

**Introduction**

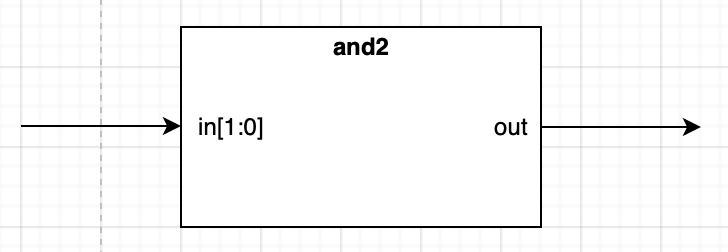
The purpose of this lab is to learn about the Vivado interface. This includes learning about adding files to our project manager, writing and understanding verilog, as well as learning how to create modules. Once we learned how to compile the project, the final task was to compare the waveform generated through the use of modules with a waveform with simple boolean algebra. The boolean algebra was our testbench case for the modules.

**Design Methodology**

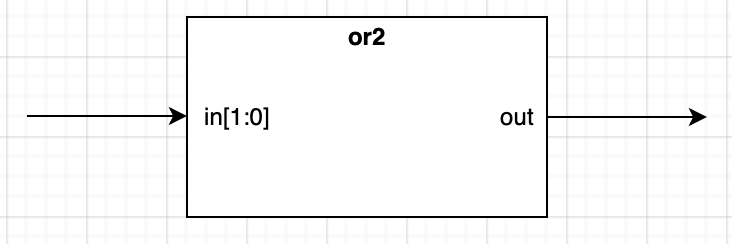
In this lab, I will be creating AND and OR modules and integrating these modules into a larger one. The purpose of this is to learn about how Verilog works and the functions it is able to perform. An AND\_OR function will be created to combine the lower level modules into a single higher level module.

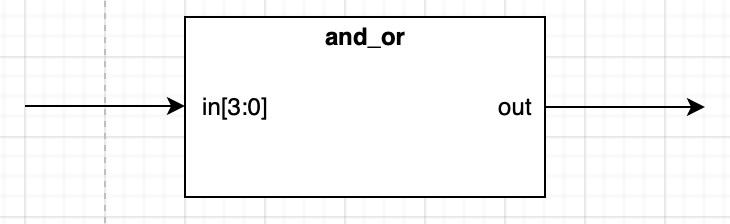
*Table 1:* Module Table

|  |  |
| --- | --- |
| Module | Function |
| AND\_OR | Consists of two AND modules and an OR module. The AND\_OR module takes 4 inputs which it distributes to two AND modules, and it takes the output from these two AND modules and feeds it into a final OR module which will produce a single output. |
| AND2 | Takes two inputs, performs a boolean algebra AND function on the inputs, and produces a single output. |
| OR2 | Takes two inputs, performs a boolean algebra OR function on the inputs, and produces a single output. |

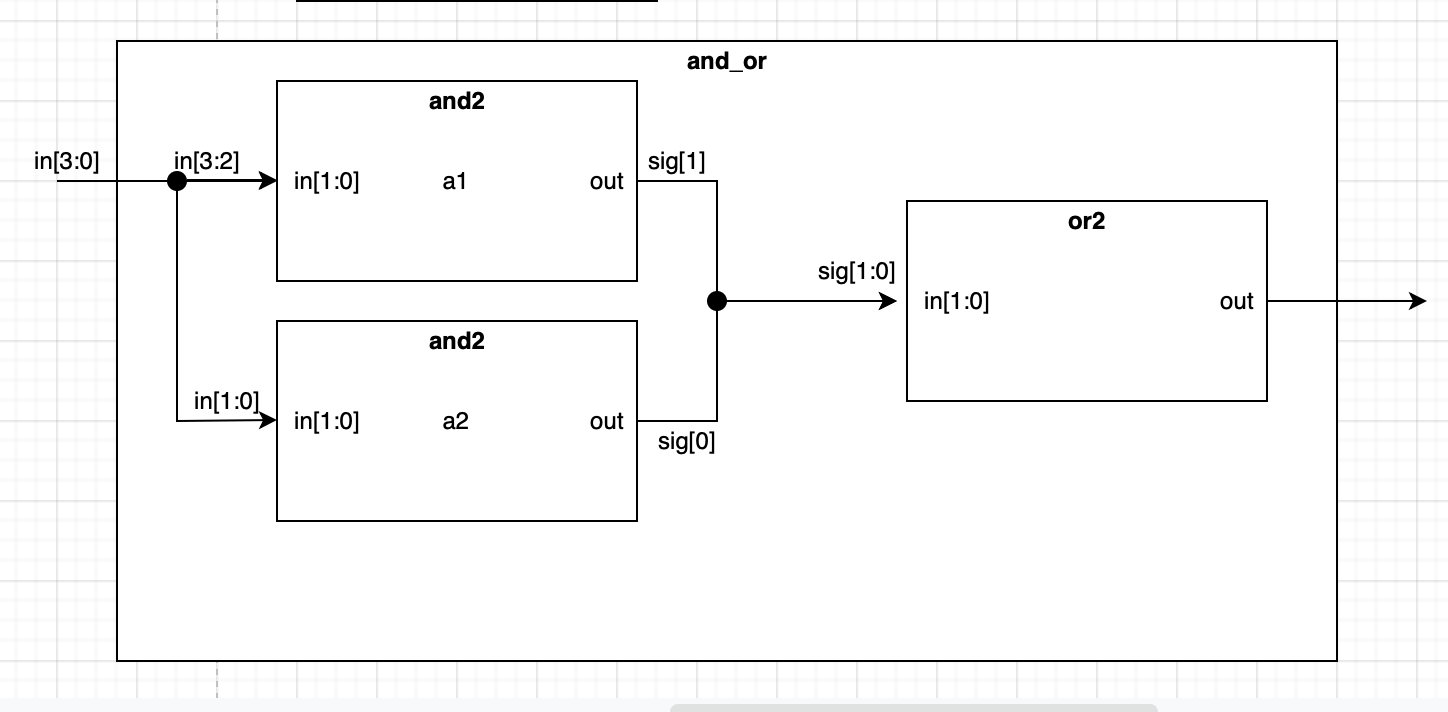
The figures below will show the block diagrams of the modules in Table 1.

