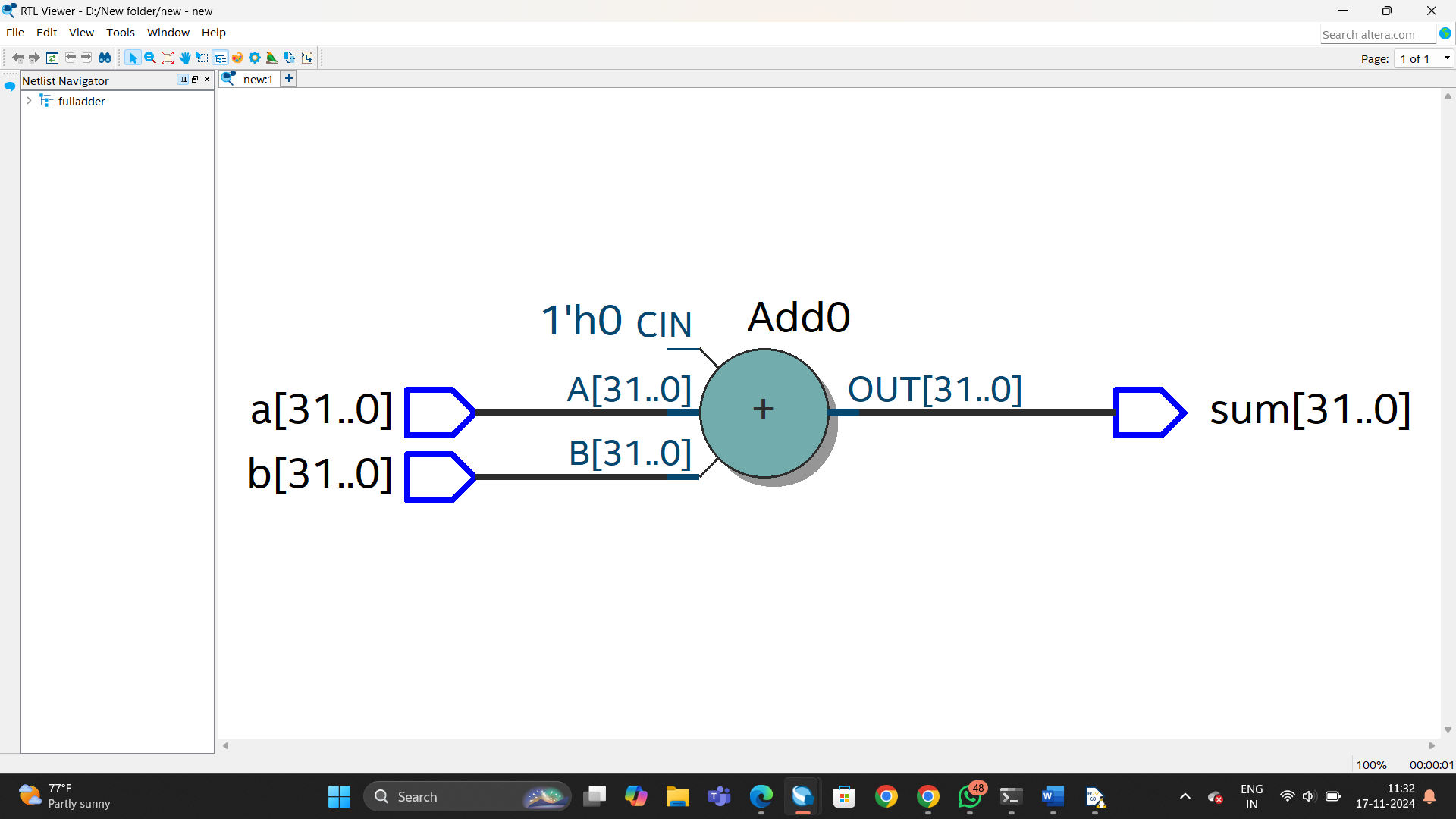
module fulladder #(parameter N=32)(a,b,sum);

input logic [N-1:0] a,b;

output logic [N-1:0] sum;

assign sum=a+b;

endmodule



1. Write a SV program for Instruction Fetch Stage

a)

module PC #(parameter N=32) (input logic clk,input logic rst,input logic [N-1:0] pc\_next,output logic [N-1:0] pc);

always\_ff@(posedge clk)

begin

if (rst)

pc<= 32'b0 ;

else

pc<= pc\_next ;

end

endmodule

b)

module instr\_mem #(parameter W=32, L=1024) (addr,instr,reset);

initial

begin

$readmemh("program\_dump.hex",mem);

end

input logic reset;

input logic [W-1:0] addr;

output logic [W-1:0] instr;

logic [W-1:0] mem [0:L-1];

assign instr=reset?0:mem[addr];

endmodule

c)

module instr\_fetch #(parameter N=32) (input logic clk,input logic reset, input logic sel, input logic [N-1:0]pc\_imm, output logic [N-1:0]pc\_new,output logic [N-1:0]inst,pc);

wire [N-1:0]line1;

