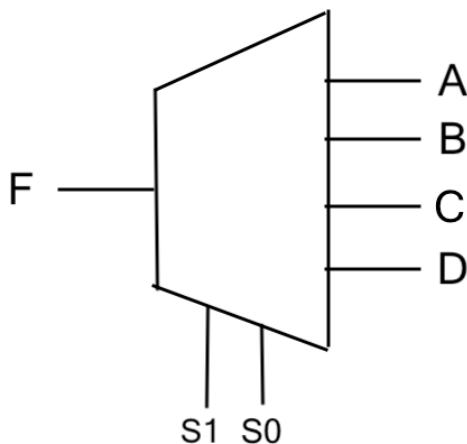


Lab 5: Building Demultiplexer with Verilog

Today's lab is about building a simple combinational logic component: A 1 -> 4 Demultiplexer using the 'always' construct on Verilog and simulating a testbench to verify the operation.

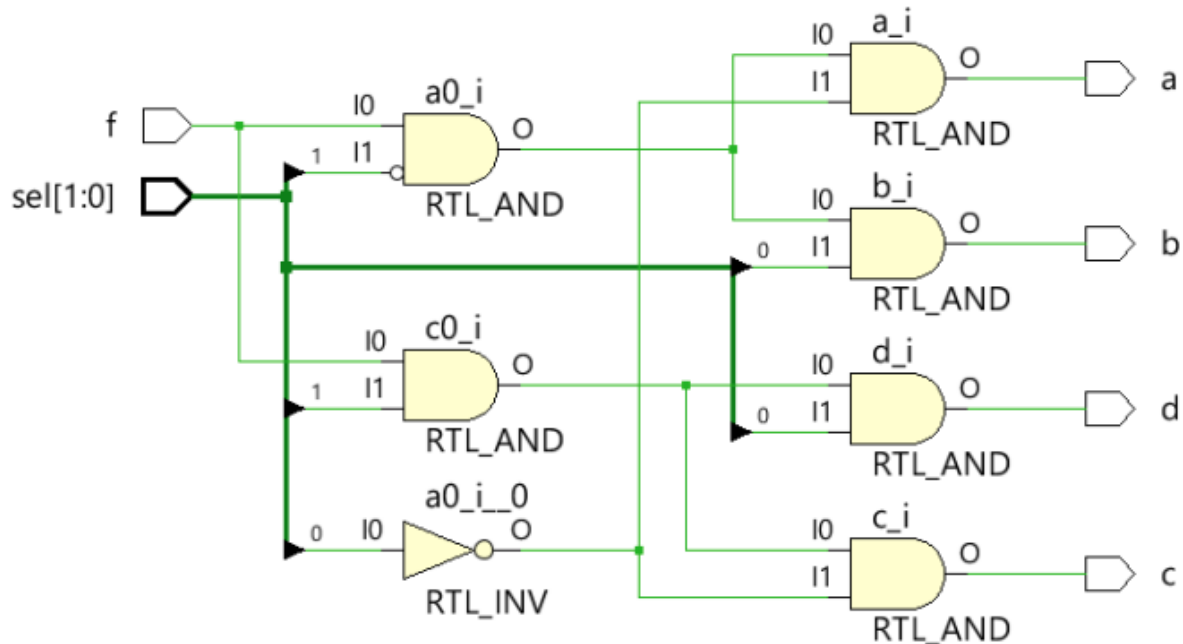
Demultiplexer is a circuit that has one input routed to one of its multiple output and the output that is active is determined by a select input. A demultiplexer has n select lines that chooses to route the input to one of its 2^n outputs. So if it has 2 select lines then the output will be $2^2 = 4$.

1×4 demux



S1	S0	D	C	B	A
0	0	0	0	0	F
0	1	0	0	F	0
1	0	0	F	0	0
1	1	F	0	0	0

The way a demultiplexer works is - if S1 and S2 are both 0 =, A gets a value of F. If S1 is 0 and S0 is 1, then B gets a value of F. Similarly, S1 and S0 for 1, 0 and 1, 1, output C and D gets the value of F. So in this lab we will basically implement this functionality of 1 -> 4 demultiplexer using Verilog code. The 1 -> 4 demultiplexer circuit in real life looks like the following -



We will be writing a Verilog module for demux that has Control bits sel[1:0] (S1 and S0) and the line f as inputs, and a,b,c,d as the 4 outputs. We will be using always block to implement the functionality described above for demultiplexer. To make the output values equal to F corresponding to the control bits 0 0, 0 1, 1 0, & 1 1, we can write Continuous assignment statements as -

