LAB 5:

Scalable Multiplexer

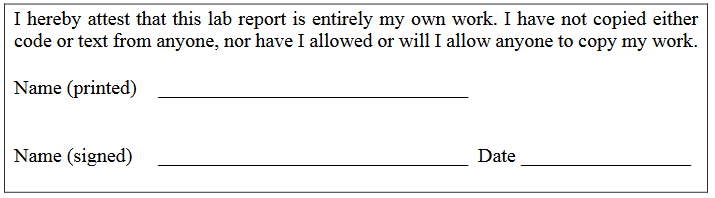
Professor Ronald Mehler

ECE 526L

Spring 2018

Garen Nikoyan

3/1/2018



**Objective:**

The purpose of this lab is to become familiar with utilizing parameters and how to use them to change a value at simulation. Three different methods will be used, including module instance parameter value assignment, defparam or in-line explicit redefinition, and lastly, named parameter passing. Aside from the use the parameters, the use of behavioral code will continue.

**Methodology:**

For this lab, the first step was to create the module for the scalable mux. It was made scalable by using parameters. A variable, Size, was created using parameter and given a value of 1. When deciding the size of the inputs and output, the parameters is called, and 1 is subtracted from it. This is so Size will be equal to the number of bits wanted. Since this uses combinational logic, always @\* is used, as it includes any signal that is read inside of the block. Inside of the always block, a ternary operator was used. This makes it much easier to implement multiplexer logic, using only one line and matching the requirements of the lab manual for the Select = x cases. The test bench instantiates the module four times, using a different parameter setting each time, this mean there are 4 outputs, 8 bit, 5 bit, 2 bit, and 1 bit. The inputs for the smaller bit-count modules were instantiated from the least significant bits of the 8 bit input, using an always block. The test bench uses 6 test vectors, which adequately test the multiplexer module. The 6 cases are:

1) SEL=1, A=!B

2) SEL=0, A=!B

3) SEL=x, A=B for first 4 bits, A!=B for last 4

4) SEL=x, A=!B

5) SEL=x, A=B=1

6) SEL=x, A=B=0

**Analysis:**

Looking at waveforms and the log, it can be seen that the use of parameters in the instantiated modules worked correctly. All test cases change every bit of the output. This is done to ensure that everything is working properly, and the correct output is not just a result of a bit carrying over from the previous test case. The inputs are created and set as an instantiation of the initial 8 bit input, since that is the largest one.

When SEL is set to 0, the output takes on the value of input A. When SEL is set to 1, the output takes on the value of input B. When SEL is set to x, it gets interesting in terms of the output. The output is equal to the input, for the bits where the two inputs are equal. The output is x when the inputs are not equal.

In Figure 1, A is set to 01010101 and B is set to 10101010. SEL is set to 1, and the outputs take on the value of B, as expected. When SEL is set to 0, the outputs take on the value of A.

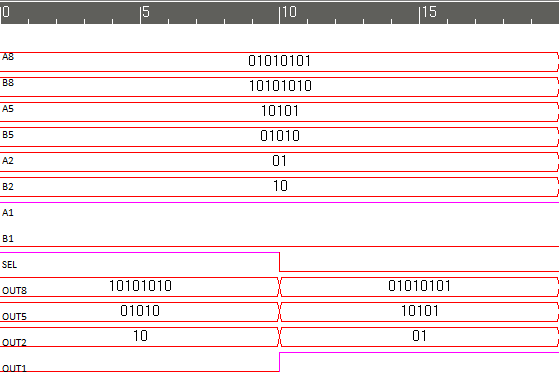


Figure 1. Testing SEL=1 and SEL=0. As expected, OUT=B when SEL=1, and OUT=A when SEL=0

In Figure 2, SEL is set to x. A and B are set equal for the first 4 bits, and inverse for the last 4. This gives the expected output of matching bits outputting that value, and non-matching bits outputting x. A is then set to 1, and B is set to 0. Since they are all different bits, the output is all x.

