30 Days of RTL Coding

-By T Tushar Shenoy

Day 1

Problem Statement: Implementing Logic Gates using different levels of abstraction.

Theory: Verilog supports 4 levels of abstraction**,**

Data flow level: At this level, the module is designed by specifying the data flow, like how the data is flowing in the circuit based on that the equation should be written. Signals are assigned by the data manipulating equations, design is implemented using continuous assignment i.e. assign statement is used in this level, the assignments present in it are concurrent in nature. Whenever RTL (Register transfer logic) is heard then the data flow level comes into action.

Behavioural level: This is the highest level of abstraction provided by HDL, this level describes the behaviour of the system. Different elements like functions, tasks, and blocks can be used, two important constructors are initial and always.

Gate level or Structural level: Module is implemented in terms of gates, which is the lowest level of abstraction. Basic logic gates are available as predefined primitives. In the digital library of Verilog, these logic gates are already saved and can be used directly like and, or, xor, nand, nor, not.

Switch Level: Module is implemented in terms of switches, we can represent the entire circuit as a CMOS circuit.

In this project we are going to implement Logic gates using Data flow behavioural and Gate level Implementation.

Verilog Code:

Dataflow

```
module logic_gates_dataflow(a,b,yor,yand,ynor,ynand,yxor,yxnor);

input a,b;
output yor,yand,ynor,ynand,yxor,yxnor;

assign yor=a|b;
assign yand=a&b;
assign ynor=~(a|b);
assign ynand=~(a&b);
assign yxor=a^b;
assign yxnor=~(a^b);

endmodule
```

Behavioural

```
module logic_gates_behavioural(a,b,yor,yand,ynor,ynand,yxor
,yxnor);

input a,b;
output reg yor,yand,ynor,ynand,yxor,yxnor;

always@(a or b)
begin
```

```
 yor=a|b;
 yand=a&b;
 ynor=~(a|b);
 ynand=~(a&b);
 yxor=a^b;
 yxnor=~(a^b);
end
endmodule
```

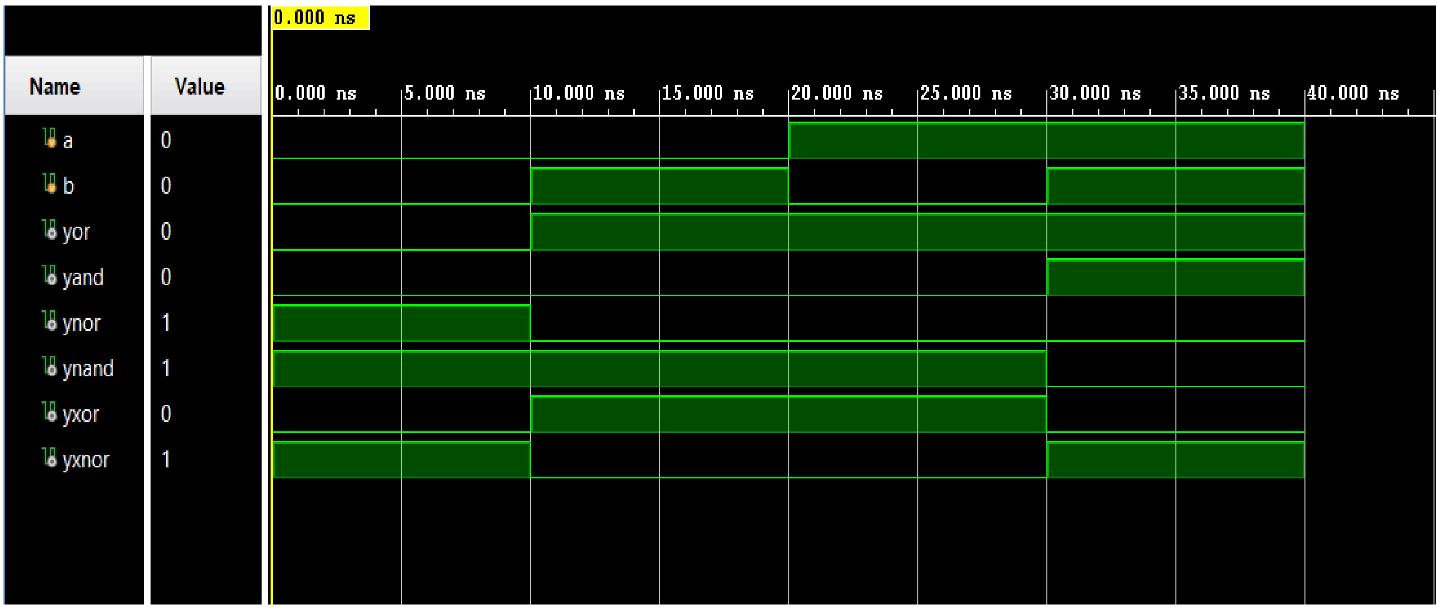
Gate or Structural

```
module logic_gates_gate(a,b,yor,yand,ynor,ynand,yxor,yxnor);

input a,b;
output  yor,yand,ynor,ynand,yxor,yxnor;

 or(yor,a,b);
 and(yand,a,b);
 nor(ynor,a,b);
 nand(ynand,a,b);
 xor(yxor,a,b);
 xnor(yxnor,a,b);

endmodule
```

