VLSI LAB Experiment-5

Harshavardhan Alimi

18EC10021

Part A:-

Aim :-

The aim of this experiment is to design a Clock with Time Period=40ns and a duty cycle of 25% by using the always and initial Statements. The Value of clock at time = 0 ,should be initialized to 0.

Description :-

- The model is implemented using only always and initial statements.
- Duty cycle of 25% indicates that the clock is on for 1/4th of the time period.
- Time period is made in ns by using (`timescale 10ns / 1ns) :- this defines 10ns for 1 timeunit i.e., 4 time units is a time period here.
- The clock is initialized with 0 and it is made to 1 after 30 ns(3 time units)(i.e., 0 is present only for 30ns).
- The clock is again modified from 1 to 0 after 10 ns(1 time unit)(i.e., 1 is present only for 10ns).
- This will make the duty cycle of 25% for the clock.

Verilog codes :-

```
`timescale 10 ns/1 ns


module CLK();


   reg clock;
```

```
    initial
    clock = 1'b0;

    always
    begin
        if(clock)
            #1 clock = ~clock;
        else
            #3 clock = ~clock;
    end

    initial
    #30 $finish;

    initial
    $monitor("time = %d*10 ns, clock = %b",$time ,clock);

    initial begin
        $dumpfile("test.vcd");
        $dumpvars;
    end

endmodule
```

Runtime log :-

VCD info: dumpfile test.vcd opened for output.

time = 0*10 ns, clock = 0

time = 3*10 ns, clock = 1

time = 4*10 ns, clock = 0

time = 7*10 ns, clock = 1

time = 8*10 ns, clock = 0

time = 11*10 ns, clock = 1

time = 12*10 ns, clock = 0

time = 15*10 ns, clock = 1

time = 16*10 ns, clock = 0

time = 19*10 ns, clock = 1

time = 20*10 ns, clock = 0

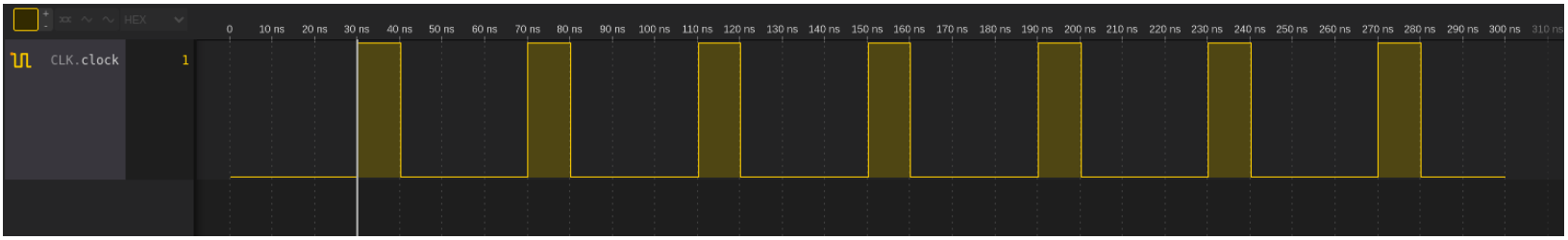
time = 23*10 ns, clock = 1

time = 24*10 ns, clock = 0

time = 27*10 ns, clock = 1

time = 28*10 ns, clock = 0

Results :-



Discussion/Conclusion :-

- The clock with duty cycle of 25% and time period of 40ns is made using only initial and always statements only.
- The output is plotted and displayed using GTKWAVE.

Drive links for verilog codes :-

Part-A

Part B:-

Aim :-

The aim of this experiment is to design a 1-to-4 DeMUX(from 1-to-2 DeMUX) using switch level Verilog description.

