ECEN 2350: Digital Logic

Assignment #4

1. [5 points.] Download the Verilog file from Canvas `lab4-q1.v` that implements a 4-bit adder, and simulate it in EDA Playground. Your output should look something like:

```
0: a=0000, b=xxxx, s=xxxx, co=0
1: a=0000, b=0000, s=0000, co=0
2: a=0000, b=0001, s=0001, co=0
3: a=0000, b=0010, s=0010, co=0
4: a=0000, b=0011, s=0011, co=0
5: a=0000, b=0100, s=0100, co=0
...
```

   Save the output of your file as `lab4-q1.txt` and include it as part of your submission on Canvas.

   This question does not tax your creativity. It is just making sure that you have a working simulation environment.

2. [10 points.] Download the Verilog file from Canvas `lab4-q2.v`, that provides a testbench for this part.

   In this part, you will make **4** simple Verilog modules that implement basic bit-wise operations. You can use EDA playground to create and test these modules.

   (a) The first module will be the `and2`, which will input two 1-bit inputs (`a`, and `b`) and output a single 1-bit output (`c`), which is only 1 if both the inputs are 1, otherwise it is 0 (bitwise AND).

   (b) The second module will be named `or2`, and will also input two 1-bit inputs (`a`, and `b`) and output a single 1-bit output (`c`). In this module, you'll implement OR, so that the output is only a 1 if at least one of the inputs is a 1.

(c) The third module will be named `xor2`, and will input two 1-bit inputs (`a`, and `b`) and output a single 1-bit output (`c`). This module will implement XOR, which will output a 1 if the two inputs are different. For instance, when $a = 1$ and $b = 0$, `c` would be 1.

(d) The last module will be named `majority`, and will input three 1-bit inputs (`a`, `b`, and `c`) and output a single 1-bit output (`d`). The output `d` will be the majority of the input three bits, meaning that if at least 2 of the 3 inputs are 1, it will output 1. For instance, $a = 0$, $b = 1$, and $c = 0$ should output 0.

It may help to write out a truth table for this function before implementing this module.

Running your design with the provided test bench should produce an output that ends with `All correct!`. If it does not, there is likely an error in one of your modules that you'll need to fix.

3. [10 points.] Write a verilog module named `selector2` that implements a 2-bit selector. The selector should take three 1-bit inputs: `a`, `b`, and `s`. Your module should output a single bit, `out`. If `s` is 0, `out` should be `a`. If `s` is 1, `out` should be `b`. Thus, the module *selects* between outputing `a` or `b`, based on `s`. Your module may not use `if` statements, and should only use bit operations (e.g. `|, &, ~`).

   It may help to start by writing a truth table for all the combinations of `a`, `b`, and `s`, and then writing verilog code. Test your code using the `lab4-q3.v` code. Your output should not output any `ERROR` lines.