$$A = \sim S1 \ \& \ \sim S0 \ \& \ F$$

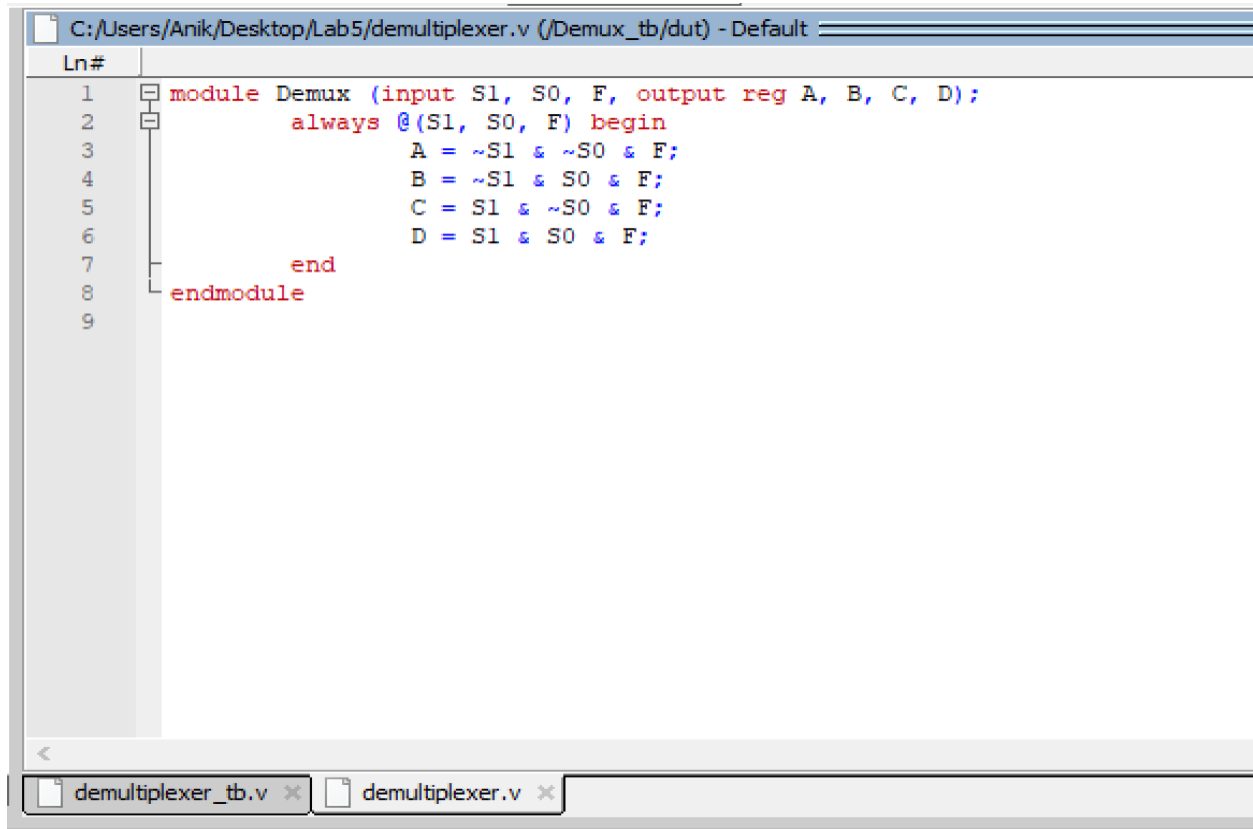
$$B = \sim S1 \ \& \ S0 \ \& \ F$$

$$C = S1 \ \& \ \sim S0 \ \& \ F$$

$$D = S1 \ \& \ S0 \ \& \ F$$

These conditional assignments will go inside the 'always' block. This implementation gives the functionality to choose A for F when selective input is 0 0, B to F for 0 1, C to F for 1 0 and D to F for 1 1. Next we will write a testbench file to simulate the demultiplexer and get the waveform to verify the demux functionality works.

Verilog Module for Demultiplexer -

A screenshot of a Verilog code editor window. The title bar shows the file path: C:/Users/Anik/Desktop/Lab5/demultiplexer.v (/Demux_tb/dut) - Default. The editor displays a Verilog module named 'Demux' with inputs S1, S0, F and outputs A, B, C, D. The code is as follows:

```
Ln# |  
1  | module Demux (input S1, S0, F, output reg A, B, C, D);  
2  |     always @(S1, S0, F) begin  
3  |         A = ~S1 & ~S0 & F;  
4  |         B = ~S1 & S0 & F;  
5  |         C = S1 & ~S0 & F;  
6  |         D = S1 & S0 & F;  
7  |     end  
8  | endmodule  
9  |
```

The bottom of the window shows two tabs: 'demultiplexer_tb.v' and 'demultiplexer.v'.

