Laboratory 2
Logic, Switches, Lights, and Multiplexers

The purpose of this exercise is to learn how to connect simple input and output devices to an FPGA chip and implement a circuit that uses these devices. We will use the switches SW9−0 on the DE1-SoC board as inputs to the circuit. We will use light emitting diodes (LEDs) and 7-segment displays as output devices.

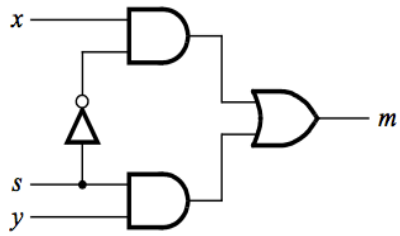
However, we will first complete everything in Simulation.

Part I – Practicing Instantiation, Verilog high-level statements

Figure 2a shows a sum-of-products circuit that implements a (2-to-1) 2:1 *multiplexer* with a select input s:

If s = 0 the multiplexer's output m is equal to the input x,
else s = 1 the multiplexer's output m is equal to the input y,

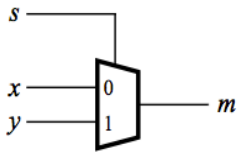
Part b of the figure gives a truth table for this multiplexer, and part c shows its circuit symbol



a) Circuit

s	m
0	x
1	y

b) Truth table



c) Symbol

Figure 2. A 2-to-1 multiplexer.

This 1-bit 2:1 multiplexer can be described by the following statement:

```
m <= (NOT (s) AND x) OR (s AND y);
```

A verilog template is given for this in one_bit_2to1mux.v and a testbench: one_bit_2to1mux_TB.v

1. Create a new Quartus project for your circuit. Include both files
2. Study the testbench. What outputs do you expect for each input. Write this down
3. Compile and test whether this code
4. It should compile, but there is a deliberate bug in the code. Find/Fix it!)

For the next part of this exercise, we want to translate this 1-bit 2:1 multiplexer into a 4-bit 2:1 multiplexer. It should exhibit the following behaviour:

If s = 0 the multiplexer's output m is equal to the 4-bit input x,
else s = 1 the multiplexer's output m is equal to the 4-bit input y,

It has the circuit symbol shown in Figure 3b, in which X, Y, and M are depicted as 4-bit wires.

