ALU:-

Design code:-

module alu(opa,opb,cin,clk,rst,ce,mode,cmd,inp\_valid,res,oflow,cout,g,l,e,err);

parameter size=8;

input signed [size-1:0] opa,opb;

input cin,clk,rst,ce,mode;

input [3:0] cmd;

input [1:0] inp\_valid;

output reg [size:0] res;

output reg oflow,cout,g,l,e,err;

always@(posedge clk or posedge rst)

begin

if(rst)

begin

res<=0;

oflow<=0;

cout<=0;

g<=0;

l<=0;

e<=0;

end

else if(ce)

begin

if(mode)

begin

case(inp\_valid)

2'b00: ;

2'b01:

begin

case(cmd)

4'b0100:{cout,res}<=opa+1;

4'b0101:{cout,res}<=opa-1;

default:{cout,res}<=0;

endcase

end

2'b10:

begin

case(cmd)

4'b0110:{cout,res}<=opb+1;

4'b0111:{cout,res}<=opb-1;

default:{cout,res}<=0;

endcase

end

2'b11:

begin

case(cmd)

4'b0000:{cout,res}<=opa+opb;

4'b0001:{cout,res}<=(opa-opb);

4'b0010:{cout,res}<=(opa+opb)+cin;

4'b0011:{cout,res}<=(opa-opb)-cin;

4'b0100:{cout,res}<=opa+1;

4'b0101:{cout,res}<=opa-1;

4'b0110:{cout,res}<=opb+1;

4'b0111:{cout,res}<=opb-1;

4'b1000:

begin

g<=(opa>opb)?1'b1:1'b0;

l<=(opa<opb)?1'b1:1'b0;

e<=(opa==opb)?1'b1:1'b0;

end

4'b1001:res<=(opa+1)\*(opb+1);

4'b1010:res<=((opa<<1)\*opb);

default:begin

{cout,res}<=0;

{g,l,e}<={1'b0,1'b0,1'b0};

end

endcase

end

default:{cout,res}<={cout,res};

endcase

oflow<=cout;

end

if(!mode)

begin

case(inp\_valid)

2'b00:;

2'b01:

begin

case(cmd)

4'b0110:res<=~opa;

4'b1000:res<=opa>>1;

4'b1001:res<=opa<<1;

default:res<=0;

endcase

end

2'b10:

begin

case(cmd)

4'b0111:res<=~opb;

4'b1010:res<=opb>>1;

4'b1011:res<=opb<<1;

default:res<=0;

endcase

end

2'b11:

begin

case(cmd)

4'b0000:res<=opa&opb;

4'b0001:res<=~(opa&opb);

4'b0010:res<=opa|opb;

4'b0011:res<=~(opa|opb);

4'b0100:res<=opa^opb;

4'b0101:res<=~(opa^opb);

4'b0110:res<=~opa;

4'b0111:res<=~opb;

4'b1000:res<=opa>>1;

4'b1001:res<=opa<<1;

4'b1010:res<=opb>>1;

4'b1011:res<=opb<<1;

4'b1100:

begin

casex(opb)

8'b0000x000:{res,err}<={opa,1'b0};

8'b0000x001:{res,err}<={{opa[size-2:0],opa[size-1]},1'b0};

8'b0000x010:{res,err}<={{opa[size-3:0],opa[size-1:size-2]},1'b0};

8'b0000x011:{res,err}<={{opa[size-4:0],opa[size-1:size-3]},1'b0};

8'b0000x100:{res,err}<={{opa[size-5:0],opa[size-1:size-4]},1'b0};

8'b0000x101:{res,err}<={{opa[size-6:0],opa[size-1:size-5]},1'b0};

8'b0000x110:{res,err}<={{opa[size-7:0],opa[size-1:size-6]},1'b0};

8'b0000x111:{res,err}<={{opa[0],opa[size-1:size-6]},1'b0};

default:{res,err}<={9'd0,1'b1};

endcase

end

4'b1101:

begin

casex(opb)

8'b0000x000:{res,err}<={opa,1'b0};

8'b0000x001:{res,err}<={{opa[0],opa[size-1:size-6]},1'b0};