Schematic diagrams and labels :-

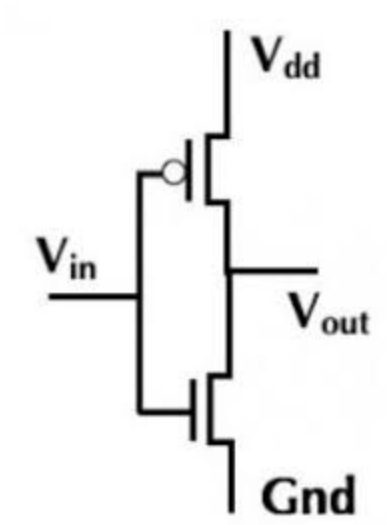


Fig.1: cmos inverter implementation using PMOS and NMOS

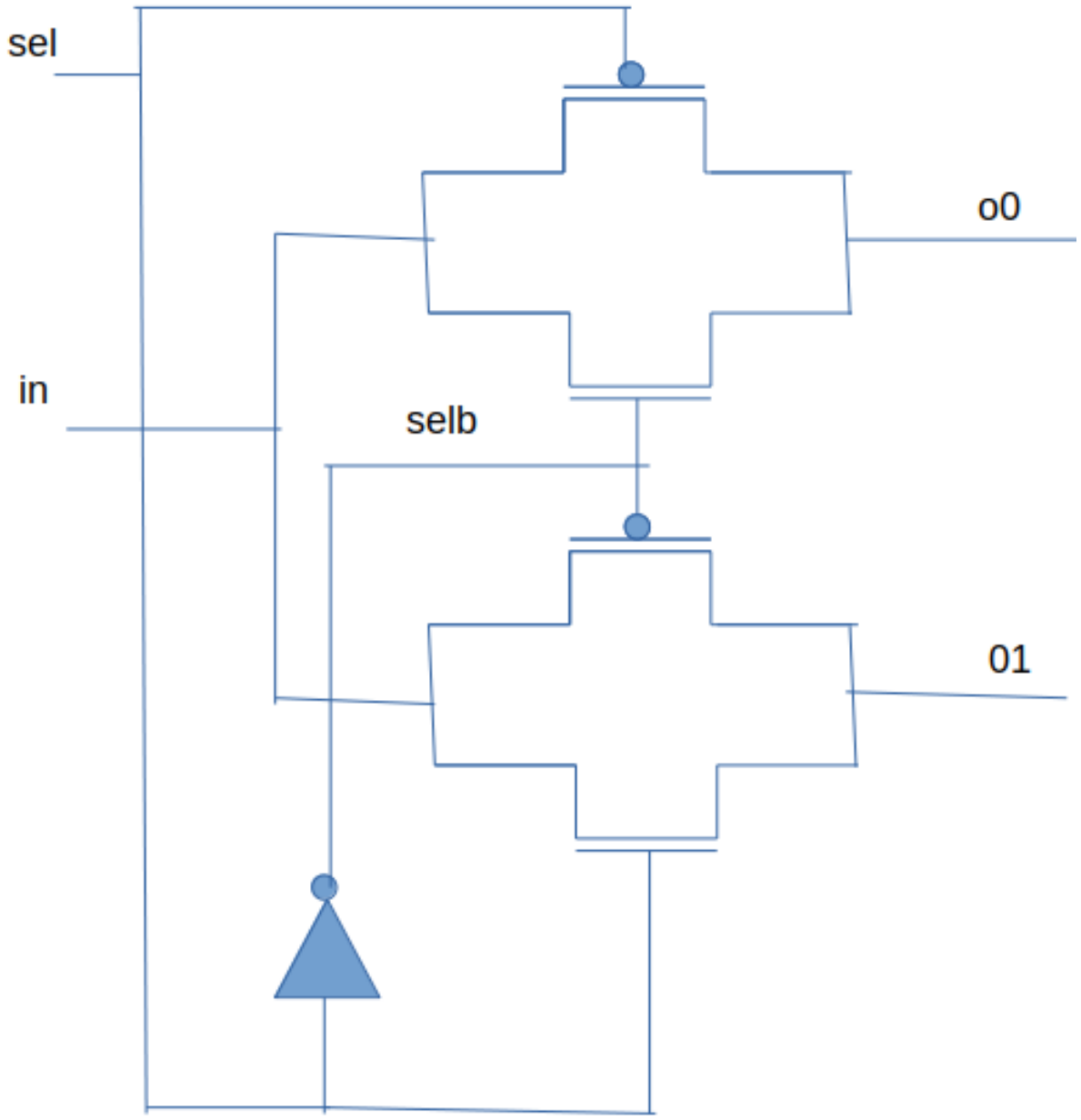


Fig.2:Implementation of 1x2 Demultiplexer using cmos(transmission logic gates) and a inverter