Testbench -

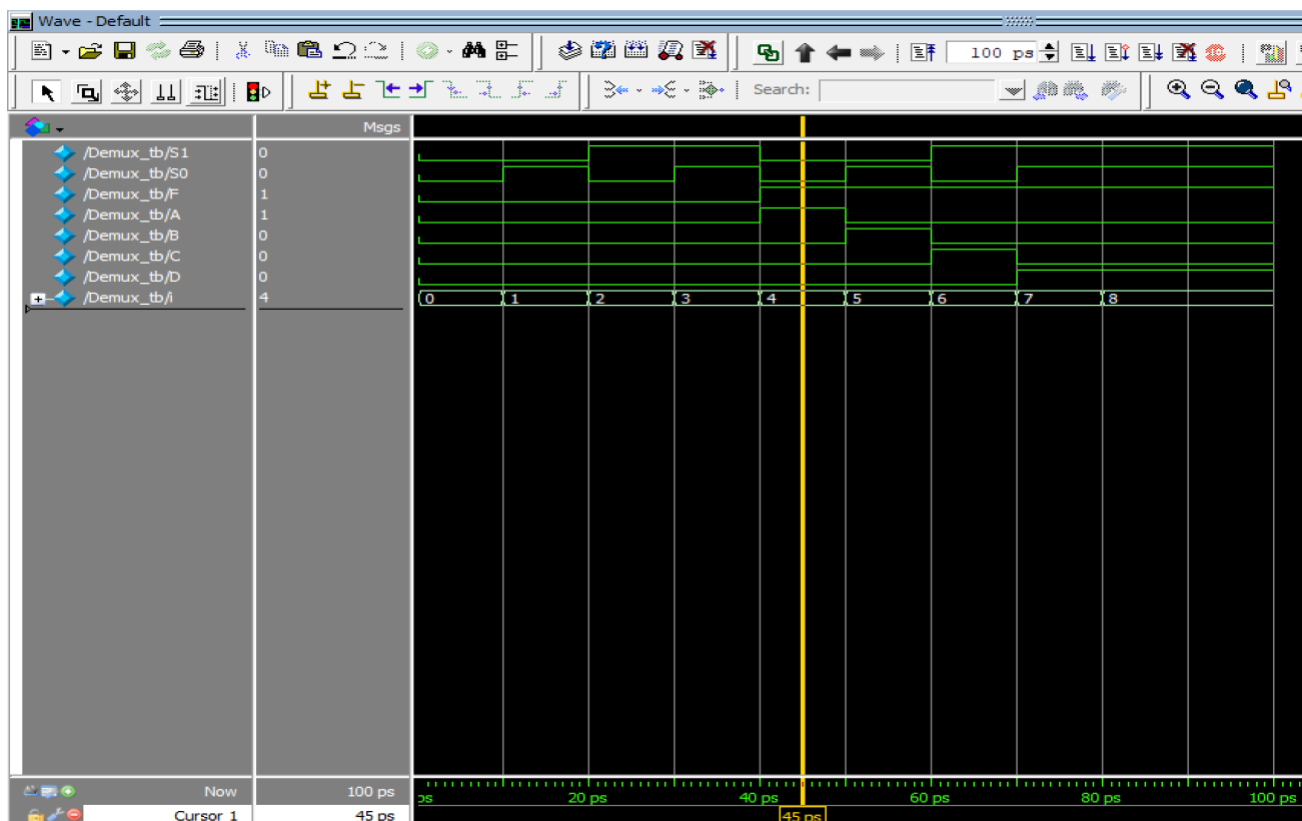
We will use a testbench to simulate our demux module to make sure it is working correctly and we can verify it using the waveform. In our test bench files, we don't have any additional input or output. But we create a temporary reg for inputs and temporary wire for outputs that will be connected to source file input and output. We force value the input and we get output based on the source file. Also we will have a integer i because we will be using a for loop in our testbench. To test a module, we write the module name (demux) followed by 'dut' and inside bracket we map all the input and outputs with the demux module. Then we start with 'initial begin' that follows by the 'end' keyword, and inside we put the testing functionality. First we make sure S1,

S0, F are all 0's. Then using a for loop, we implement 8 possible values ($2^3 = 8$) for 3 inputs S1, S0 and F. We also include the time delay (#10) inside the for loop for each force value input.

```
C:\Users\Anik\Desktop\Lab5\demultiplexer_tb.v (/Demux_tb) - Default
Ln#
1  module Demux_tb();
2
3      reg S1, S0, F;
4      wire A, B, C, D;
5      integer i;
6
7      Demux dut (.S1(S1), .S0(S0), .F(F), .A(A), .B(B), .C(C), .D(D));
8
9      initial begin
10         S1 <= 0;
11         S0 <= 0;
12         F <= 0;
13
14         $monitor ("S1=%0b S0=%0b F=%0b D=%0b C=%0b B=%0b A=%0b", S1, S0, F, A, B, C, D);
15
16         for (i = 0; i < 8; i = i + 1) begin
17             {F, S1, S0} = i;
18             #10;
19         end
20     end
21 endmodule
22
```

Now if we simulate it we will get the waveforms.

Waveform explanations -



We can verify our demultiplexer using the waveform. Let's pick for $S_1 = 0$ and $S_2 = 0$ and $F = 1$.

We know when S_1 & S_2 are both 0, we map F to the output A . So A will be 1. We can see it in the waveform that for $S_1 = 0$, $S_2 = 0$ and $F = 1$, we get $A = 1$. Similarly for $S_1 = 0$ and $S_2 = 1$ and $F = 1$, we map the F to output B and we can see B has value B is 1. Next for $S_1 = 1$ and $S_2 = 0$ and $F = 1$, we see C goes to 1 and for $S_1 = 1$ and $S_2 = 1$ and $F = 1$, D goes to 1. So our waveforms shows that 1 -> 4 demultiplexer functionality is correct.