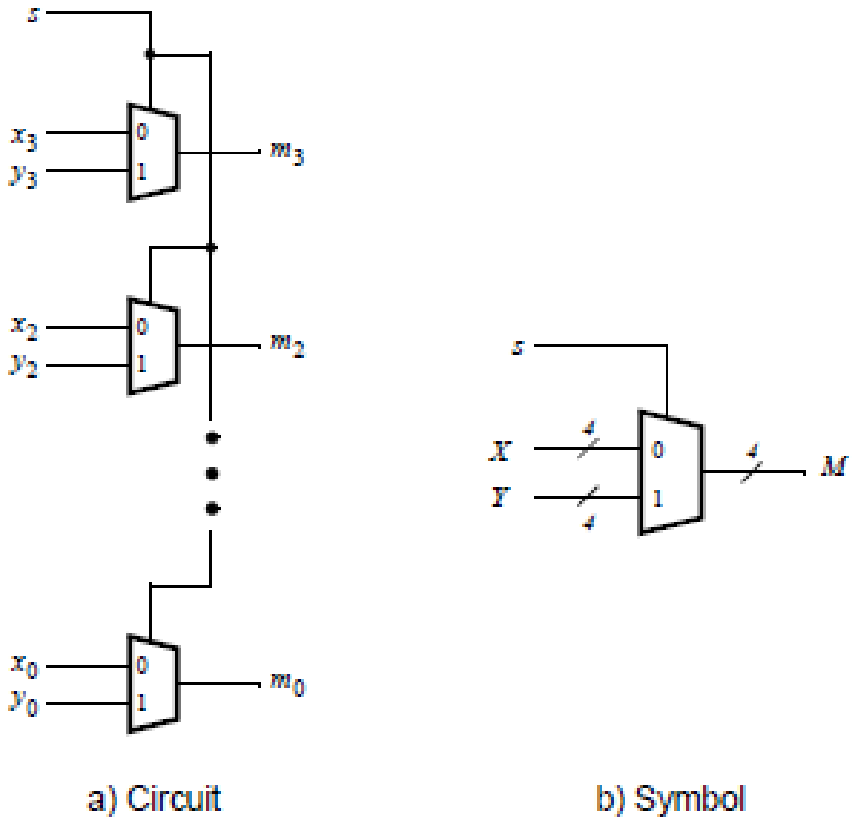


Figure 3. An four-bit wide 2-to-1 multiplexer.

Perform the following steps:

5. Create a new Quartus project for your circuit.
6. Include:
   - the template four-bit wide 2:1 multiplexer in your project (four_bit_2to1mux.v).
   - the provided 1-bit wide 2:1 multiplexer in your project (one_bit_2to1mux.v).
   - the testbench for this 1-bit wide 2-to-1 multiplexer in your project (part1_TB.v).
7. Study the testbench. What outputs do you expect for each input. Write this down.
8. Are you comfortable changing any test vectors? What do you expect to happen (be ready to discuss with a demonstrator
9. Complete the code and test that it works
10. Complete (and understand!) the following template for a high level verilog coding style to implement the same 4-bit 2:1 mux without needing to use the low level Boolean equations and multiple instantiations: four_bit_2to1muxV2.v
11. Test this second module using the same testbench (altered as necessary).

Get a demonstrator to check your understanding of these two versions of four-bit wide 2:1 multiplexers and mark completion

Part II - Practicing Instantiation, Verilog high-level statements again

In Figure 2 we showed a 1-bit 2:1 multiplexer that selects between the two inputs *x* and *y* based on a 1-bit select input s.
Now we wish to generate a 1-bit 4:1 multiplexer that selects between the two inputs *x* and *y* based on a 2-bit select input s(1:0).

s = 00 the multiplexer's output m is equal to the input u,
s = 01 the multiplexer's output m is equal to the input v,
s = 10 the multiplexer's output m is equal to the input w,
s = 11 the multiplexer's output m is equal to the input w,

A circuit symbol to achieve this functionality is shown below:

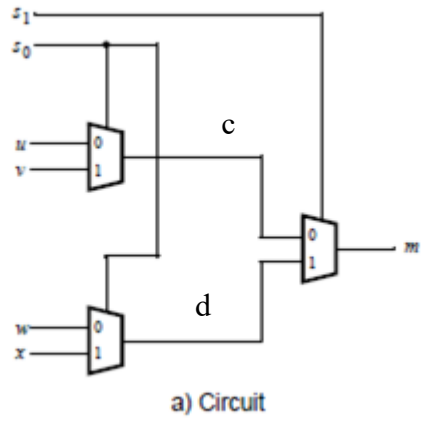


Figure 4. A two-bit wide 4-to-1 multiplexer.

1. Create a new Quartus project for your circuit.
2. Include:
   - the template 1-bit wide 4:1 multiplexer in your project (one_bit_4to1mux.v).
   - the provided 1-bit wide 2:1 multiplexer in your project (one_bit_2to1mux.v). (from part1)
   - the testbench for this 1-bit wide 4:1 multiplexer in your project (part2_TB.v).
3. Study the testbench. What outputs do you expect for each input. Write this down
4. Add some of your own test vectors to the testbench. Be ready to explain why you added these to your demonstrators
5. Complete the code and test that it works
6. Complete (and understand!) the following template for a high level verilog coding style to implement the 1-bit 4:1 mux without needing to use the low level Boolean equations and multiple instantiations: one_bit_4to1muxV2.v
7. Test this second module using the same testbench

