Ziyu (Selina) Wang

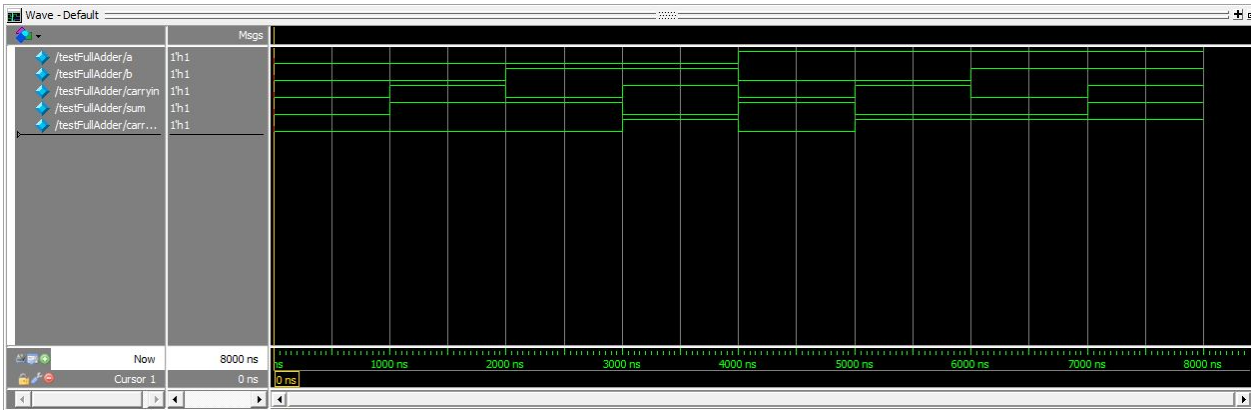
CompArch

Sep 21, 2015

HW2 Write-up

I. Results and Explanation:

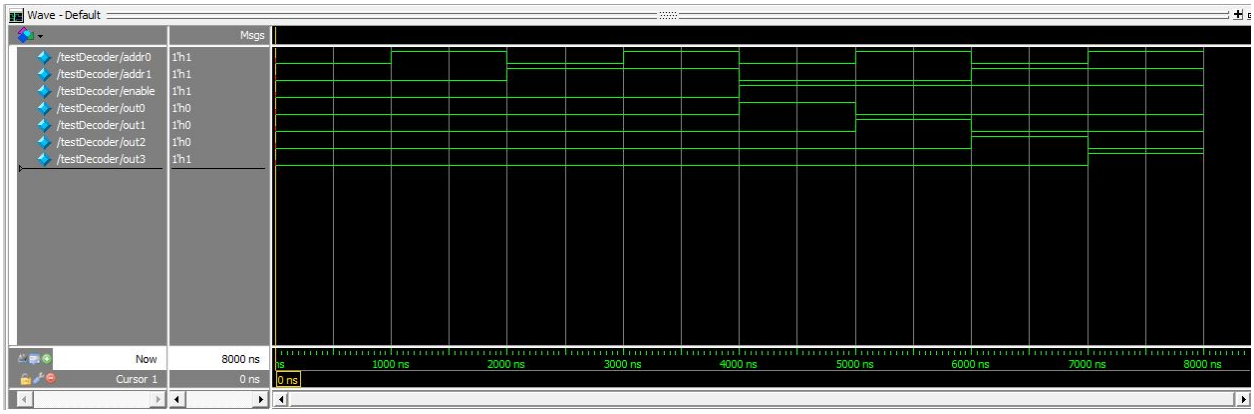
1. 1-bit Full Adder with Behavioral Verilog




```
# vsim
# Start time: 23:36:41 on Sep 21,2015
# Loading work.testFullAdder
# Loading work.behavioralFullAdder
# a b carryin | sum carryout | Expected Output
# 0 0    0    |  0     0     | 0  0
# 0 0    1    |  1     0     | 1  0
# 0 1    0    |  1     0     | 1  0
# 0 1    1    |  0     1     | 0  1
# 1 0    0    |  1     0     | 1  0
# 1 0    1    |  0     1     | 0  1
# 1 1    0    |  0     1     | 0  1
# 1 1    1    |  1     1     | 1  1
```

The test bench works because every possible input combination yields the correct output.

