Lab 2: Full Adder

Author: Andy Lazcano, 016824184

Date Published: 08/18/2024

I - Introduction

Adders and subsequently subtractors are fundamental components in digital circuits, essential for arithmetic operations in computers. These circuits form arithmetic and logic units (ALUs) in processors, enabling binary addition and subtraction. Adders perform binary addition and calculate the sum of two binary numbers, while subtractors perform the opposite. To explore the design and functionality of these circuits, I implemented a simple full adder with two input components at the gate level.

II- Design

I began with analyzing the truth table provided and determined the correct outputs for a device that will take two inputs and a spare carry in to determine the sum and carry output of the logical device.

The following table determined K-map.

A	B	Cin	Sum (S)	Carry Out (Cout)
0	0	0	0	0
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

Table 1: Full adder truth table. The carryout column is highlighted due to being a deliverable request and provided for ease of use.

The sum output can be expressed as the following Boolean function. I utilized this table to construct a K-map and determined the function.

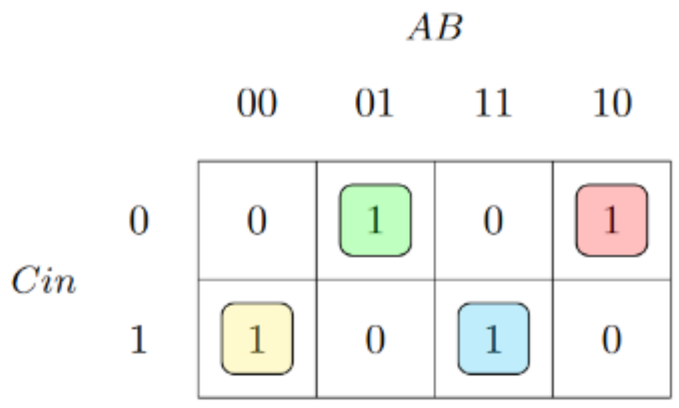


Figure 1: K-map for summing circuit.

$$S = A'B'Cin + A'B'Cin + ABCin + A'BCin'$$

I reduced the expression and algebraically determined the reduced form:

$$S = A \oplus B \oplus Cin$$

This represents the summing circuit and will be used to construct the design of the adder in Verilog. To represent the Carry circuit, I mapped the Carry outputs to another function and created a Boolean representation for Verilog implementation. I performed the same process.

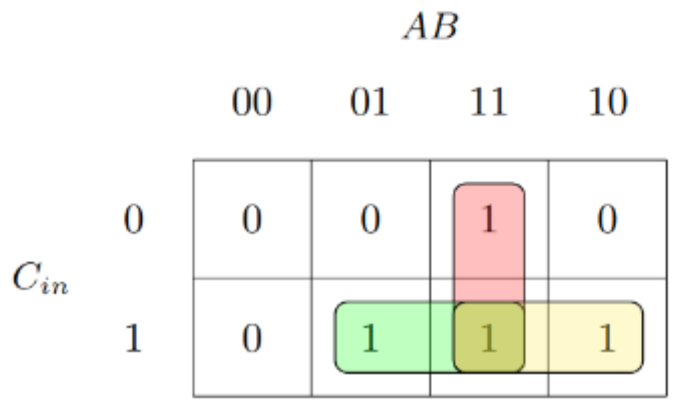


Figure 2: K-map for carry out.

The derived expression is:

$$Cout = AB + BCin + ACin.$$

I minimized this equation for the sake of maintaining logical equivalence and to minimize gates. The equivalent equation I derived was

$$Cout = AB + Cin(A + B)$$

And can be sketched informally as the following.

