Combinational loops is created when the output of a combinational logic is fed back into the same logic without involvement of any memory element like flipflop.

A picture containing diagram

Description automatically generated

Types :- I have defined combinational loops in two types :

1.equivalent to Latch

2.Not equivalent to Latch

* **Combination loops equivalent to latch :-**

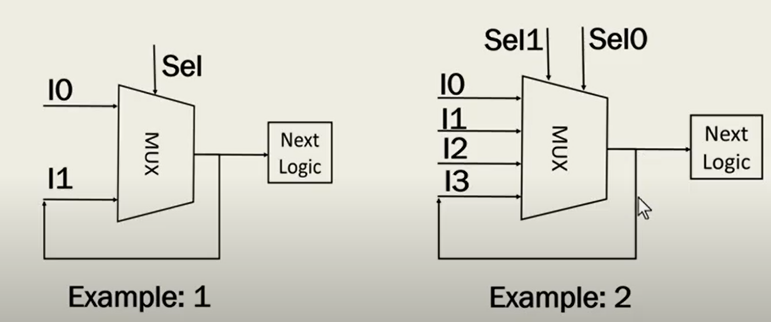


Fig 1 : combinational loops in 2:1 and 4:1 Mux

In example 1 when Sel signal is zero Input Io will pass to output but as soon as Sel change to one I1 will pass to output means output value is lathed.

Similarly in example 2 as soon as Sel1 and Sel0 change to 1 each whatever previous value of output is present it will be latched.

* How these Latches got Inferred in our design.

When we write the HDL code intentionally or unintentionally we miss the Else statement or we simply pass the previous value in Else statement.

Case 1:

When code is written in terms of if else statement

*If (sel==0)*

*Out<=Io;*

When code is written in terms of Case statement

*Case(Sel)   
 1’b0 : Io;*

Case 2:

When code is written in terms of if else

*If (sel==0)*

*Out<=Io;*

Else

Out<=Out;

When code is written in terms of Case

*Case(Sel)   
 1’b0 : Io;*

Default :

Out <=Out

So in above two cases combinational loop will be created which is acting as latch.

Lets see another example :

Suppose we have three Boolean expression(bool\_expr\_1, bool\_expr\_2, bool\_expr\_3) and we have corresponding output (val\_expr\_1, val\_expr\_2, val\_expr\_3)

If we write the verilog code for above conditions and we missed mentioning about the condition when no Boolean expression is matching (Missed Else statement in Verilog code) then system will consider it as latch as shown in Fig 2 :

Verilog code :