8'b0000x010:{res,err}<={{opa[size-7:0],opa[size-1:size-6]},1'b0};

8'b0000x011:{res,err}<={{opa[size-6:0],opa[size-1:size-5]},1'b0};

8'b0000x100:{res,err}<={{opa[size-5:0],opa[size-1:size-4]},1'b0};

8'b0000x101:{res,err}<={{opa[size-4:0],opa[size-1:size-3]},1'b0};

8'b0000x110:{res,err}<={{opa[size-3:0],opa[size-1:size-2]},1'b0};

8'b0000x111:{res,err}<={{opa[size-2:0],opa[size-1]},1'b0};

default:{res,err}<={9'd0,1'b1};

endcase

end

default:res<=0;

endcase

end

default:res<=res;

endcase

end

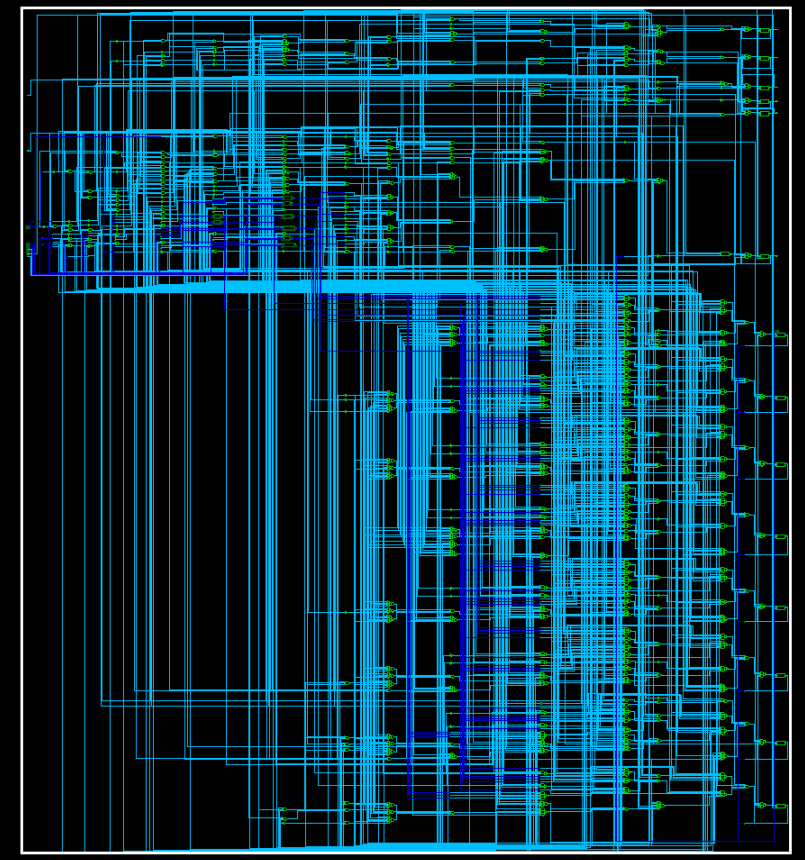
end

else res<=res;

end

endmodule

Schematic:-



1101 Sequence detector :-

Design code:-

module mealy\_1101\_nonoverlap(clk,rst,seq\_in,detect\_out);

input clk,rst,seq\_in;

output reg detect\_out;

parameter idle=2'b00,one=2'b01,oneone=2'b10,oneonezero=2'b11;

reg [1:0] current\_state,next\_state;

always@(posedge clk or posedge rst)

begin

if(rst==1)

current\_state<=idle;

else

current\_state<=next\_state;

end

always@(current\_state or seq\_in)

begin

case(current\_state)

idle:

begin

if(seq\_in==1)

begin

next\_state=one;

detect\_out=1'b0;

end

else

begin

next\_state=idle;

detect\_out=1'b0;

end

end

one:

begin

if(seq\_in==1)

begin

next\_state=oneone;

detect\_out=1'b0;

end

else

begin

next\_state=idle;

detect\_out=1'b0;

end

end

oneone:

begin

if(seq\_in==0)

begin

next\_state=oneonezero;

detect\_out=1'b0;

end

else

begin

next\_state=oneone;

detect\_out=1'b0;

end

end

oneonezero:

begin

if(seq\_in==1)

begin

next\_state=idle;

detect\_out=1'b1;

end

else

begin

next\_state=idle;

detect\_out=1'b0;

end

end

default:

begin

next\_state = idle;

detect\_out = 1'b0;