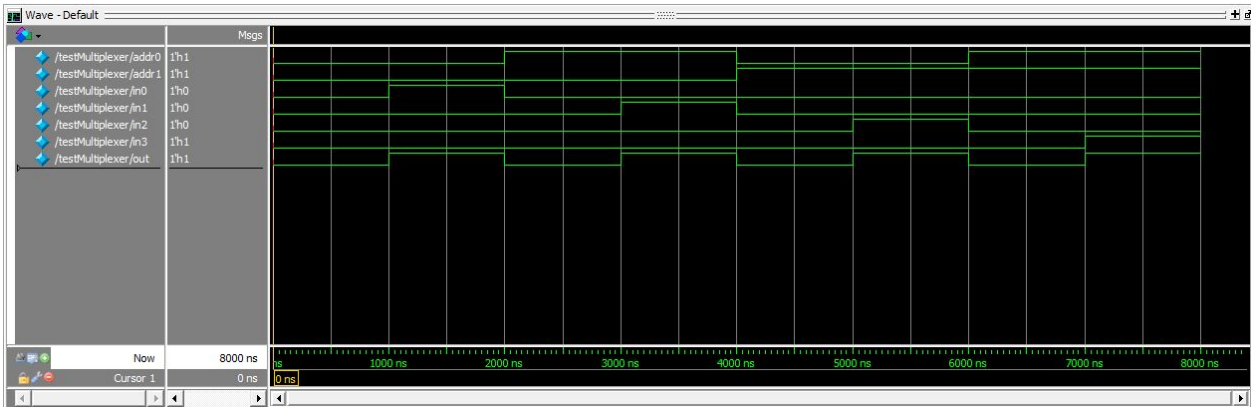
2. 2-bit Decoder with Enable with Behavioral Verilog




```
# vsim
# Start time: 23:39:54 on Sep 21,2015
# Loading work.testDecoder
# Loading work.behavioralDecoder
# En A0 A1| O0 O1 O2 O3 | Expected Output
# 0  0  0 |  0  0  0  0 | All false
# 0  1  0 |  0  0  0  0 | All false
# 0  0  1 |  0  0  0  0 | All false
# 0  1  1 |  0  0  0  0 | All false
# 1  0  0 |  1  0  0  0 | O0 Only
# 1  1  0 |  0  1  0  0 | O1 Only
# 1  0  1 |  0  0  1  0 | O2 Only
# 1  1  1 |  0  0  0  1 | O3 Only
```

The test bench works because every possible input combination yields the correct output.

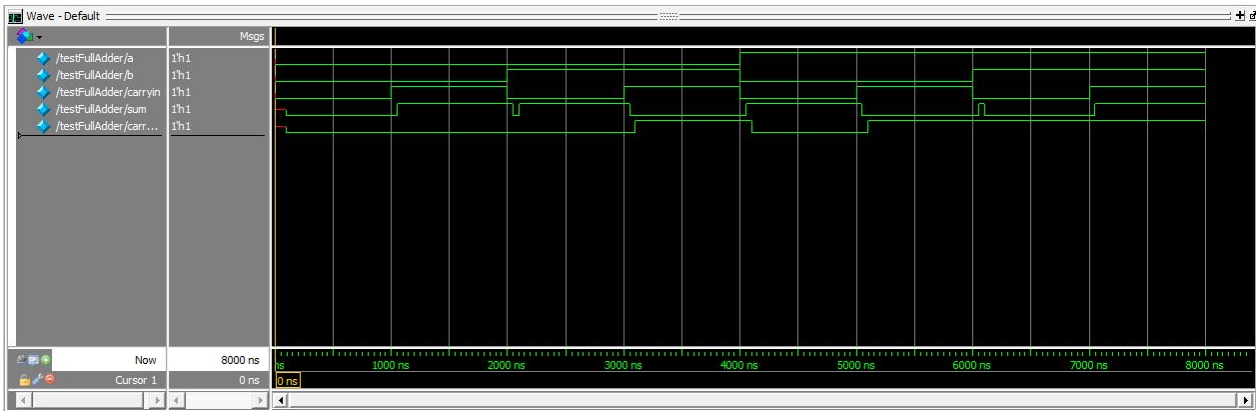
3. 4:1 Multiplexer with Behavioral Verilog



```
# vsim
# Start time: 23:41:40 on Sep 21,2015
# Loading work.testMultiplexer
# Loading work.behavioralMultiplexer
# I0 I1 I2 I3 A0 A1 | Out | Expected Output
# 0  0  0  0  0  0  |  0  |       0
# 1  0  0  0  0  0  |  1  |       1
# 0  0  0  0  1  0  |  0  |       0
# 0  1  0  0  1  0  |  1  |       1
# 0  0  0  0  0  1  |  0  |       0
# 0  0  1  0  0  1  |  1  |       1
# 0  0  0  0  1  1  |  0  |       0
# 0  0  0  1  1  1  |  1  |       1
```

