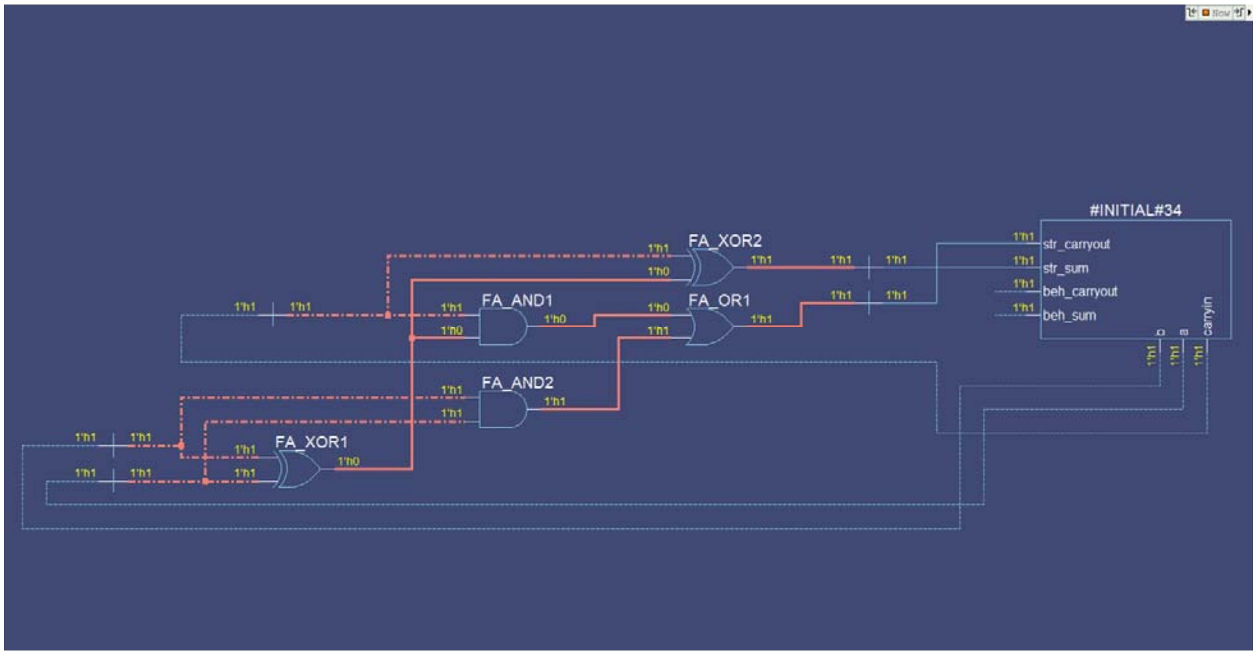
Computer Architecture HW1

Ryan Eggert

Full Adder

Schematic



The full adder was made of two XOR gates, two AND gates, and one OR gate [all 2-input]. The schematic of the system implemented in structural Verilog is shown above. For the Verilog implementation and testbench, please see `adder.v`. For execution of the testbench, please see `adders.do`. Notice that any signal will travel through, at most, three gates.

Notice that all testbenches in this assignment test both their respective structural implementation and behavioral implementation at once. Doing so allows the results to easily be displayed in the same truth table and waveform display, making the comparisons called for in this assignment more natural.