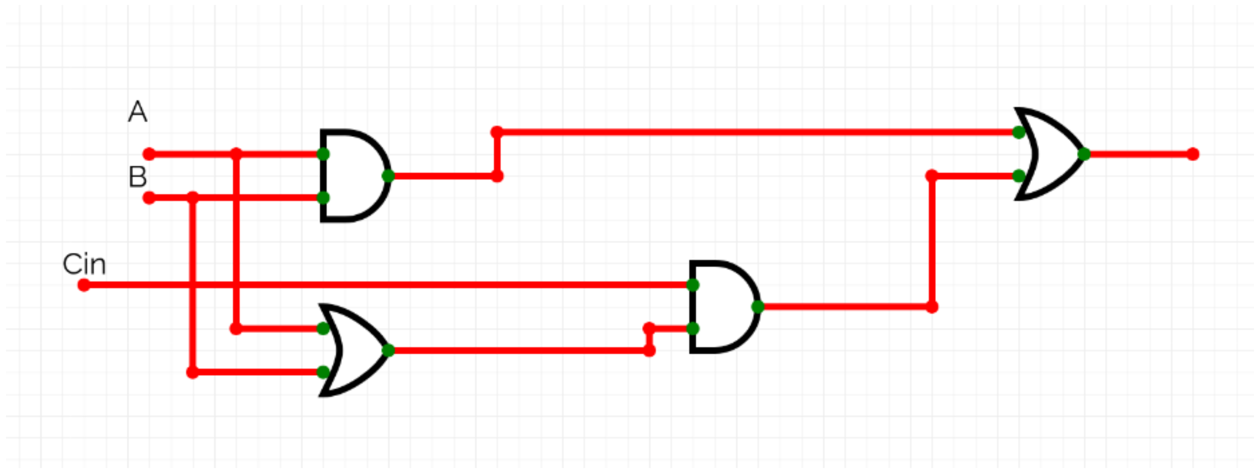
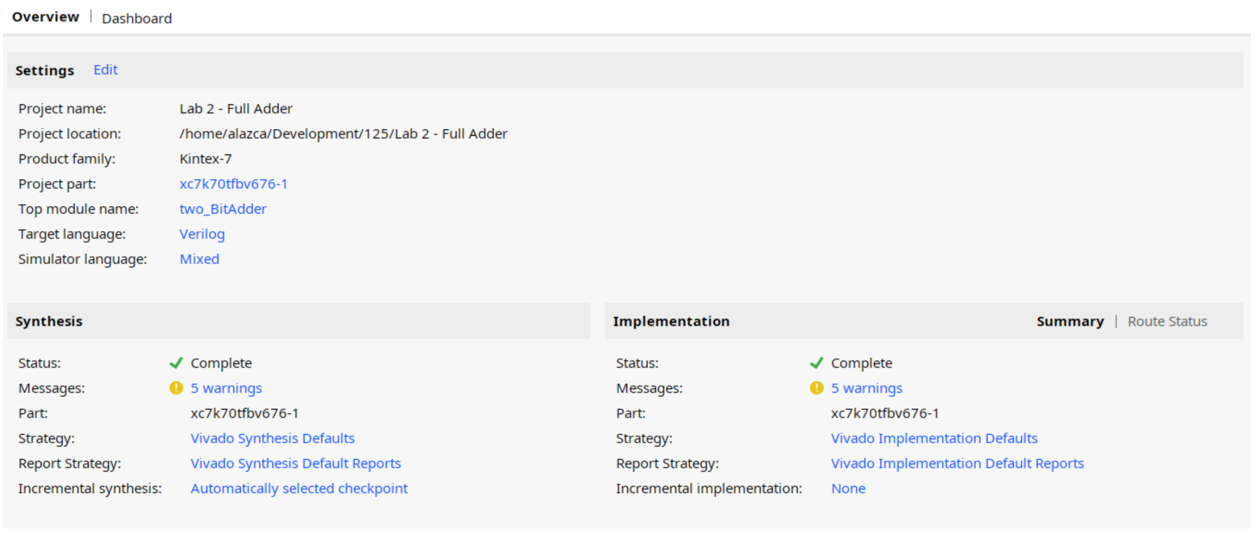


Figure 3: Simple one line diagram to represent the carry out function in minimal gates.