Since when A0 and A1 are both 0 the multiplexer selects I0, the rest of the inputs do not matter and the same logic to all the situations. Therefore, I put in 0s for the non-important inputs in each situation instead of Xs since X in Verilog means the input is uninitialized or unknown. The test bench works because every possible input combination yields the correct output. The test bench works because every possible input combination (with inputs that matter in each situation) yields the correct output.

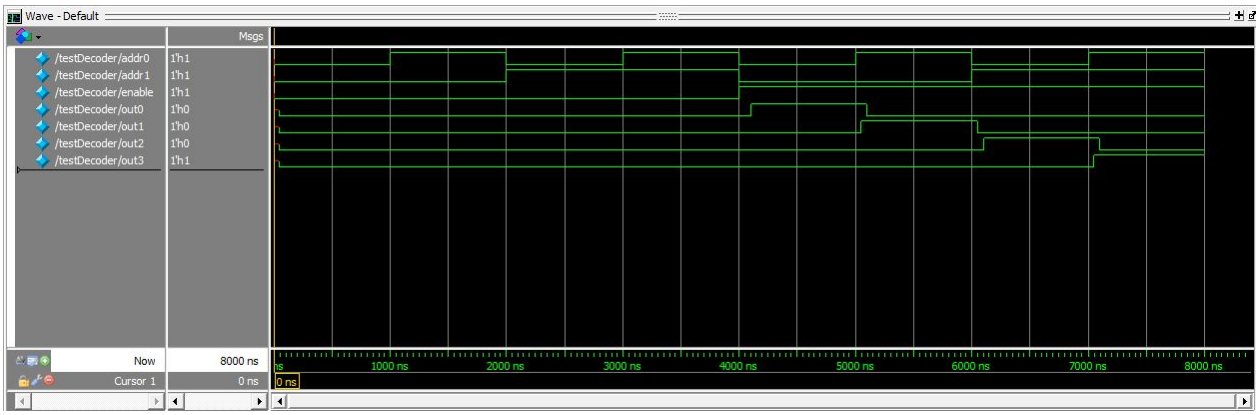
4. 1-bit Full Adder with Structural Verilog




```
# vsim
# Start time: 23:46:30 on Sep 21,2015
# Loading work.testFullAdder
# Loading work.structuralFullAdder
# a b carryin | sum carryout | Expected Output
# 0 0    0    |  0     0     | 0  0
# 0 0    1    |  1     0     | 1  0
# 0 1    0    |  1     0     | 1  0
# 0 1    1    |  0     1     | 0  1
# 1 0    0    |  1     0     | 1  0
# 1 0    1    |  0     1     | 0  1
# 1 1    0    |  0     1     | 0  1
# 1 1    1    |  1     1     | 1  1
```

The little jumps/glitches and delays at the end of a simulation you see in the waveforms are cause by the delays between inputs and outputs due to the 50 units of time delay assigned to each gate. The test bench works because every possible input combination yields the correct output.