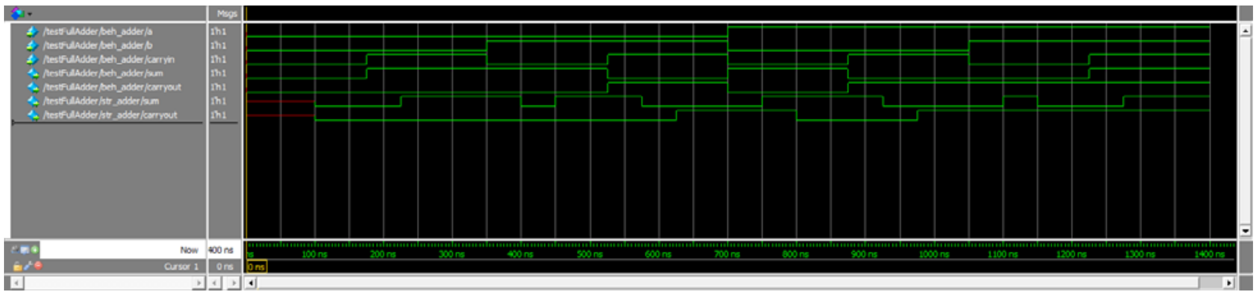
Truth Tables

```
# Loading work.testFullAdder
# Loading work.behavioralFullAdder
# Loading work.structuralFullAdder
#  Inputs   | Behavioral | Structural | Expected
# A B C_In | Sum C_Out  | Sum C_Out  |Sum C_Out
# 0 0 0    |  0  0      |  0  0      | 0   0
# 0 0 1    |  1  0      |  1  0      | 1   0
# 0 1 0    |  1  0      |  1  0      | 1   0
# 0 1 1    |  0  1      |  0  1      | 0   1
# 1 0 0    |  1  0      |  1  0      | 1   0
# 1 0 1    |  0  1      |  0  1      | 0   1
# 1 1 0    |  0  1      |  0  1      | 0   1
# 1 1 1    |  1  1      |  1  1      | 1   1
```

Above is the output from my testbench module. We test every possible combination of the full adder's three inputs. The both the behavioral and structural implementations of the full adder match the expected outputs for any given set of inputs.

Waveforms

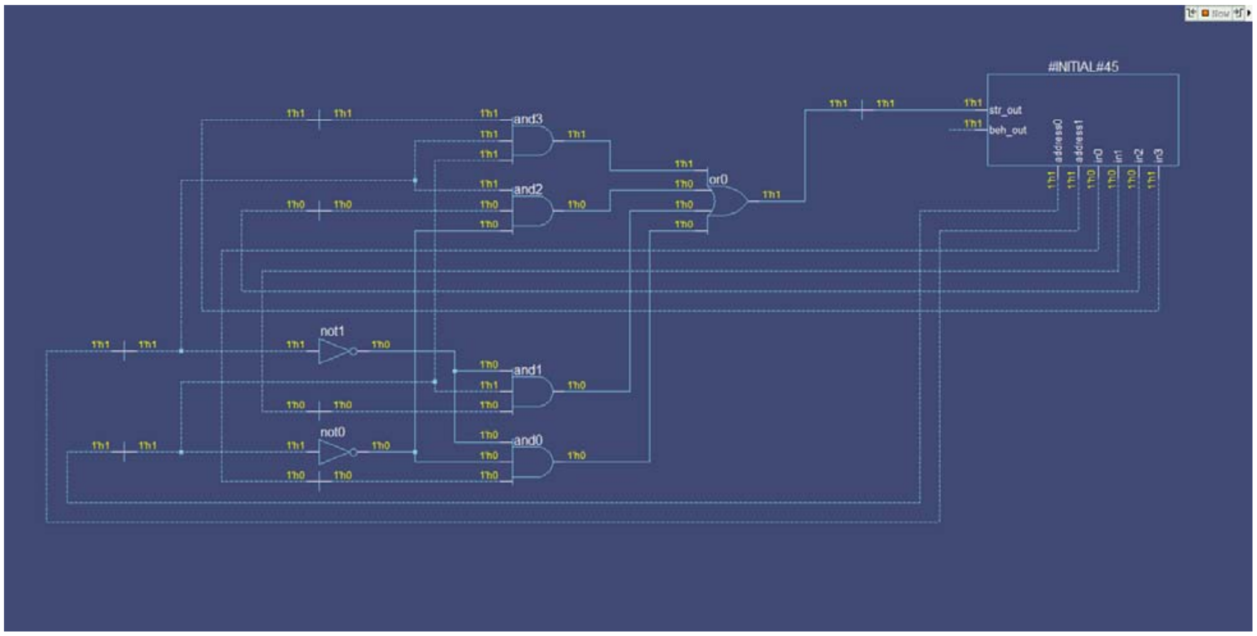
The waveforms above show the adders' three inputs, the two outputs of the behavioral adder, and the two outputs of the structural adder. Notice the delays present in the structural implementation and not in the behavioral implementation. With the exception of these delays, the behavioral implementation behaves in the same manner as the structural implementation.


