0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Again, you should be able to follow the steps in section 3.2 to test the adder on hardware. Currently the `ml505top.v` file does not have an instantiation for the `FA.v` module so you will have to do it yourself. The instantiation should be very similar to the `Mux2_1.v` module. Make sure to read the comments in the code for port specifications.

3.5 Extending the Design

Now that we've finished a single full adder, we're going to try something a little less boring and create a ripple adder. In the lab directory, you should also find an `Adder.v` module where you are to instantiate the full adders you wrote earlier in the lab. Disregard the `BehavioralAdder.v` file for now; we're only going to deal with structural Verilog for this lab.

We want you to create an 8 bit ripple adder which means that your circuit will contain 8 instantiations of the full adder module you wrote earlier. Instead of manually writing out 8 full adder instantiations, Verilog contains a `generate` structure that makes life easier by allowing you to generate multiple copies of a module.

To use the generate statement we also need a parameter which is defined for you in the `Adder.v` file.

The generate statement and parameters are declared as follows:

```
module foo(
     input [N-1:0] x;
     output [N-1:0] y;
);
     // declaration of the parameter
     parameter N = <default value>;
```

```
    // the parameter can be used to specify wire widths
    wire [N-1:0] z;
    // variable to be used in generate statement
    genvar i;
    // declaration of a generate statement
    generate for( i = 0; i < N; i = i + 1 )
        begin:<unique_name>
        /* Here, module baz has a parameter named width*/
                baz #(.width(N))
                   b(    .a(x[i]),
                         .b(y[i]),
                         .c(z[i]));
        end
    endgenerate
endmodule
```

You job is to use the `generate` statement to instantiate a full adder and complete the `Adder.v` module. If you are confused about the generate construct, ask your TA.

When you complete the `Adder.v` module, run the usual `make clean, make, make impact` sequence. When you finish programming the board, if you wrote your Adder.v file correctly, the GPIO LED 5 should light up. Otherwise you will have to fix your adder.

4 Checkoff

For checkoff, you will show a TA your implementation on hardware as well as your answers to the checkoff questions.

Please have the check off questions open in a text editor or written on paper (they are not collected so there is no need to print if you type).

Checkoff Requirements:

1. Show the adder and mux working on the board.
2. Show the 3 Verilog files you modified.
3. How many logic gates did your adder require?
4. Show your implementation of the Adder.v file and run `make impact` on the board for your TA.