I then proceeded to generate hardware simulation and coded the adder and tested using Vivado.

III - Code and Testbench



Settings

Project name:	Lab 2 - Full Adder
Project location:	/home/alazca/Development/125/Lab 2 - Full Adder
Product family:	Kintex-7
Project part:	xc7k70tfbv676-1
Top module name:	two_BitAdder
Target language:	Verilog
Simulator language:	Mixed

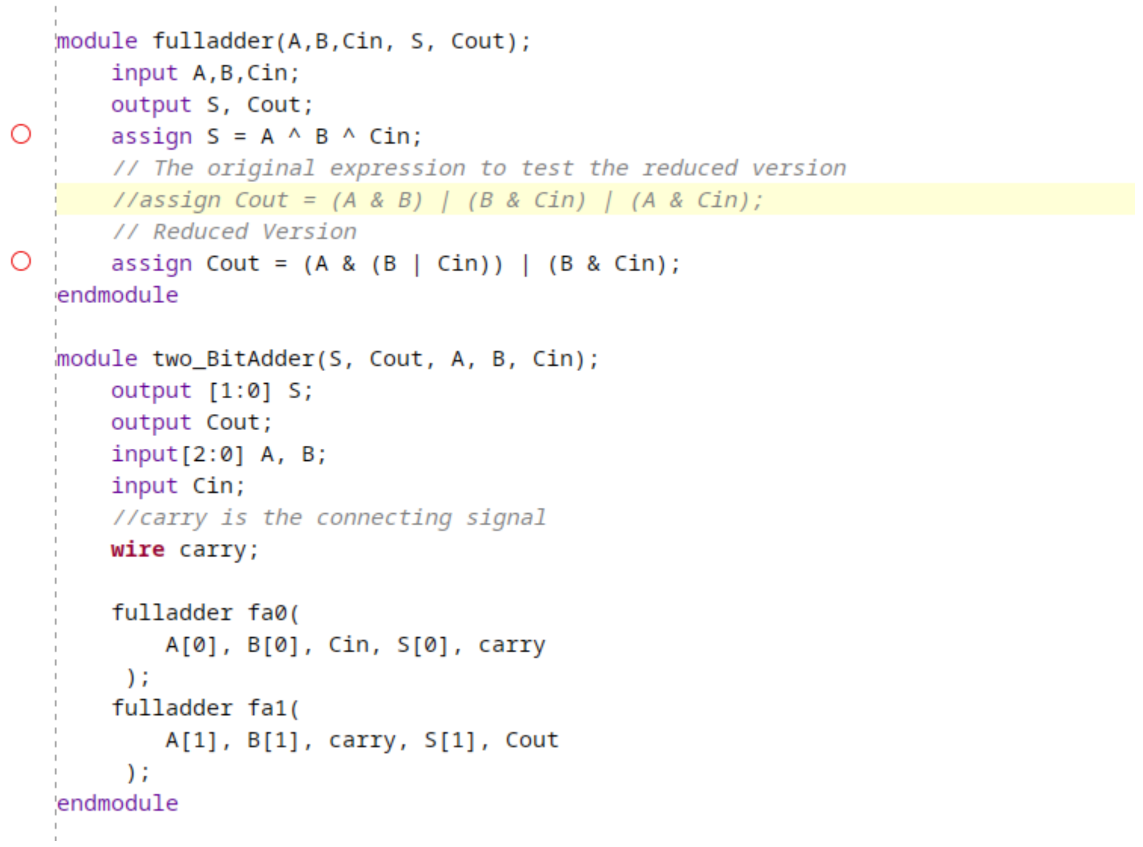
Synthesis

Status:	✓ Complete
Messages:	5 warnings
Part:	xc7k70tfbv676-1
Strategy:	Vivado Synthesis Defaults
Report Strategy:	Vivado Synthesis Default Reports
Incremental synthesis:	Automatically selected checkpoint

Implementation

Status:	✓ Complete
Messages:	5 warnings
Part:	xc7k70tfbv676-1
Strategy:	Vivado Implementation Defaults
Report Strategy:	Vivado Implementation Default Reports
Incremental implementation:	None

Figure 4: Summary of proper implementation of the entire Full adder component.