Get a demonstrator to check your understanding of these two versions (and the testbench) of 1-bit wide 4:1 multiplexers and mark completion

Part III – Can you put together what you learnt from part 1 and part 2

Figure 5 extends this concept to a two-bit wide 4:1 multiplexer:

s = 00 the multiplexer's output m is equal to the 2-bit input u,
s = 01 the multiplexer's output m is equal to the 2-bit input v,
s = 10 the multiplexer's output m is equal to the 2-bit input w,
s = 11 the multiplexer's output m is equal to the 2-bit input w,

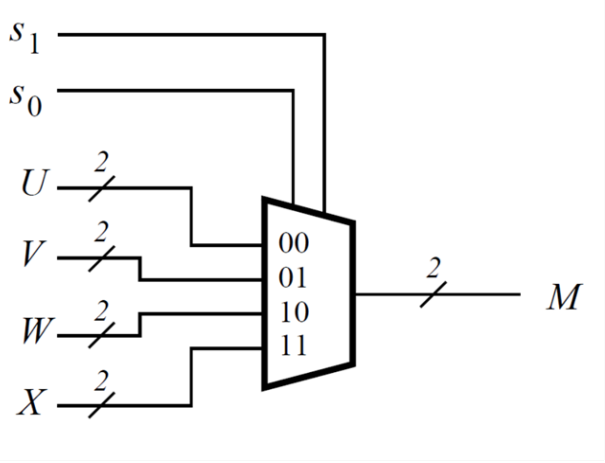


Figure 5. A two-bit wide 4-to-1 multiplexer.

1. Once again, complete this using instantiation and high level Verilog:
2. Create a hand-drawing of how to implement this circuit using two instances of 1-bit 4:1 muxes (look at part 1)
3. Create a new Quartus project for your circuit
4. Include:
   - One of your working modules for part 2 and all its submodules (one_bit_4to1mux.v and one_bit_2to1mux.v) or one_bit_2to1muxV2.v
   - the template for a 2-bit wide 4:1 multiplexer in your project (two_bit_4to1mux.v).
   - the testbench for this 2-bit wide 4:1 multiplexer in your project (part3_TB.v).
5. Complete the code and test that it works
6. Complete (and understand!) the template for a high level verilog coding style to implement the 1-bit 4:1 mux without needing to use the low level Boolean equations and multiple instantiations: two_bit_4to1muxV2.v
7. Test this second module using the same testbench

Get a lab demonstrator to check your understanding of all of these sections (including your drawing) and mark completion

AIM TO GET AT LEAST HERE BY THE END OF THE LAB

Templates below here only provided after next week's lecture, but feel free to have a go yourself, writing from scratch – If you can do this already, you are well set for the course!

Part IV – Implementing a decoder with a truth table

The objective of this part is to prepare to display a character on a 7-segment display, with the specific character displayed depends on a two-bit input. Figure 6 shows a 7-segment decoder entity that has the two-bit input C(1:0).

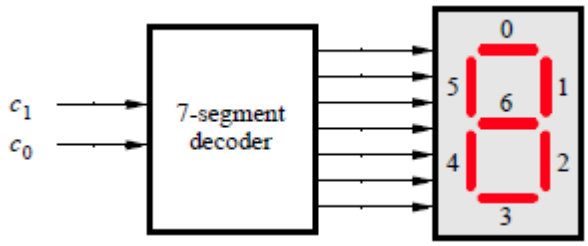


Figure 6. A 7-segment decoder.