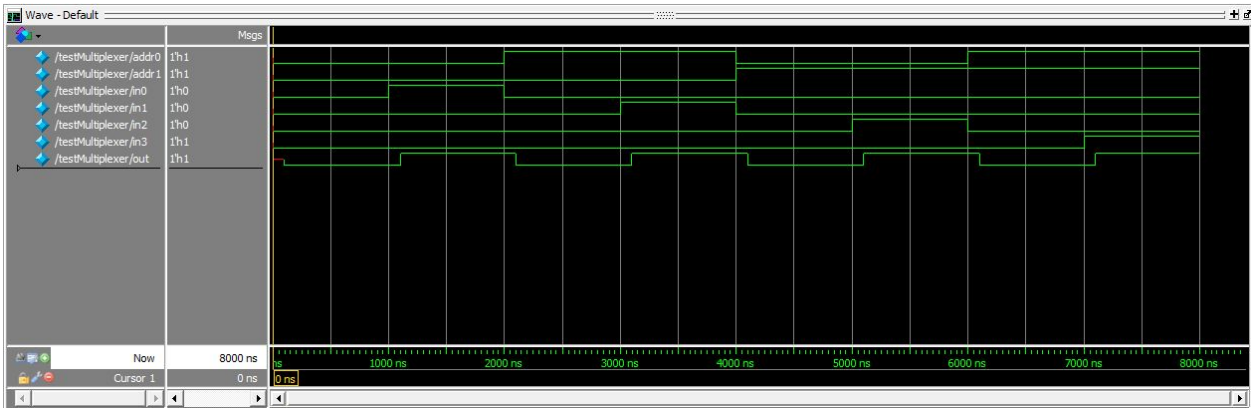
5. 2-bit Decoder with Enable with Structural Verilog




```
# vsim
# Start time: 23:43:23 on Sep 21,2015
# Loading work.testDecoder
# Loading work.structuralDecoder
# En A0 A1| O0 O1 O2 O3 | Expected Output
# 0  0  0 |  0  0  0  0 | All false
# 0  1  0 |  0  0  0  0 | All false
# 0  0  1 |  0  0  0  0 | All false
# 0  1  1 |  0  0  0  0 | All false
# 1  0  0 |  1  0  0  0 | O0 Only
# 1  1  0 |  0  1  0  0 | O1 Only
# 1  0  1 |  0  0  1  0 | O2 Only
# 1  1  1 |  0  0  0  1 | O3 Only
```

Delays at the end of a simulation you see in the waveforms are cause by the delays between inputs and outputs due to the 50 units of time delay assigned to each gate. The test bench works because every possible input combination yields the correct output.

6. 4:1 Multiplexer with Structural Verilog

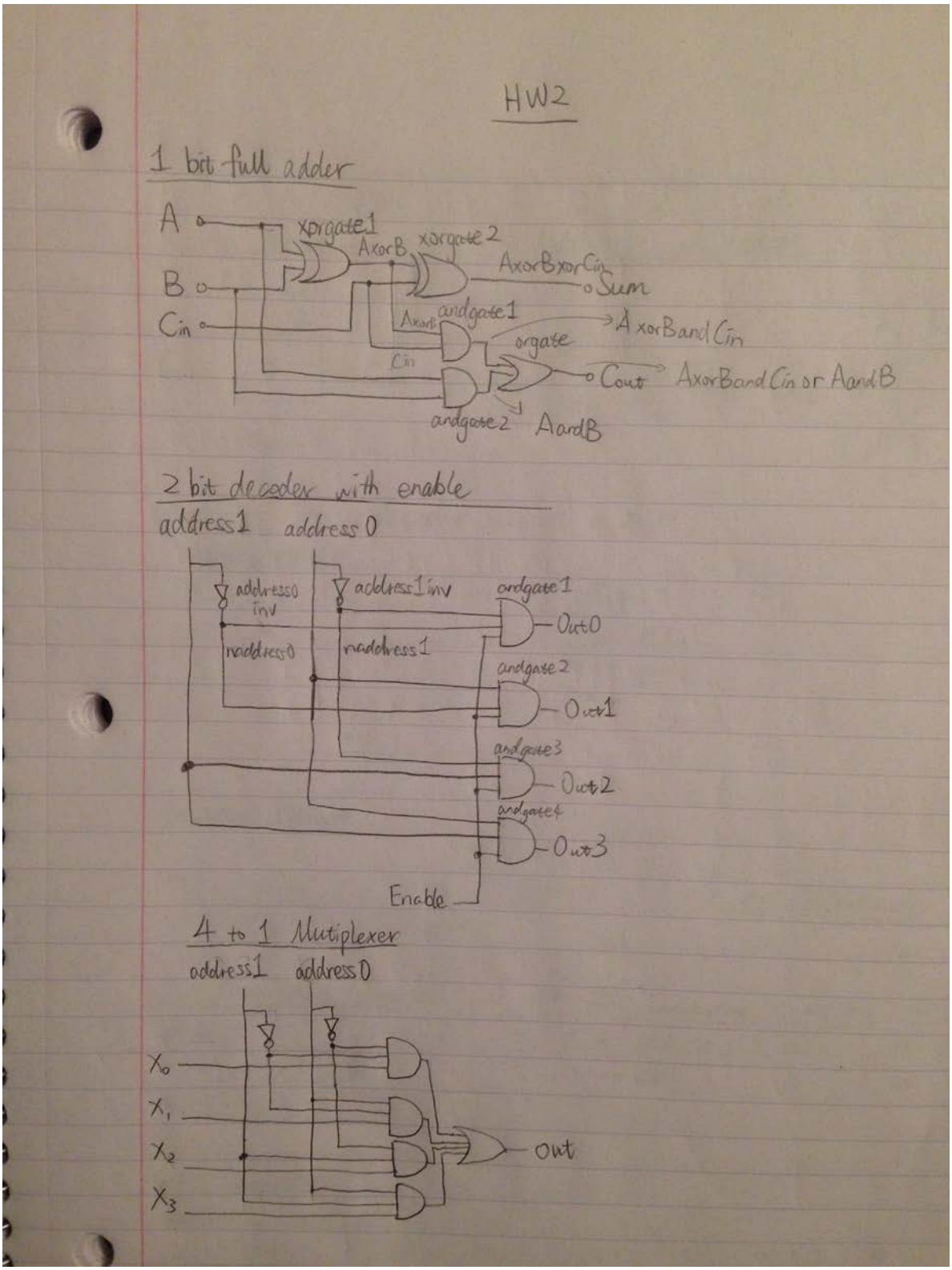


```
# vsim
# Start time: 23:44:54 on Sep 21,2015
# Loading work.testMultiplexer
# Loading work.structuralMultiplexer
# I0 I1 I2 I3 A0 A1 | Out | Expected Output
# 0  0  0  0  0  0  |  0  |      0
# 1  0  0  0  0  0  |  1  |      1
# 0  0  0  0  1  0  |  0  |      0
# 0  1  0  0  1  0  |  1  |      1
# 0  0  0  0  0  1  |  0  |      0
# 0  0  1  0  0  1  |  1  |      1
# 0  0  0  0  1  1  |  0  |      0
# 0  0  0  1  1  1  |  1  |      1
```