*Figure 1:* Block Diagram of AND2 module

*Figure 2:* Block Diagram of OR2 module



*Figure 3:* Block Diagram of AND\_OR module



*Figure 4:* Block Diagram Interior of AND\_OR module

**Simulation Results**

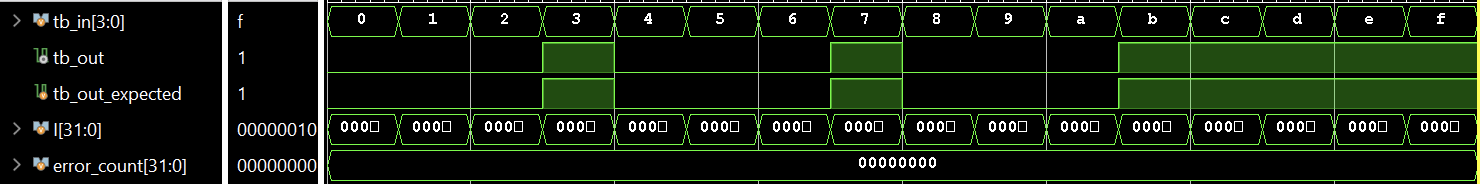
To test the module, we used an input of 0000 through 1111, as shown in figure 5 below. The output from the module is represented as tb\_out and the input is represented as tb\_in[3:0].



*Figure 5:* AND\_OR Waveform

Anytime that there are two 1’s in the first half or the second half of the binary input, the output will be a logic high. For example, a 1100 will produce a logic high, but a 0110 will not because the 1’s are not in the same half of the binary number.

To verify that our waveform is true, a simulation waveform was created to compare the results by creating a file that directly contained the boolean algebra for the AND\_OR module. The comparison is shown in figure 6 below. As seen, there are no errors detected.



*Figure 5:* Expected waveform comparison

**Conclusion**

Overall, we learned how to create modules in the Vivado design suite through the use of verilog. The purpose of these modules is to allow for reusability of a certain function, which makes implementation of certain procedures much simpler. In this lab, we had a single task: successfully simulate the waveform for the AND\_OR module. This task was successfully completed (A). Also, I learned how to verify the validity of a waveform through the use of a testbench file which contained the boolean algebra version of the modules. This was labeled as the tb\_and\_or\_self\_checking.v file. In the upcoming labs, we will learn about FPGA validation as well as generating a bitstream to load onto our FPGA board.

**Appendix**

*Table 2:* Source Code

|  |
| --- |
| and\_or.v |
| module and\_or(  input wire [3:0] in,  output wire out  );  wire [1:0] sig;    and2 A1 (  .in (in[3:2]),  .out (sig[1])  );    and2 A2 (  .in (in[1:0]),  .out (sig[0])  );    or2 O1 (  .in (sig),  .out (out)  );    endmodule |

|  |
| --- |
| and2.v |
| module and2(  input wire [1:0] in,  output wire out  );    assign out = in[1] & in[0];      endmodule |

|  |
| --- |
| or2.v |
| module or2(  input wire [1:0] in,  output wire out  );    assign out = in[1] | in[0];      endmodule |

|  |
| --- |
| tb\_and\_or.v |
| module tb\_and\_or;  reg [3:0] tb\_in;  wire tb\_out;    and\_or DUT (  .in (tb\_in),  .out (tb\_out)  );    initial begin  tb\_in = 4'b0000; #5; tb\_in = 4'b0001; #5;  tb\_in = 4'b0010; #5; tb\_in = 4'b0011; #5;  tb\_in = 4'b0100; #5; tb\_in = 4'b0101; #5;  tb\_in = 4'b0110; #5; tb\_in = 4'b0111; #5;  tb\_in = 4'b1000; #5; tb\_in = 4'b1001; #5;  tb\_in = 4'b1010; #5; tb\_in = 4'b1011; #5;  tb\_in = 4'b1100; #5; tb\_in = 4'b1101; #5;  tb\_in = 4'b1110; #5; tb\_in = 4'b1111; #5;  $display ("Simulation Finished");  $finish;  end  endmodule |

|  |
| --- |
| tb\_and\_or\_self\_checking.v |
| module tb\_and\_or\_self\_checking;  reg [3:0] tb\_in;  wire tb\_out;    reg tb\_out\_expected;    and\_or DUT (  .in (tb\_in),  .out (tb\_out)  );    integer I;  integer error\_count = 0;    initial begin  for(I = 0; I < 2 \*\* 4; I = I + 1) begin  tb\_in = I;  tb\_out\_expected = (tb\_in[3] & tb\_in[2]) | (tb\_in[1] & tb\_in[0]);  #5;  if(tb\_out\_expected != tb\_out) begin  $display("Error, result incorrect %d != %d", tb\_out\_expected, tb\_out);  error\_count = error\_count + 1;  end  end  $display ("Simulation Finished");  $finish;  end  endmodule |