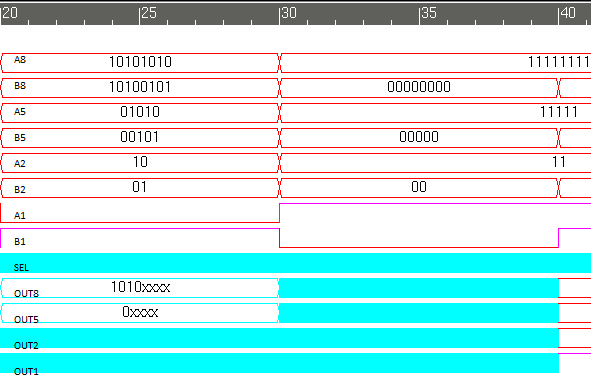


Figure 2. Testing SEL=x, with A[7:4]=B[7:4], A[3:0]!=B[3:0], and A!=B

In Figure 3, SEL is set to x, and A and B are set to 1. The outputs take on a value of 1 since A=B. This is tested again with A and B set to 0, where the outputs change to 0.

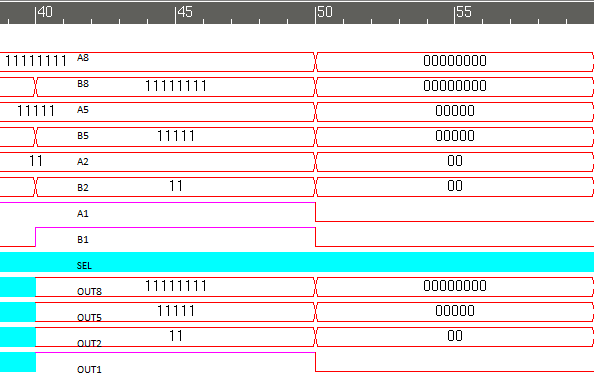


Figure 3. Testing SEL=x with A=B=1, and A=B=0

**Modules:**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\* ECE526L Experiment #5 Garen Nikoyan, Spring 2018 \*\*\*

\*\*\* Scalable Multiplexer \*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\* Filename: scale\_mux.v Created by: Garen Nikoyan, 3/1/2018 \*\*\*

\*\*\* -Revision History \*\*\*

\*\*\* 3/1/2018: First draft \*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\* This module models a scalable multiplexer \*\*\*

\*\*\* The amount of the bits of the 2 inputs and output can be passed in

\*\*\* through the parameter "Size"

\*\*\* if SEL=1, OUT=B, if SEL=0, OUT=A

\*\*\* if SEL=x, and A=B, OUT=A, if SEL=x, and A!=B, OUT=x

\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

`timescale 1ns/1ns

module scale\_mux(A, B, SEL, OUT);

parameter Size = 1;

output reg [Size-1:0] OUT;

input [Size-1:0] A, B;

input SEL;

always @\*

OUT = SEL ? B : A; // if SEL=1, OUT=B, if SEL=0, OUT=A

// if SEL=x, then A=B causes OUT=A=B, and A!=B, OUT=x

endmodule

**Testbench:**

`timescale 1ns/1ns

module scale\_tb();

//using parameter SIZES of 1,2,5,8

reg [7:0]A8,B8;

reg [4:0]A5,B5;

reg [1:0]A2,B2;

reg A1,B1;

reg SEL;

wire [7:0] OUT8;

wire [4:0] OUT5;

wire [1:0] OUT2;

wire OUT1;

// Using module instance parameter value

scale\_mux #(8) MUX8(A8,B8,SEL,OUT8);

// Using named parameter passing

scale\_mux #(.Size(5)) MUX5(A5,B5,SEL,OUT5);

// Using defparam

defparam MUX2.Size=2;

scale\_mux MUX2(A2,B2,SEL,OUT2);

// Instantiating MUX with regular Size of 1

scale\_mux MUX1(A1,B1,SEL,OUT1);

initial begin

$vcdpluson;

$monitor("%d ns, OUT8=%b, OUT5=%b, OUT2=%b, OUT1=%b \n\t\t A8=%b ,B8=%b, SEL=%b",$time,OUT8,OUT5,OUT2,OUT1,A8,B8,SEL);

end

always @\* begin // making 5,2, and 1

A5=A8[4:0];