/testFullAdder/beh_adder/a
/testFullAdder/beh_adder/b
/testFullAdder/beh_adder/carryin
/testFullAdder/beh_adder/sum
/testFullAdder/beh_adder/carryout
/testFullAdder/str_adder/sum
/testFullAdder/str_adder/carryout
Msgs
Now
Cursor 1
0 ns
100 ns
200 ns
300 ns
400 ns
500 ns
600 ns
700 ns
800 ns
900 ns
1000 ns
1100 ns
1200 ns
1300 ns
1400 ns

Note, for example, the "glitch" in the structural implementation when the b-bit goes from low to high while the carryin bit goes from high to low. The sum bit temporarily goes low, a behavior not observed in the non-delayed behavioral implementation. This is because the carryin bit goes directly to the second XOR gate while the b-bit must be processed by the first XOR gate before its change propagates to the second XOR gate.

Multiplexer

Schematic



The full adder was made of two NOT gates, four three-input AND gates, and one four-input OR gate. The schematic of the system implemented in structural Verilog is shown above. For the Verilog implementation and testbench, please see `multiplexer.v`. For execution of the testbench, please see `muxes.do`. Notice that any signal will travel through, at most, three gates.

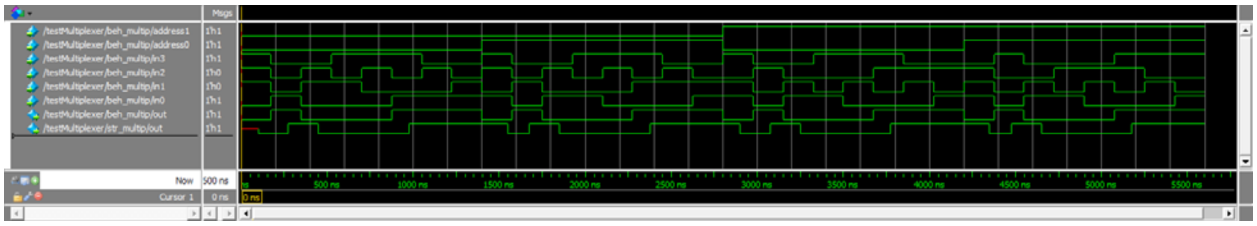
Truth Tables

```
# Start time: 15:36:52 on Sep 25,2015
# Loading work.testMultiplexer
# Loading work.behavioralMultiplexer
# Loading work.structuralMultiplexer
#         Inputs         | Beh.| Str.| Exp.
# In3 In2 In1 In0 | A1 A0 | Out | Out | Out
# 1   1   1   0   | 0  0  | 0   | 0   | 0
# 0   0   0   1   | 0  0  | 1   | 1   | 1
# 0   1   1   0   | 0  0  | 0   | 0   | 0
# 1   0   1   0   | 0  0  | 0   | 0   | 0
# 1   1   0   0   | 0  0  | 0   | 0   | 0
# 1   0   0   1   | 0  0  | 1   | 1   | 1
# 0   1   0   1   | 0  0  | 1   | 1   | 1
# 0   0   1   1   | 0  0  | 1   | 1   | 1
# 1   1   0   1   | 0  1  | 0   | 0   | 0
# 0   0   1   0   | 0  1  | 1   | 1   | 1
# 0   1   0   1   | 0  1  | 0   | 0   | 0
# 1   0   0   1   | 0  1  | 0   | 0   | 0
# 1   1   0   0   | 0  1  | 0   | 0   | 0
# 1   0   1   0   | 0  1  | 1   | 1   | 1
# 0   1   1   0   | 0  1  | 1   | 1   | 1
# 0   0   1   1   | 0  1  | 1   | 1   | 1
# 1   0   1   1   | 1  0  | 0   | 0   | 0
# 0   1   0   0   | 1  0  | 1   | 1   | 1
# 0   0   1   1   | 1  0  | 0   | 0   | 0
# 1   0   0   1   | 1  0  | 0   | 0   | 0
# 1   0   1   0   | 1  0  | 0   | 0   | 0
# 1   1   0   0   | 1  0  | 1   | 1   | 1
# 0   1   1   0   | 1  0  | 1   | 1   | 1
# 0   1   0   1   | 1  0  | 1   | 1   | 1
# 0   1   1   1   | 1  1  | 0   | 0   | 0
# 1   0   0   0   | 1  1  | 1   | 1   | 1
# 0   0   1   1   | 1  1  | 0   | 0   | 0
# 0   1   0   1   | 1  1  | 0   | 0   | 0
# 0   1   1   0   | 1  1  | 0   | 0   | 0
# 1   1   0   0   | 1  1  | 1   | 1   | 1
# 1   0   1   0   | 1  1  | 1   | 1   | 1
# 1   0   0   1   | 1  1  | 1   | 1   | 1
```