See explanation for #3. Delays at the end of a simulation you see in the waveforms are cause by the delays between inputs and outputs due to the 50 units of time delay assigned to each gate. The test bench works because every possible input combination yields the correct output.

II. Appendix

Below are my schematics and programs for generating the previous results.



1) 1 bit Full Adder

Adder.v

```
// define gates with delays
`define AND and #50
`define OR or #50
`define NOT not #50
`define XOR xor #50

module behavioralFullAdder(sum, carryout, a, b, carryin);
output sum, carryout;
input a, b, carryin;
assign {carryout, sum}=a+b+carryin;
endmodule

module structuralFullAdder(out, carryout, a, b, carryin);
output out, carryout;
input a, b, carryin;
wire axorb, axorbandcarryin, aandb;
`XOR xorgate1(axorb, a, b); // xor gate produces axorb from a and b
`XOR xorgate2(out, axorb, carryin); // xor gate produces out(sum) from axorb and carryin
`AND andgate1(axorbandcarryin, axorb, carryin); // and gate produces axorbandcarryinn from
axorb and carryin
`AND andgate2(aandb, a, b);
`OR orgate(carryout, axorbandcarryin, aandb);
endmodule

module testFullAdder;
reg a, b, carryin;
wire sum, carryout;
//behavioralFullAdder adder (sum, carryout, a, b, carryin);
structuralFullAdder adder (sum, carryout, a, b, carryin);

initial begin
$display("a b carryin | sum carryout | Expected Output");
a=0;b=0;carryin=0; #1000
$display("%b %b    %b    |  %b     %b     | 0  0", a, b, carryin, sum, carryout);
a=0;b=0;carryin=1; #1000
$display("%b %b    %b    |  %b     %b     | 1  0", a, b, carryin, sum, carryout);
a=0;b=1;carryin=0; #1000
$display("%b %b    %b    |  %b     %b     | 1  0", a, b, carryin, sum, carryout);
a=0;b=1;carryin=1; #1000
$display("%b %b    %b    |  %b     %b     | 0  1", a, b, carryin, sum, carryout);
a=1;b=0;carryin=0; #1000
$display("%b %b    %b    |  %b     %b     | 1  0", a, b, carryin, sum, carryout);
a=1;b=0;carryin=1; #1000
$display("%b %b    %b    |  %b     %b     | 0  1", a, b, carryin, sum, carryout);
a=1;b=1;carryin=0; #1000
$display("%b %b    %b    |  %b     %b     | 0  1", a, b, carryin, sum, carryout);
a=1;b=1;carryin=1; #1000
$display("%b %b    %b    |  %b     %b     | 1  1", a, b, carryin, sum, carryout);
end
endmodule
```