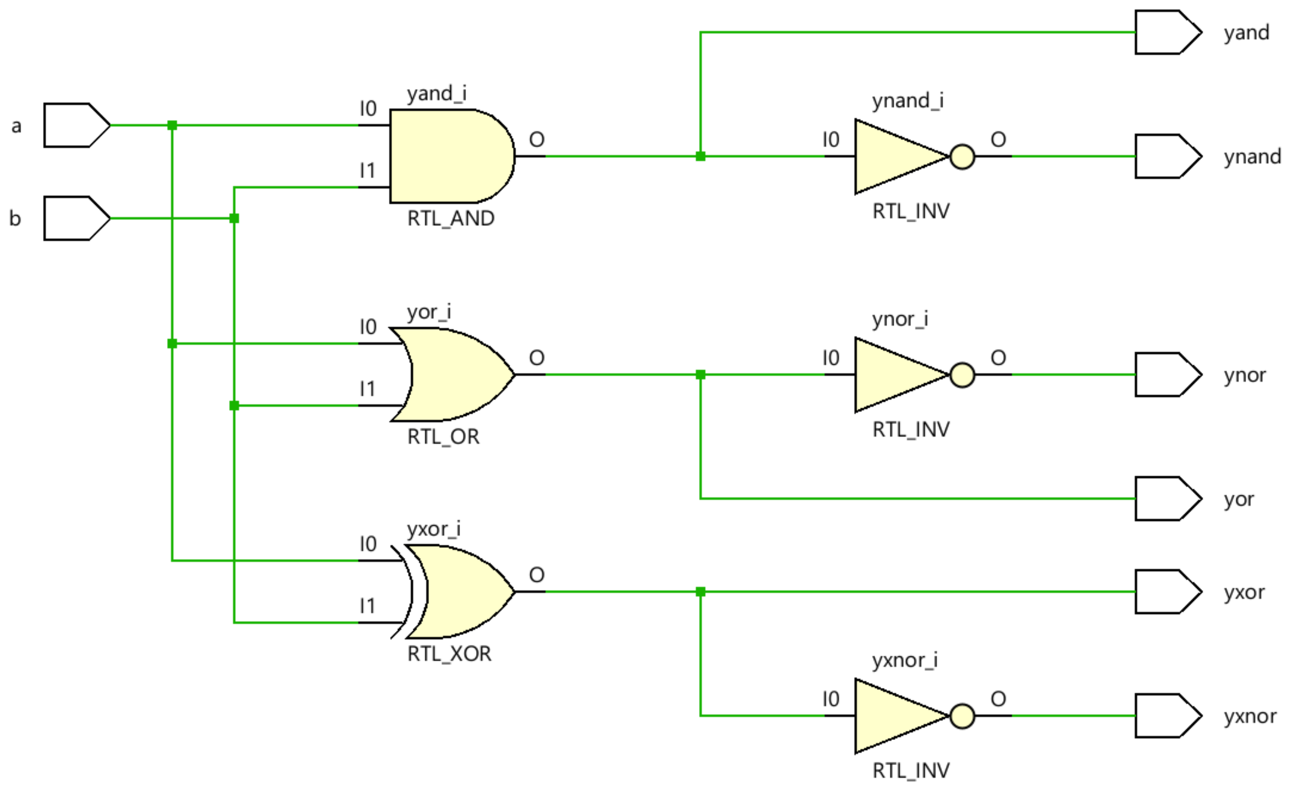
Testbench Code:

```
module logic_gates_tb;


reg a,b;

wire yor,yand,ynor,ynand,yxor,yxnor;


//change name to module name of required style

logic_gates_dataflow dut(.a(a),.b(b),.yor(yor), .yand(yand),
.ynor(ynor),.ynand(ynand), .yxor(yxor),.yxnor(yxnor));


initial begin

//stimulus

    a=0; b=0;

#10 a=0; b=1;

#10 a=1; b=0;

#10 a=1; b=1;

#10 $finish;

end


endmodule
```

Simulation Output:



Schematic:



GitHub Repository Url :-
https://github.com/tusharshenoy/RTL-Day-1-Logic-Gates