*If(bool\_expr\_1)*

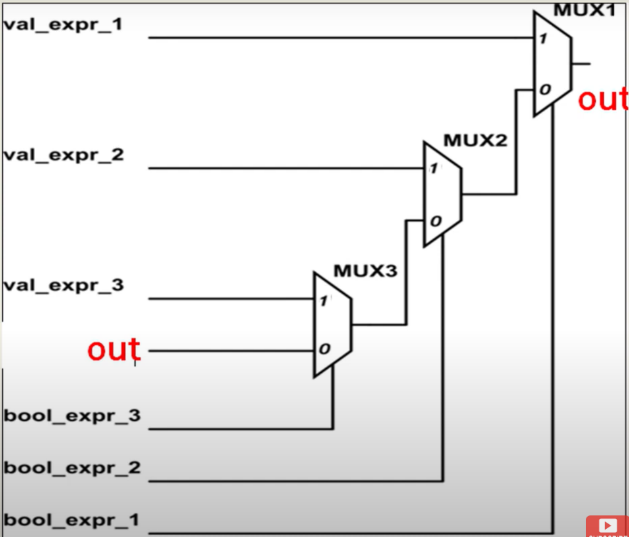
*Out<=Val\_expr\_1;*

*Else If(bool\_expr\_2)*

*Out<=Val\_expr\_2;*

*Else If(bool\_expr\_3)*

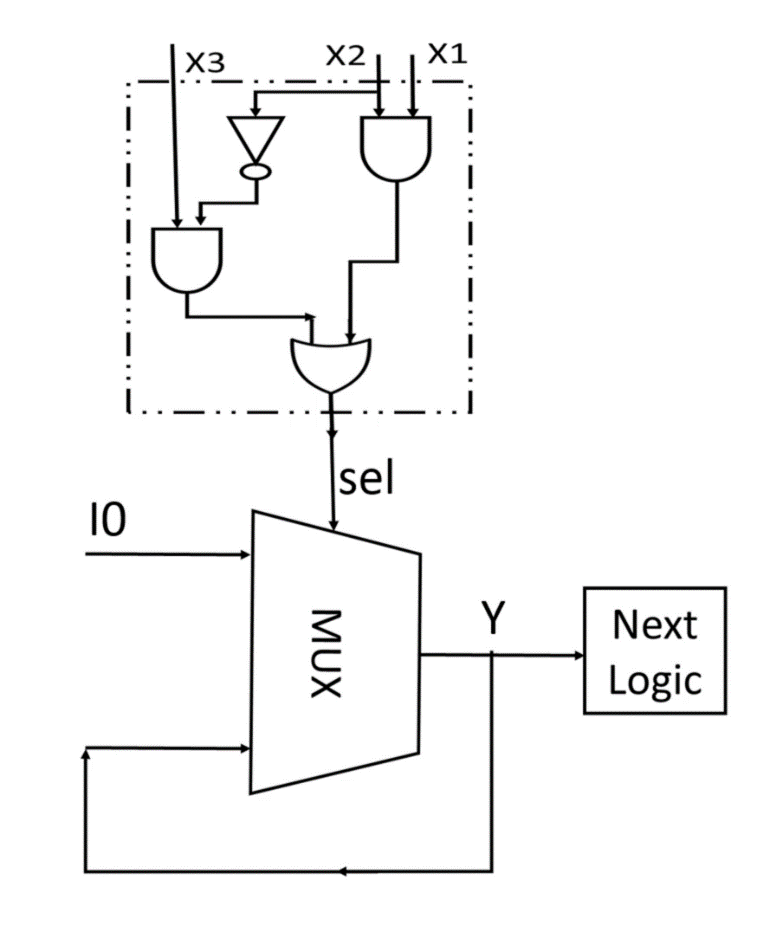
*Out<=Val\_expr\_3;*



*Fig 2 :- 4 :1 Mux in different notation*

* **Why to avoid combinational loops of type latch in digital design :-**

Lets assume a circuit shown in Fig 3 and perform a timing analysis in vivado.



*Fig 3*

Here in above circuit I have assumed delay in all the gate is same and that is 2ns. So there is going to be a glitch happen and select line will take the latest value of Io and that will be latched.

Let’s Write the Verilog code for the same and understand this Step By Step using simulation in vivado.

Text

Description automatically generated

Fig 4: Verilog code for the circuit in Fig 3

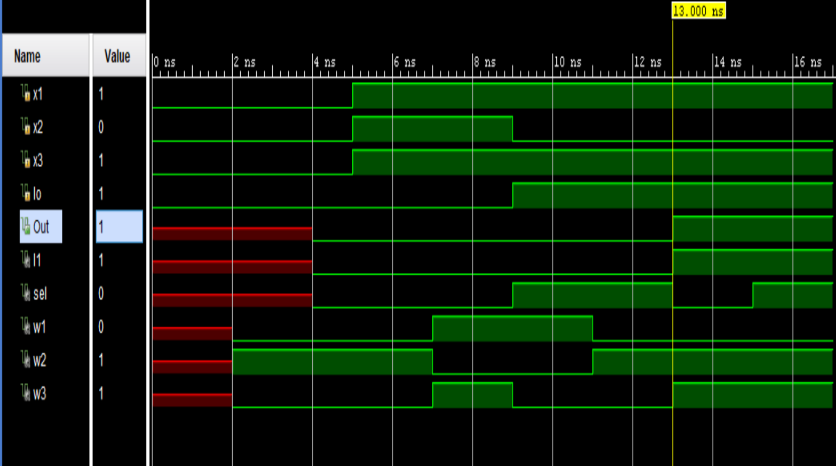


Fig 5 : simulation of the circuit in Fig 3

Step 1: set x3=0,x2=0,x1=0,Io=0 and run the simulation for 5ns just for testing purpose.