Above is the output from my testbench module. Notice that I do not test every combination of six binary inputs; instead I show that when the two address bits select input *n*, the output is input *n* regardless of whether input *n* is true or false. I then show that when the two address bits select input *n*, none of the other input bits can individually change the output. Because each of the input [In3, In2, In1, In0] are independently processed by separate AND gates (which feed into an OR gate), there is no possibility of two or more non-selected inputs changing the output. This process is repeated for every combination of address bits.

Both the structural and behavioral implementations behave as expected.

Waveforms



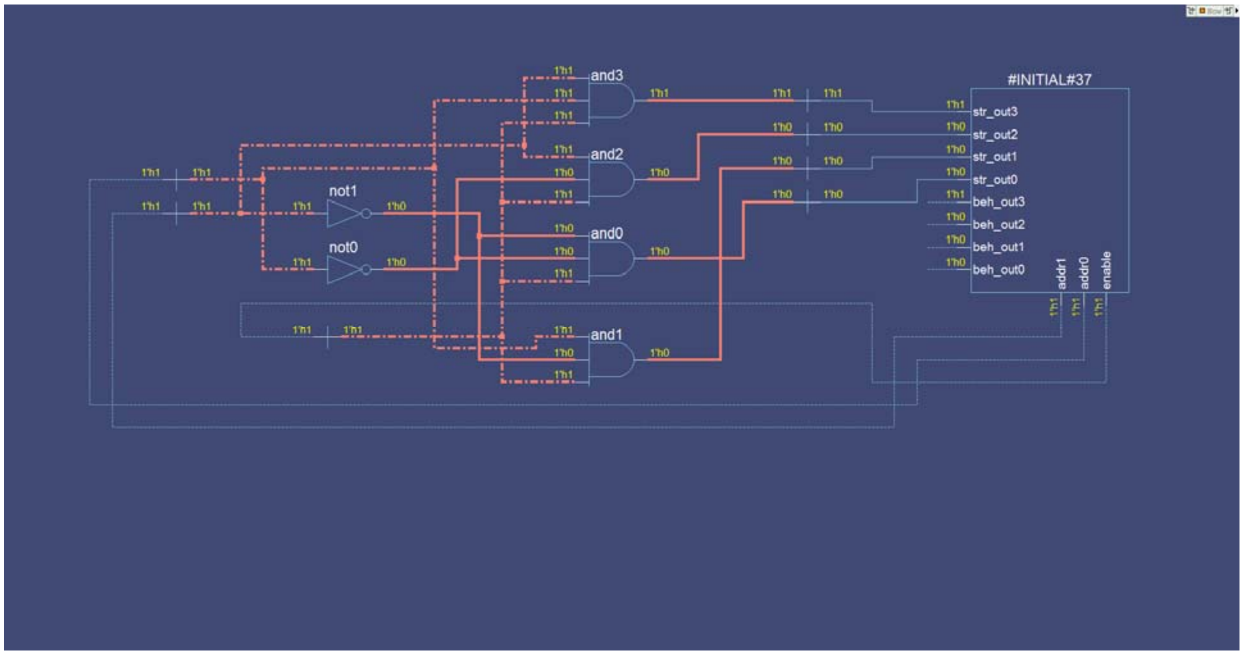
```
/testMultiplexer/beh_multip/address1   1'h1
/testMultiplexer/beh_multip/address0   1'h1
/testMultiplexer/beh_multip/in3        1'h1
/testMultiplexer/beh_multip/in2        1'h0
/testMultiplexer/beh_multip/in1        1'h0
/testMultiplexer/beh_multip/in0        1'h1
/testMultiplexer/beh_multip/out        1'h1
/testMultiplexer/str_multip/out        1'h1
                                 Now   500 ns
                            Cursor 1   0 ns
```

The waveforms above show the multiplexers' six inputs, the output of the behavioral multiplexer, and the output of the structural multiplexer. Notice the delays present in the structural implementation and not in the behavioral implementation. With the exception of these delays, the behavioral implementation behaves in the same manner as the structural implementation.