B5=B8[4:0];

A2=A8[1:0];

B2=B8[1:0];

A1=A8[0];

B1=B8[0];

end

initial begin

// SEL=1, so OUT=B

$display("Test 1: SEL=1, A=!B, OUT=B"); A8=8'b01010101; B8=8'b10101010; SEL=1'b1;

// SEL=0, so OUT=A, A and B stay the same

#10 $display("Test 2: SEL=0, OUT=A");SEL=1'b0;

// SEL=x, so if A=B, OUT=A=B, if A!=B, OUT=x

// SEL=x, some common and some different bits between A and B

#10 $display("Test 3: SEL=x, A[7:4]=B[7:4], OUT[7:4]=A[7:4], OUT[3:0]=x"); A8=8'b10101010; B8=8'b10100101; SEL=1'bx;

// SEL=x, A=!B OUT=x

#10 $display("Test 4: SEL=x, A!=B, OUT=x"); A8=8'b11111111; B8=8'b00000000; SEL=1'bx;

// SEL=x, A=B, OUT=1

#10 $display("Test 5: SEL=x, A=B=1, OUT=A=0"); A8=8'b11111111; B8=8'b11111111; SEL=1'bx;

// SEL=x, A=B=0, OUT=0

#10 $display("Test 6: SEL=x, A=B=0, OUT=A=0"); A8=8'b00000000; B8=8'b00000000; SEL=1'bx;

#10 $finish;

end

endmodule

**Log:**

Chronologic VCS simulator copyright 1991-2017

Contains Synopsys proprietary information.

Compiler version M-2017.03-SP1\_Full64; Runtime version M-2017.03-SP1\_Full64; Mar 1 19:27 2018

VCD+ Writer M-2017.03-SP1\_Full64 Copyright (c) 1991-2017 by Synopsys Inc.

Test 1: SEL=1, A=!B, OUT=B

0 ns, OUT8=10101010, OUT5=01010, OUT2=10, OUT1=0

A8=01010101 ,B8=10101010, SEL=1

Test 2: SEL=0, OUT=A

10 ns, OUT8=01010101, OUT5=10101, OUT2=01, OUT1=1

A8=01010101 ,B8=10101010, SEL=0

Test 3: SEL=x, A[7:4]=B[7:4], OUT[7:4]=A[7:4], OUT[3:0]=x

20 ns, OUT8=1010xxxx, OUT5=0xxxx, OUT2=xx, OUT1=x

A8=10101010 ,B8=10100101, SEL=x

Test 4: SEL=x, A!=B, OUT=x

30 ns, OUT8=xxxxxxxx, OUT5=xxxxx, OUT2=xx, OUT1=x

A8=11111111 ,B8=00000000, SEL=x

Test 5: SEL=x, A=B=1, OUT=A=0

40 ns, OUT8=11111111, OUT5=11111, OUT2=11, OUT1=1

A8=11111111 ,B8=11111111, SEL=x

Test 6: SEL=x, A=B=0, OUT=A=0

50 ns, OUT8=00000000, OUT5=00000, OUT2=00, OUT1=0

A8=00000000 ,B8=00000000, SEL=x

$finish called from file "scale\_tb.v", line 61.

$finish at simulation time 60

V C S S i m u l a t i o n R e p o r t

Time: 60 ns

CPU Time: 0.240 seconds; Data structure size: 0.0Mb

Thu Mar 1 19:27:43 2018

**Lab report question:**

This model will function the same as the gate-level model simulated in Lab 1 only if the behavioral model is used for a 1 bit multiplexer. Even the cases where SEL=x are similar in both models. The gate-level models shows its flaws when more than a single bit multiplexer is needed, because the module would need to be instantiated for each bit, which is very inefficient.

**Conclusion:**

Everything in this lab worked as expected without running into any issues. Special care was taken to minimize test vectors and to ensure that the bits would be changing from their previous result. One of the biggest takeaways from this lab is that parameters can be used to change a certain aspect of a module, like the bit width in this lab, to make it work for different use cases, without having to rewrite the code.