Step 2: set x3=1,x2=1,x1=1,Io=0 and run the simulation for another 4ns (till 9ns).

Step 3: change x2=0,Io=1 and run the simulation for another 4ns we get a glitch for 2ns and here in 2ns output had taken the new value of Io because sel value is chaned to 0 for 2ns. And that new value of Io will be latched and Output will keep showing that new value for the subsequent time.

* Another reason why we should avoid latches is that in FPGA predefined cells for latches is not defined and they are synthesized using LUTs..
* Another big problem is power consumption. Latches are transparent to glitches as they are level sensitive, while the flip-flops are edge-sensitive and hence reject any possible glitches.
* **Solution :**

There are three solution that I can think to solve this problem :

1. Either we remove the hazard from the combinational circuit by using overlapping groups in kmap.
2. We must write the **Else** statement in case of if else code or **default** statement statement in case of Case. So that loops won’t occur.
3. We must have to use the Memory device in feedback path that is operated with time period greater than maximum path delay of combinational circuit and synchronised with the inputs that we are giving to combinational circuit.

* **Example to count number of latches in a Verilog case statement :-**

module latch\_count(

input [1:0] in1,in2,in3,

input [2:0] sel,

output reg [1:0] d1,d2,d3 );

always@(\*)

case(sel)

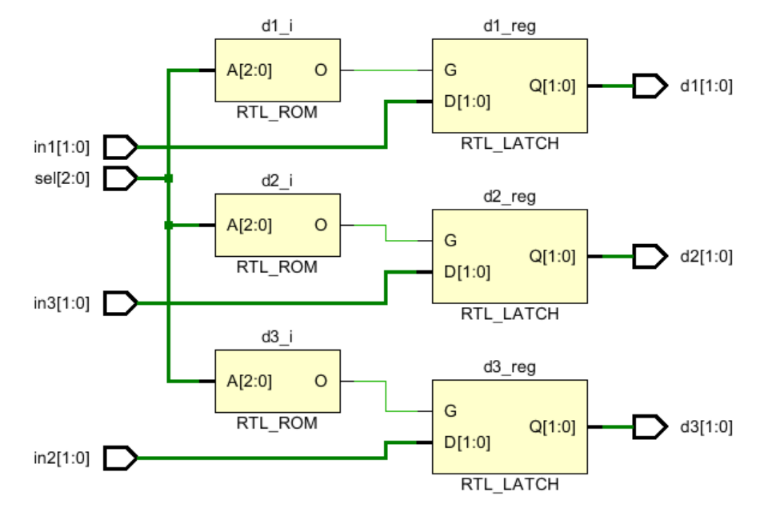
3'b111 : d1 = in1;   //branch-1

3'b010 : d2 = in3;  //branch-2

3'b100 : d3 = in2;  //branch-3

endcase

endmodule



Latch count will always depend on the non-closed output in the case branches irrespective of you have given default branch or not.  Even if you give default here, latch count will be same.  To remove latch, what you need to do is to assign d1,d2 and d3 something in every branch of the case. As an example,  3'b111 :   begin   d1 = in1 ; d2 = 2'b00 ;   d3 = 2'b00; end Also, as d1,d2 and d3 are 2 bit reg, they create two latches each. LDCE is a latch in the figure.

So, total 6 latches are here.

* **Another example**

input [1:0] in1,in2,in3,

input [2:0] sel,

output reg [1:0] d1,d2,d3);

always@(\*)

case(sel)

3'b001:

begin

d1 = in1;

d2 = in2;

d3= in3;

end

3'b010:

begin

d1 = in2;

d2 = in3;

d3= in1;

end

3'b111:

begin

d1 = in3;

d2 = 2'b00;

d3 = 2'b01;

end

endcase

Diagram, schematic

Description automatically generated

In this case, CASE is there with only 3 of its branches (001,010,111) from available 8 branches (000,001,010......111).

And there is no default or there is no default-assignment of d1,d2 and d3 before the case starts.