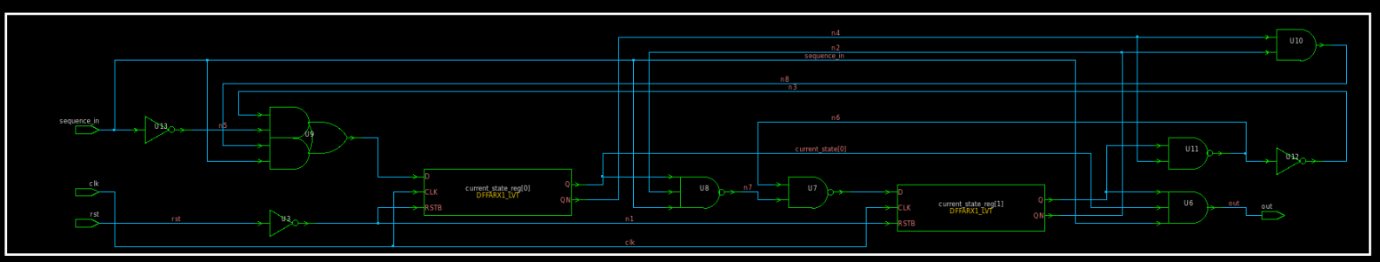
end

endcase

end

endmodule

Schematic:-



USR:-

Design code:-

module usr(mode,rst,data\_in,data\_out,clk,msb\_in,lsb\_in);

input [1:0] mode;

input [3:0] data\_in;

output reg [3:0] data\_out;

input msb\_in,lsb\_in,clk,rst;

always@(posedge clk)

begin

if(!rst)data\_out<=0;

case(mode)

2'b00:data\_out<=data\_out;

2'b01:data\_out<={msb\_in,data\_in[3:1]};

2'b10:data\_out<={data\_in[2:0],lsb\_in};

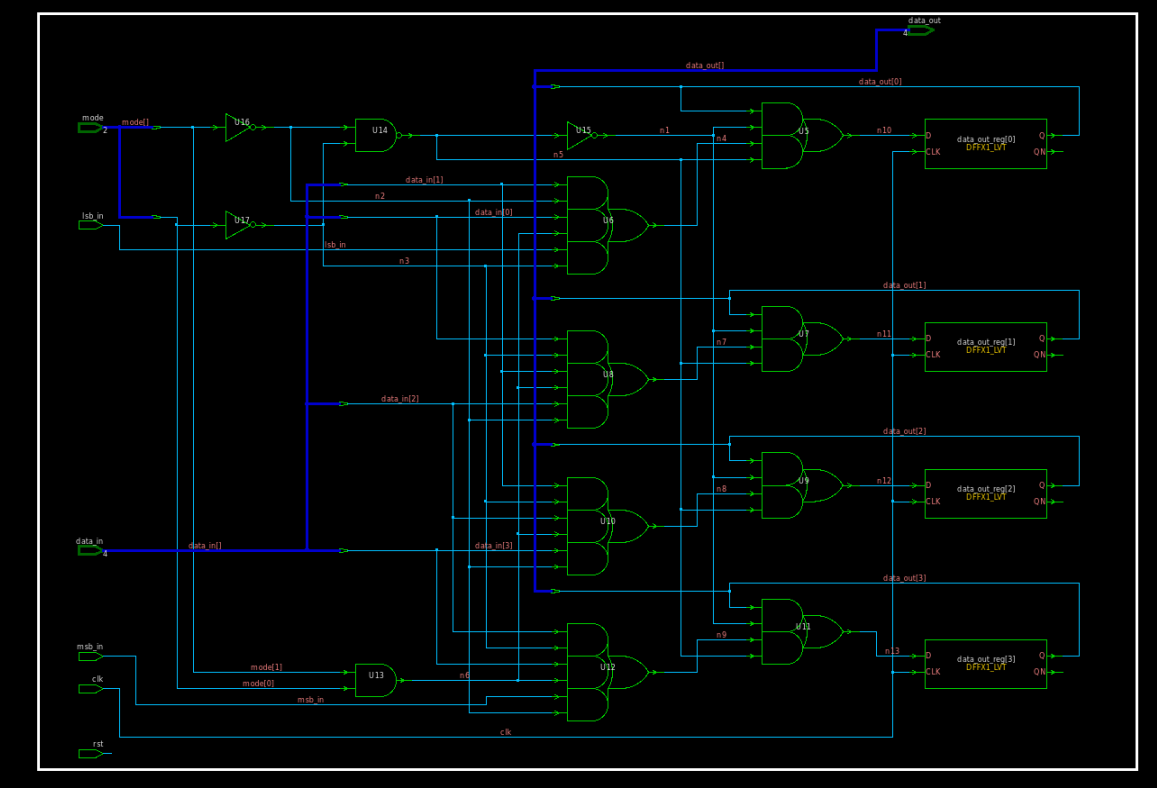
2'b11:data\_out<=data\_in;