Adder.do

```
vlog -reportprogress 300 -work work adder.v
vsim -voptargs="+acc" testFullAdder
add wave -position insertpoint  \
sim:/testFullAdder/a \
sim:/testFullAdder/b \
sim:/testFullAdder/carryin \
sim:/testFullAdder/sum \
```

```
sim:/testFullAdder/carryout
run -all
wave zoom full
```

2) 2 bit Decoder with Enable

`Decoder.v`

```
// define gates with delays
`define AND and #50
`define OR or #50
`define NOT not #50
`define XOR xor #50

module behavioralDecoder(out0,out1,out2,out3, address0,address1, enable);
output out0, out1, out2, out3;
input address0, address1;
input enable;
assign {out3,out2,out1,out0}=enable<<{address1,address0};
endmodule

module structuralDecoder(out0,out1,out2,out3, address0,address1, enable);
output out0, out1, out2, out3;
input address0, address1;
input enable;
wire naddress0, naddress1;
`NOT address0inv(naddress0, address0); // inverter produces signal naddress0 and takes
signal address0, and is named address0inv
`NOT address1inv(naddress1, address1); // inverter produces signal naddress1 and takes
signal address1, and is named address1inv
`AND andgate1(out0, naddress0, naddress1, enable); // and gate produces out0 from naddress0,
naddress1, and enable
`AND andgate2(out1, address0, naddress1, enable); // and gate produces out1 from address0,
naddress1, and enable
`AND andgate3(out2, naddress0, address1, enable); // and gate produces out2 from naddress0,
address1, and enable
`AND andgate4(out3, address0, address1, enable); // and gate produces out3 from address0,
address1, and enable
endmodule

module testDecoder;
reg addr0, addr1;
reg enable;
wire out0,out1,out2,out3;
//behavioralDecoder decoder (out0,out1,out2,out3,addr0,addr1,enable);
structuralDecoder decoder (out0,out1,out2,out3,addr0,addr1,enable); // Swap after testing

initial begin
$display("En A0 A1| O0 O1 O2 O3 | Expected Output");
enable=0;addr0=0;addr1=0; #1000
$display("%b  %b  %b |  %b  %b  %b  %b | All false", enable, addr0, addr1, out0, out1, out2,
out3);
enable=0;addr0=1;addr1=0; #1000
$display("%b  %b  %b |  %b  %b  %b  %b | All false", enable, addr0, addr1, out0, out1, out2,
out3);
enable=0;addr0=0;addr1=1; #1000
$display("%b  %b  %b |  %b  %b  %b  %b | All false", enable, addr0, addr1, out0, out1, out2,
out3);
enable=0;addr0=1;addr1=1; #1000
$display("%b  %b  %b |  %b  %b  %b  %b | All false", enable, addr0, addr1, out0, out1, out2,
out3);
enable=1;addr0=0;addr1=0; #1000
$display("%b  %b  %b |  %b  %b  %b  %b | O0 Only", enable, addr0, addr1, out0, out1, out2,
out3);
enable=1;addr0=1;addr1=0; #1000
```

```
$display("%b  %b  %b |  %b  %b  %b  %b | O1 Only", enable, addr0, addr1, out0, out1, out2,
out3);
enable=1;addr0=0;addr1=1; #1000
$display("%b  %b  %b |  %b  %b  %b  %b | O2 Only", enable, addr0, addr1, out0, out1, out2,
out3);
enable=1;addr0=1;addr1=1; #1000
$display("%b  %b  %b |  %b  %b  %b  %b | O3 Only", enable, addr0, addr1, out0, out1, out2,
out3);
end
endmodule
```

`Decoder.do`

```
vlog -reportprogress 300 -work work decoder.v
vsim -voptargs="+acc" testDecoder
add wave -position insertpoint  \
sim:/testDecoder/addr0 \
sim:/testDecoder/addr1 \
sim:/testDecoder/enable \
sim:/testDecoder/out0 \
sim:/testDecoder/out1 \
sim:/testDecoder/out2 \
sim:/testDecoder/out3
run -all
wave zoom full
```