And all branches are closed, i.e. d1,d2 and d3 are all assigned in all the 3 branches.

Latches will come into picture when sel is not in (001,010,111).

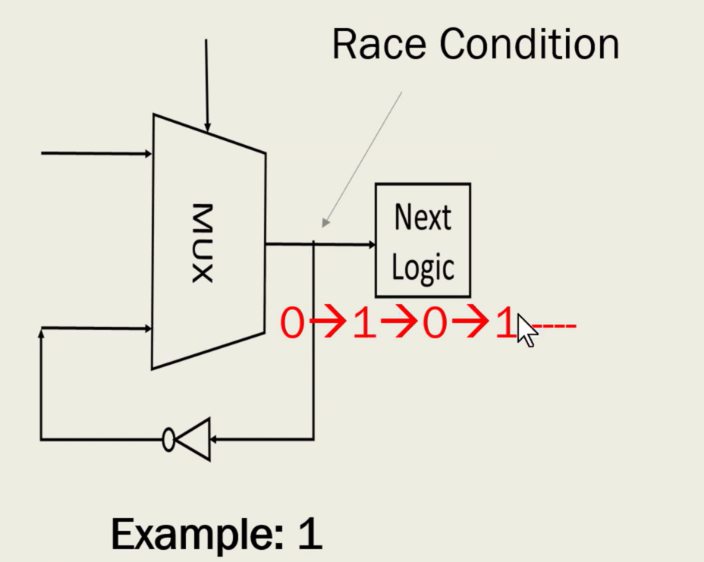
but latches will come into picture only for the LHS reg (d1,d2,d3) in the case statement.

ONE THING FOR SURE IS NUMBER OF LATCHES WILL DEPEND ON d1,d2,d3 IN ANY CASE (not on number of branches or anything else).

Now, d1,d2 and d3 all are 2 bit wide. so, Total 6 latches will be there.

* **Combinational loop not equivalent to latch :-**

Let’s assume the same case that we did in Fig 1 but this time we will connect a inverter in feedback path.



This circuit will continuously keep toggling its output that is called Race around condition.

Verilog code for such circuit is as follows :-

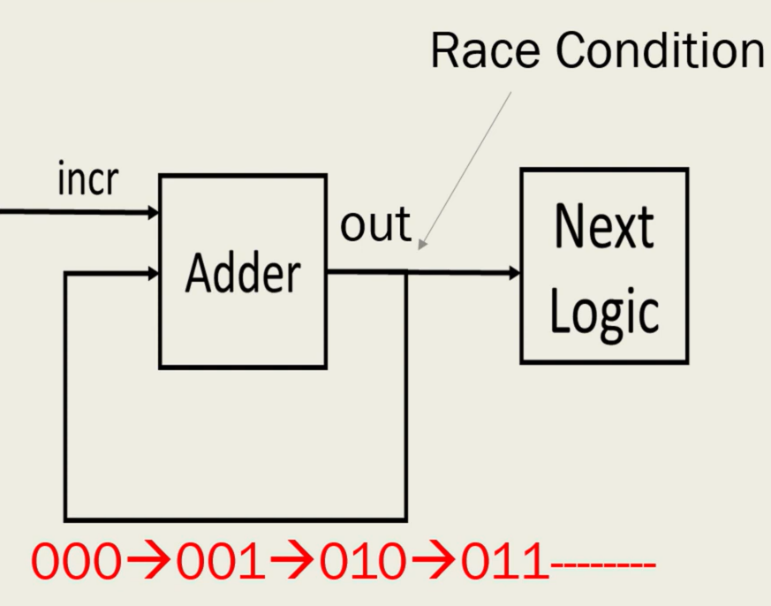
*If(sel==1’b0)*

*Out= Io ;*

*Else*

*Out = ~Out ;*

Another example for such circuits is as follows :-



This circuit will continuously keep adding output keeps on changing output.

Verilog code for such circuit is as follows :-

Assign out < =out+incr ;

One common Example of such type of circuit is t flipflop that’s why we use

**Solution :-**

We should avoid the passing of output signal directly to input again without involvement of sequential circuits like flipflops.