default:data\_out<= 4'b0000;

endcase

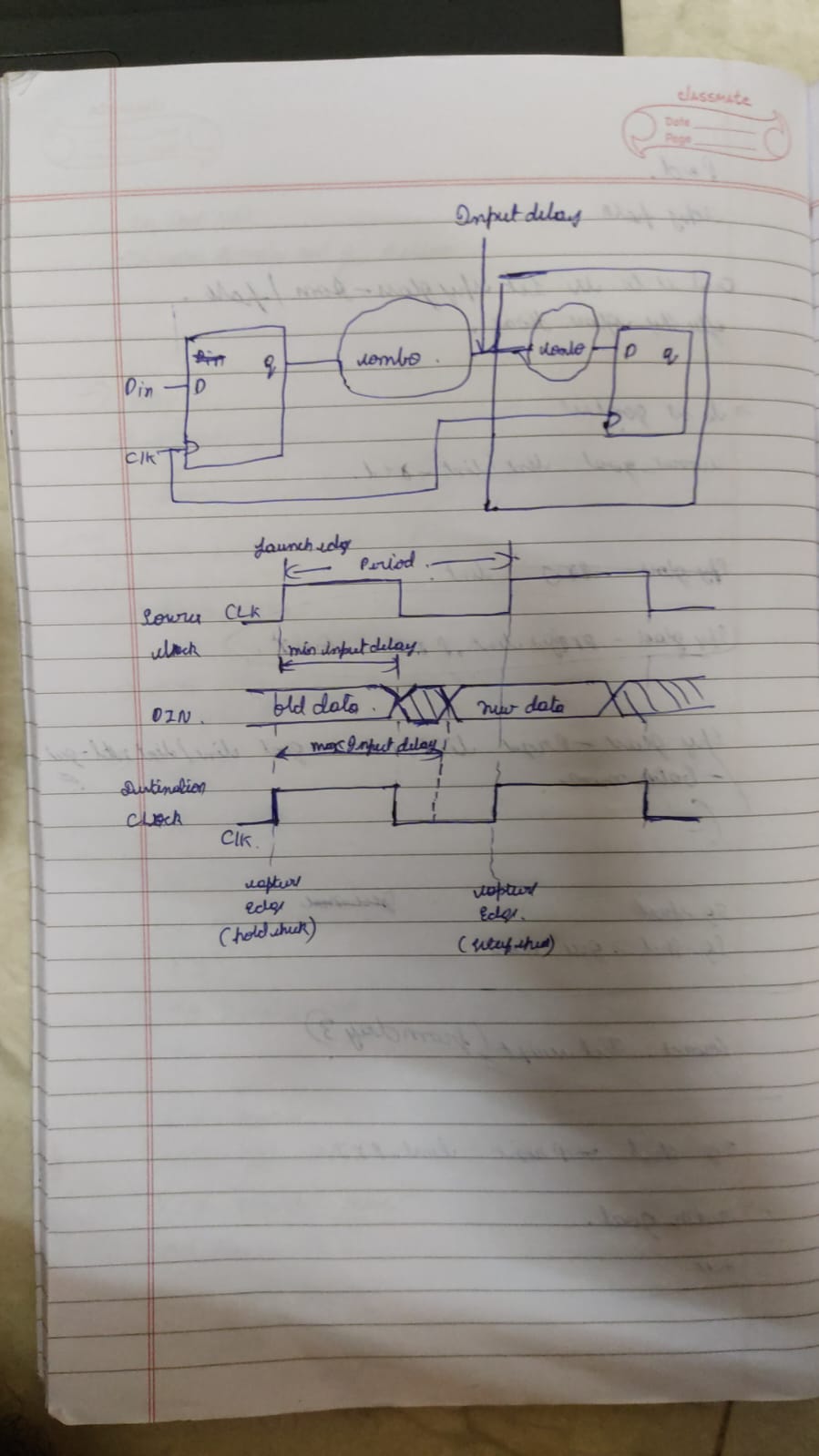
end

endmodule

Schematic:

Q1. What is input delay? explain with the help of diagram...How to constrain input ports?  
   How to review input port timing?

Input delay refers to the time taken for a signal to travel from an external source (e.g., a chip pin) to the