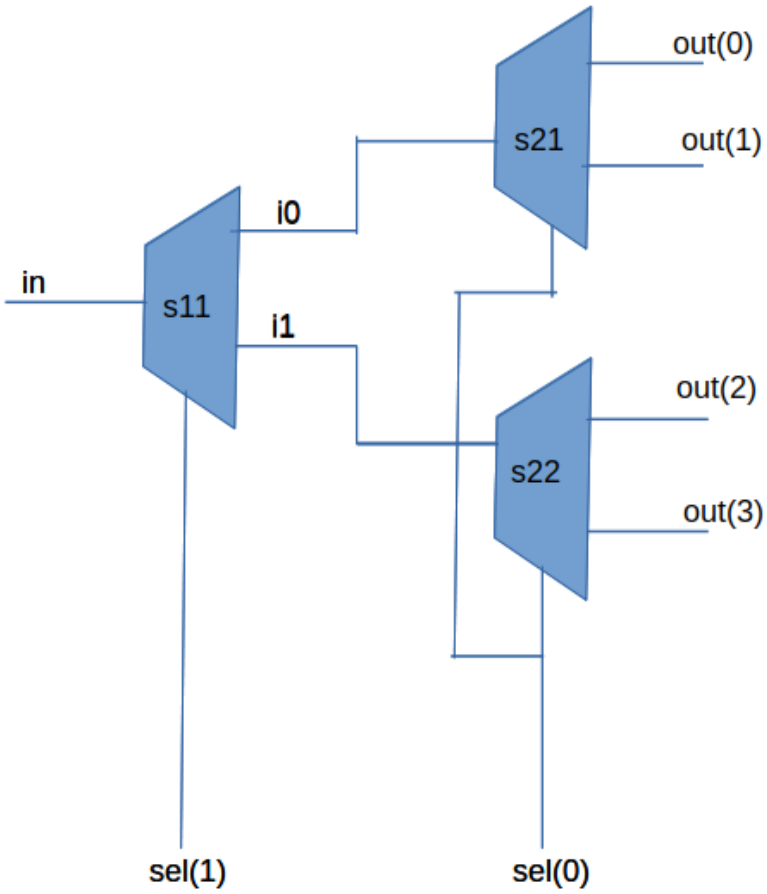


Fig.3:Implementation of 1x4 Demux using 1x2 Demux

Description :-

- The model is implemented using the transmission logic gates..
- A demultiplexer is a device that takes a single input line and routes it to one of several digital output lines.
- A demultiplexer of 2^n outputs has n select lines, which are used to select which output line to send the input. A demultiplexer is also called a data distributor.
- CMOS switches(transmission logic gates) are used to implement the 1x2 demultiplexer.
- a 1x4 demultiplexer is implemented using three 1x2 demultiplexers(which are implemented in switch level).
- The input data{in} will be read at one of the terminals {out(0) , out(1) , out(2) , out(3) } depending on the select lines.

Verilog codes :-

```
module cmos_inverter (o,i);
   input i;
   output o;

   supply1 vdd;
   supply0 gnd;

   pmos(o,vdd,i);
   nmos(o,gnd,i);

endmodule

module Mux_1x2 (o0,o1,sel,in);
   input sel,in;
   output o0,o1;

   wire selb;

   cmos_inverter nt(selb,sel);
   //Transmission logic gates
   cmos(o0,in,selb,sel);
   cmos(o1,in,sel,selb);

endmodule

module Mux_1x4(out,s,in);
```

```
    input in;
    input [1:0]s;
    output [3:0]out;

    wire i0,i1;
    // First Layer of the Demux
    Mux_1x2 S11(i0,i1,s[1],in);
    // Second Layer of the Demux
    Mux_1x2 S21(out[0],out[1],s[0],i0);
    Mux_1x2 S22(out[2],out[3],s[0],i1);

endmodule

module test_bench();
    reg in;
    reg [1:0] sel;
    wire [3:0]out;

    Mux_1x4 uut(out,sel,in);

    initial
    begin
        $monitor("time=%5d, input=%b, select=%b, output=%b",$time,in,sel,out);
        //initialisation of select lines
        #0 in = 1'b1;sel = 2'b00;
        #5 sel = 2'b01;
        #5 sel = 2'b10;
        #5 sel = 2'b11;
```

```
        #5 in = 1'b0;sel = 2'b00;
        #5 sel = 2'b01;
        #5 sel = 2'b10;
        #5 sel = 2'b11;
        #5 $finish;
    end

    initial begin
        $dumpfile("test.vcd");
        $dumpvars;
    end

endmodule
```

Runtime log :-

VCD info: dumpfile test.vcd opened for output.

time= 0, input=1, select=00, output=zzz1

time= 5, input=1, select=01, output=zz1z

time= 10, input=1, select=10, output=z1zz

time= 15, input=1, select=11, output=1zzz

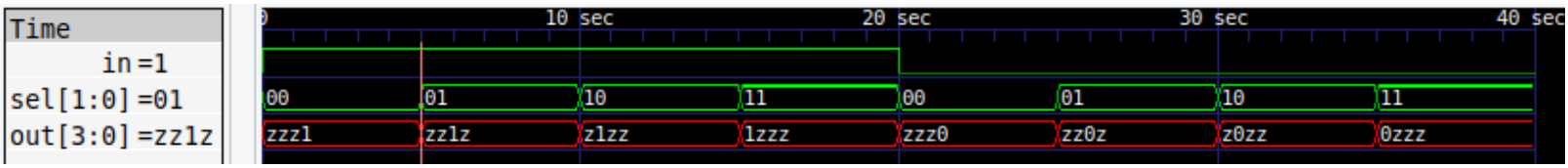
time= 20, input=0, select=00, output=zzz0

time= 25, input=0, select=01, output=zz0z

time= 30, input=0, select=10, output=z0zz

time= 35, input=0, select=11, output=0zzz

Results :-



Discussion/Conclusion :-

- The output is plotted and displayed using GTKWAVE.
- When input is **1**,then output line depending on the sel lines will get **1**,the other output lines will get **z(floating bits i.e.,undefined state)** as output and same as the case when input is **0**.

Drive links for verilog codes :-

Part-B