The waveforms show "glitches" which generally occur on address bit changes. This is due to some of the address signals being delayed by NOT gates. Transient states occur while the noninverting address signals wait for their inverting counterparts to arrive.

Decoder

Schematic



The 2-bit enabled decoder was made of two NOT gates, and four three-input AND gates. The schematic of the system implemented in structural Verilog is shown above. For the Verilog implementation and testbench, please see `decoder.v`. For execution of the testbench, please see `decoders.do`. Notice that any signal will travel through, at most, two gates.

Notice that the decoder very closely resembles a subset of the previously discussed multiplexer. This is because the multiplexer decodes the two-bit address signal when selecting inputs.

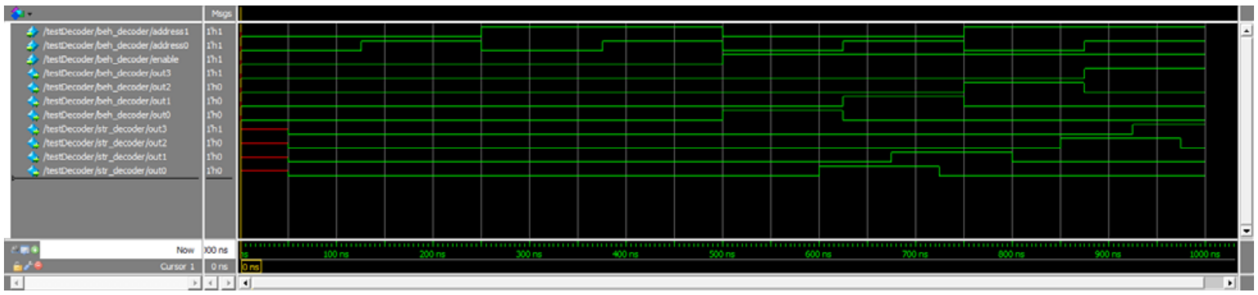
Truth Tables

```
# Loading work.testDecoder
# Loading work.behavioralDecoder
# Loading work.structuralDecoder
#  Inputs | Behavioral  | Structural  | Expected
# En A0 A1| O0 O1 O2 O3 | O0 O1 O2 O3 | Outputs
# 0  0  0 |  0  0  0  0 |  0  0  0  0 | All false
# 0  1  0 |  0  0  0  0 |  0  0  0  0 | All false
# 0  0  1 |  0  0  0  0 |  0  0  0  0 | All false
# 0  1  1 |  0  0  0  0 |  0  0  0  0 | All false
# 1  0  0 |  1  0  0  0 |  1  0  0  0 | O0 Only
# 1  1  0 |  0  1  0  0 |  0  1  0  0 | O1 Only
# 1  0  1 |  0  0  1  0 |  0  0  1  0 | O2 Only
# 1  1  1 |  0  0  0  1 |  0  0  0  1 | O3 Only
```

Above is the output from the testbench module. Please note that although the testbench module was modified by me to simultaneously test the behavioral and structural implementations, the test cases were pre-written and provided with the assignment.

Notice that both the behavioral and structural implementations behave as expected.

Waveforms


Msgs
/testDecoder/beh_decoder/address1 1'h1
/testDecoder/beh_decoder/address0 1'h1
/testDecoder/beh_decoder/enable 1'h1
/testDecoder/beh_decoder/out3 1'h1
/testDecoder/beh_decoder/out2 1'h0
/testDecoder/beh_decoder/out1 1'h0
/testDecoder/beh_decoder/out0 1'h0
/testDecoder/str_decoder/out3 1'h1
/testDecoder/str_decoder/out2 1'h0
/testDecoder/str_decoder/out1 1'h0
/testDecoder/str_decoder/out0 1'h0
Now 000 ns
Cursor 1 0 ns
100 ns
200 ns
300 ns
400 ns
500 ns
600 ns
700 ns
800 ns
900 ns
1000 ns
0 ns

The waveforms above show the decoders' three inputs, the four outputs of the behavioral decoder, and the two outputs of the structural decoder. Notice the delays present in the structural implementation and not in the behavioral implementation. With the exception of these delays, the behavioral implementation behaves in the same manner as the structural implementation.