3) 4:1 Multiplexer

`Multiplexer.v`

```
// define gates with delays
`define AND and #50
`define OR or #50
`define NOT not #50
`define XOR xor #50

module behavioralMultiplexer(out, address0,address1, in0,in1,in2,in3);
output out;
input address0, address1;
input in0, in1, in2, in3;
wire[3:0] inputs = {in3, in2, in1, in0};
wire[1:0] address = {address1, address0};
assign out = inputs[address];
endmodule

module structuralMultiplexer(out, address0,address1, in0,in1,in2,in3);
output out;
input address0, address1;
input in0, in1, in2, in3;
wire naddress0, naddress1, naddress0andnaddress1andin0, address0andnaddress1andin1,
naddress0andaddress1andin2, address0andaddress1andin3;
`NOT address0inv(naddress0, address0); // inverter produces signal naddress0 and takes
signal address0, and is named address0inv
`NOT address1inv(naddress1, address1); // inverter produces signal naddress1 and takes
signal address1, and is named address1inv
`AND andgate1(naddress0andnaddress1andin0, naddress0, naddress1, in0); // and gate produces
out0 from naddress0, naddress1, and in0
`AND andgate2(address0andnaddress1andin1, address0, naddress1, in1); // and gate produces
out1 from address0, naddress1, and in1
`AND andgate3(naddress0andaddress1andin2, naddress0, address1, in2); // and gate produces
out2 from naddress0, address1, and in2
`AND andgate4(address0andaddress1andin3, address0, address1, in3); // and gate produces
out3 from address0, address1, and in3
```

```
`OR orgate(out, naddress0andnaddress1andin0, address0andnaddress1andin1,
naddress0andaddress1andin2, address0andaddress1andin3);
endmodule

module testMultiplexer;
reg addr0, addr1;
reg in0, in1, in2, in3;
wire out;
//behavioralMultiplexer multiplexer (out, addr0,addr1, in0,in1,in2,in3);
structuralMultiplexer multiplexer (out, addr0,addr1, in0,in1,in2,in3);

initial begin
$display("I0 I1 I2 I3 A0 A1 | Out | Expected Output");
in0=0;in1=0;in2=0;in3=0;addr0=0;addr1=0; #1000
$display("%b  %b  %b  %b  %b  %b  |  %b  |      0", in0, in1, in2, in3, addr0, addr1, out);
in0=1;in1=0;in2=0;in3=0;addr0=0;addr1=0; #1000
$display("%b  %b  %b  %b  %b  %b  |  %b  |      1", in0, in1, in2, in3, addr0, addr1, out);
in0=0;in1=0;in2=0;in3=0;addr0=1;addr1=0; #1000
$display("%b  %b  %b  %b  %b  %b  |  %b  |      0", in0, in1, in2, in3, addr0, addr1, out);
in0=0;in1=1;in2=0;in3=0;addr0=1;addr1=0; #1000
$display("%b  %b  %b  %b  %b  %b  |  %b  |      1", in0, in1, in2, in3, addr0, addr1, out);
in0=0;in1=0;in2=0;in3=0;addr0=0;addr1=1; #1000
$display("%b  %b  %b  %b  %b  %b  |  %b  |      0", in0, in1, in2, in3, addr0, addr1, out);
in0=0;in1=0;in2=1;in3=0;addr0=0;addr1=1; #1000
$display("%b  %b  %b  %b  %b  %b  |  %b  |      1", in0, in1, in2, in3, addr0, addr1, out);
in0=0;in1=0;in2=0;in3=0;addr0=1;addr1=1; #1000
$display("%b  %b  %b  %b  %b  %b  |  %b  |      0", in0, in1, in2, in3, addr0, addr1, out);
in0=0;in1=0;in2=0;in3=1;addr0=1;addr1=1; #1000
$display("%b  %b  %b  %b  %b  %b  |  %b  |      1", in0, in1, in2, in3, addr0, addr1, out);
end
endmodule
```

Multiplexer.do

```
vlog -reportprogress 300 -work work multiplexer.v decoder.v adder.v
vsim -voptargs="+acc" testMultiplexer
add wave -position insertpoint  \
sim:/testMultiplexer/addr0 \
sim:/testMultiplexer/addr1 \
sim:/testMultiplexer/in0 \
sim:/testMultiplexer/in1 \
sim:/testMultiplexer/in2 \
sim:/testMultiplexer/in3 \
sim:/testMultiplexer/out
run -all
wave zoom full
```