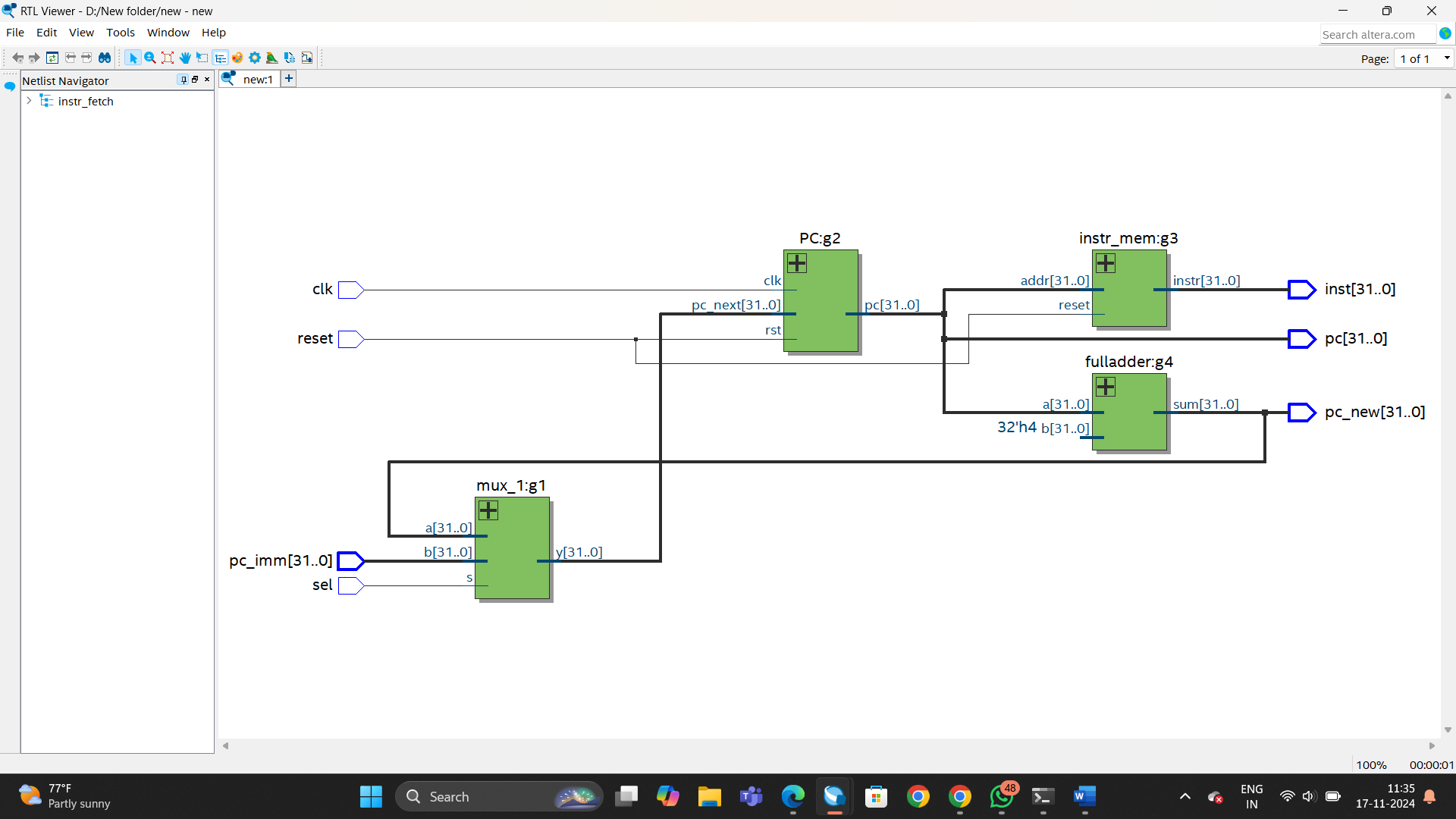
mux\_1 #(32) g1(pc\_new,pc\_imm,sel,line1);

PC #(32) g2(clk,reset,line1,pc);

instr\_mem #(32) g3(pc,inst,reset);

fulladder #(32) g4(pc,32'd4,pc\_new);

endmodule



1. Write a SV program for Instruction Memory

module instr\_mem #(parameter W=32, L=1024) (addr,instr,reset);

initial

begin

$readmemh("program\_dump.hex",mem);

end

input logic reset;

input logic [W-1:0] addr;

output logic [W-1:0] instr;

logic [W-1:0] mem [0:L-1];

assign instr=reset?0:mem[addr];

endmodule