input pin of a register. This delay includes the time for the signal to propagate through the input buffer and any additional routing within the chip before reaching the destination register.



To constrain input ports,

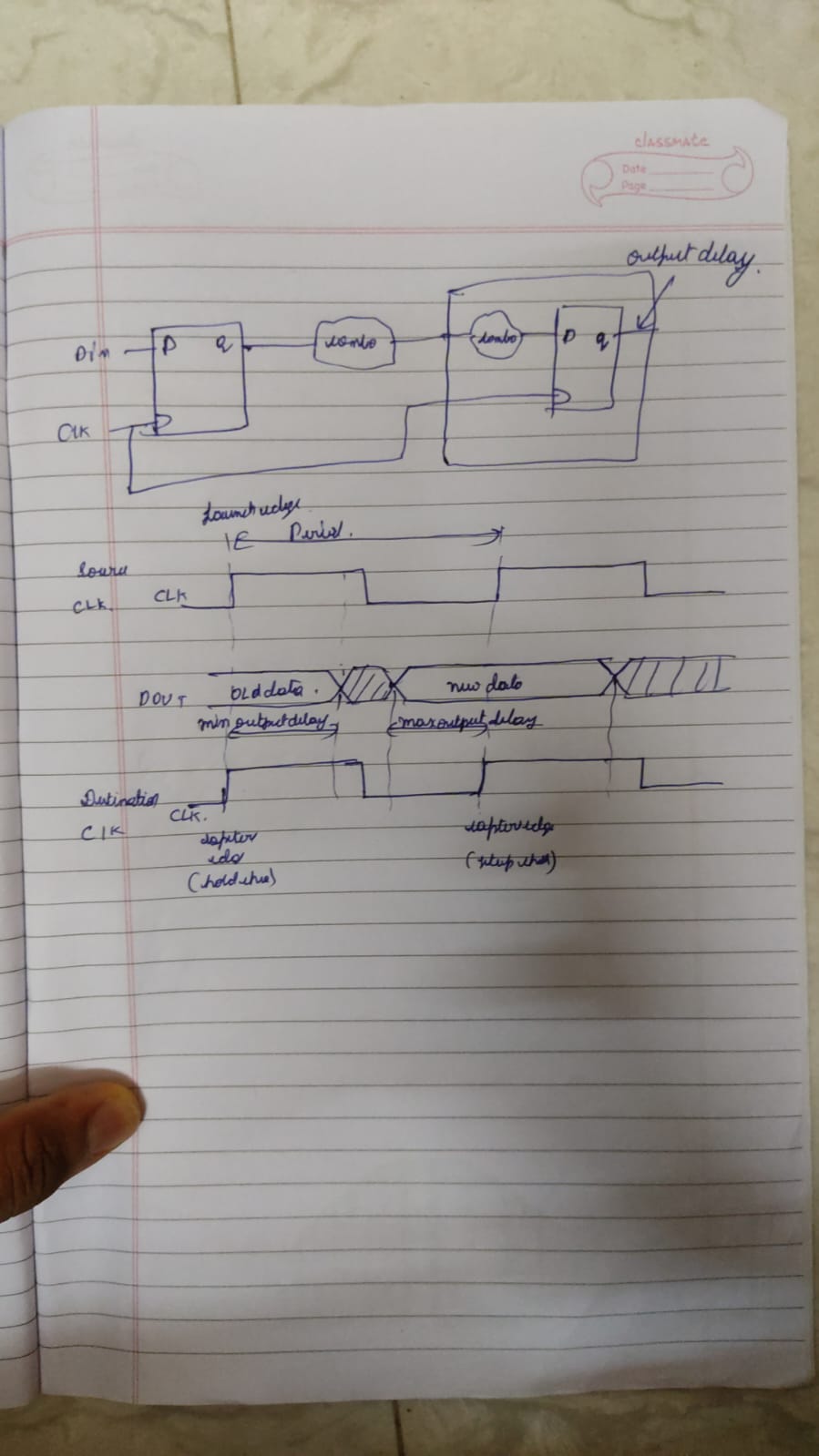
you use the set\_input\_delay command in the SDC (Synopsys Design Constraints) file.