When you implement on an FPGA, you will have to design specific codes. However, since we first want to test simulation only, we will make a simplified decoder:

- When C(1:0) = 00; Display the binary code: 0000111
- When C(1:0) = 01; Display the binary code: 1110000
- When C(1:0) = 10; Display the binary code: 0011100
- When C(1:0) = 11; Display the binary code: 1100011

Perform the following steps:

1. Add to your project**:**
   - The template decoder: **decoder_7seg.v**
   - The provided testbench: **part4_TB.v**

2. Why does your circuit output the same value most of the time when using the testbench?

Complete the template and get a lab demonstrator to check your understanding of all of these sections and mark completion

Part V – Putting it all together

Consider the circuit shown in Figure 7:

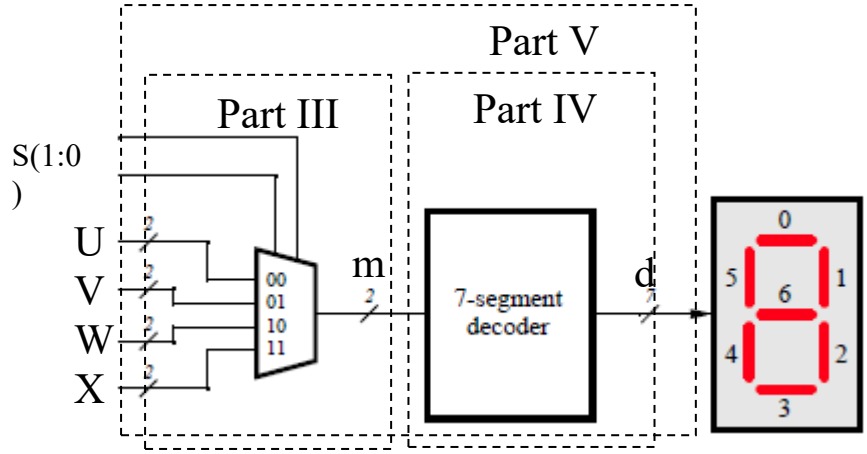


Figure 7. A circuit that can select and display one of five characters.

Discuss as a group what this does. Suppose U = 00, V = 01, W = 10, X = 11:

S = 00; what is m? what is d?
S = 01; what is m? what is d?
S = 10; what is m? what is d?
S = 11; what is m? what is d?

Discuss as a group what this does. Now suppose U = 01, V = 10, W = 11, X = 00:

S = 00; what is m? what is d?
S = 01; what is m? what is d?
S = 10; what is m? what is d?
S = 11; what is m? what is d?

(be ready to answer this to your demonstrator, also be able to explain why and what happens with different inputs)

1. Include your Verilog files for part III, part IV (including relevant subfiles), and the template for part V: **part5.v**
2. Complete the template testbench: **part5_TB.v.**
   Your testbench should test all the above conditions, and also sequences with your own (different) values for U, V, W, X, S

Complete the template and get a lab demonstrator to check your understanding of all of these sections and mark completion

Part VI – Rotating Display

You are now to extend this code so that it uses four 7-segment displays rather than just one. (See end of week 4 lecture if this helps). The purpose of your circuit is to initially display any word on the four displays that is composed of the characters in Table 1 (or you can modify this to your own word), and be able to rotate this word in a circular fashion depending on S(1:0).

As an example, U=00, V=01, W=10, X=11, then the chosen characters are d E 1 blank. Then depending on S(1:0), you should be able to produce the following:

S(1:0)	Character pattern			
00	0000111	1110000	0011100	1100011
01	1110000	0011100	1100011	0000111
10	1110000	0011100	1100011	0000111
11	1100011	0000111	1110000	0011100

Table 2

Create a new Quartus project for your circuit.

1. Create a hand-drawing of how to implement this circuit. You will need 4 instances of **part5.v**, and appropriate wiring.
2. Include: your Verilog files for **part5.v**, and all relevant subfiles (you should know what they are), and the template **part6.v**
3. Complete the template PartVI.v and the template testbench: testbench_partVI.v
   Your testbench should test what will happen for different inputs of S, U, V, W, X. You must be able to explain what you expect should happened to a demonstrator

You must be able to explain your code, drawing and testbench to lab demonstrators