This command specifies the delay from an external source to the input port of the FPGA/ASIC.

After defining design and constraints, use the timing analysis features within your digital design software. These tools analyze critical paths (the longest paths) from inputs to outputs to verify if signals arrive within the specified timing windows.

Q2. What is output delay? explain with the help of diagram...How to constrain output ports?  
   How to review output port timing?

The time it takes for a processed signal to stabilize and become available at the output port of a digital circuit. This duration includes delays caused by internal logic operations and the output driver circuitry.



To ensure the output signal timing is correct, specify output delay constraints. This guarantees that the sending circuit does not drive the signal too early (which could cause setup time violations at the receiving device) or too late (which could cause hold time violations).

Example Syntax:

set\_output\_delay -clock <clock\_name> -delay <delay\_value> <output\_port\_name>

After defining your design and constraints, use the timing analysis features in your digital design software. These tools analyze critical paths (the longest paths) from inputs to outputs to verify that signals arrive within the specified timing windows.

Examine the timing analysis report for any setup time violations (where the output signal does not arrive early enough) or hold time violations (where the output signal does not stay stable long enough) at the output ports. Identifying these violations can help in diagnosing potential malfunctions.

Q3. When do we use multicycle path exceptions?

Multicycle path exceptions are used when certain data paths in a design are intentionally allowed to take more than one clock cycle to complete their operation. This is often necessary when the logic in these paths performs complex computations that cannot be completed within a single clock cycle. Multicycle paths are also used in cases where the design can tolerate longer delays, such as in slower operational modes or non-critical control signals. By specifying multicycle path exceptions, designers can relax timing constraints, allowing for more efficient resource utilization and potentially lower power consumption. This helps in achieving timing closure for paths that would otherwise be difficult to meet within a single cycle constraint.

Q4. How to constrain and analyze multicycle paths?

* Identify the paths that should take multiple clock cycles.
* Set constraints using set\_multicycle\_path for both setup and hold times in your SDC file.
* Run STA to analyze these paths and ensure they meet the specified timing requirements.
* Review timing reports to verify that all multicycle paths are correctly constrained and free of timing violations.

Q5. What is false path exception?

A false path is a timing path that does not need to be analyzed for timing violations because it does not impact the functional operation of the design. These paths are either not used during the normal operation or their timing is not critical to the functionality of the design. By declaring a path as false, you instruct the timing analysis tool to ignore it when performing timing checks.

Q6. Write three differences between ASIC and FPGA

|  |  |  |
| --- | --- | --- |
|  | ASIC (Application-Specific Integrated Circuit) | FPGA (Field-Programmable Gate Array) |
| Architecture | Custom-designed for specific applications | Reconfigurable, can be programmed for various functions |
| Power Consumption | Low, optimized for power efficiency | Higher due to overhead of programmable logic |
| Per-Unit Cost | Fixed once fabricated | Can be reprogrammed multiple times |