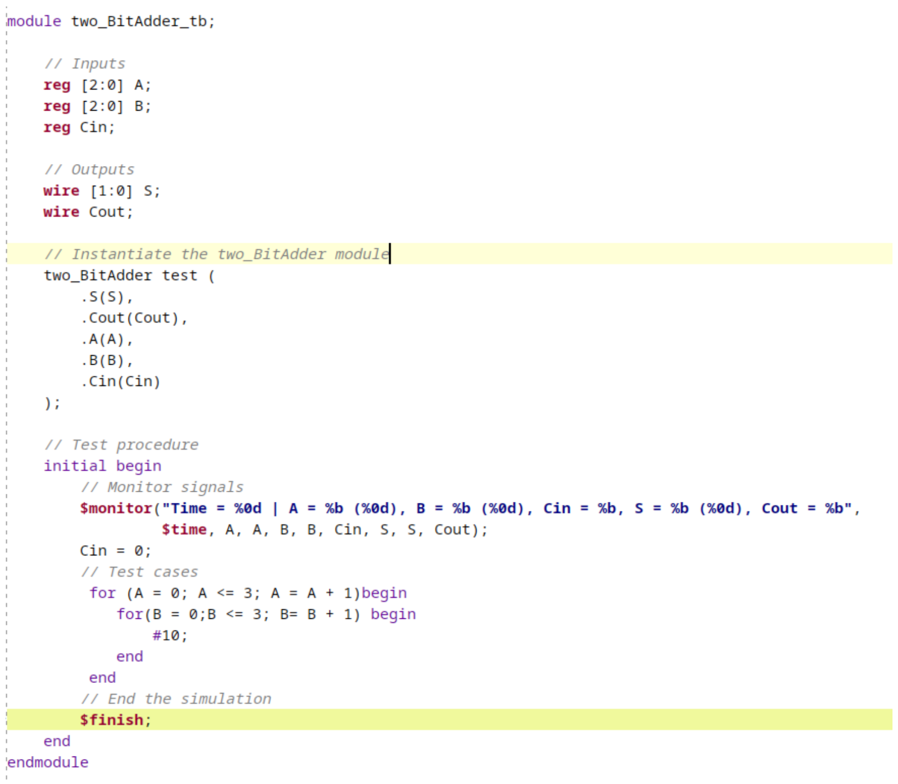


```
module fulladder(A,B,Cin, S, Cout);
    input A,B,Cin;
    output S, Cout;
    assign S = A ^ B ^ Cin;
    // The original expression to test the reduced version
    //assign Cout = (A & B) | (B & Cin) | (A & Cin);
    // Reduced Version
    assign Cout = (A & (B | Cin)) | (B & Cin);
endmodule

module two_BitAdder(S, Cout, A, B, Cin);
    output [1:0] S;
    output Cout;
    input[2:0] A, B;
    input Cin;
    //carry is the connecting signal
    wire carry;

    fulladder fa0(
        A[0], B[0], Cin, S[0], carry
     );
    fulladder fa1(
        A[1], B[1], carry, S[1], Cout
     );
endmodule
```

Figure 5: Screenshot of the full adder module. I tested both the original expression and the reduced version to ensure they produced the same results while minimizing gates.



```
module two_BitAdder_tb;

    // Inputs
    reg [2:0] A;
    reg [2:0] B;
    reg Cin;

    // Outputs
    wire [1:0] S;
    wire Cout;

    // Instantiate the two_BitAdder module
    two_BitAdder test (
        .S(S),
        .Cout(Cout),
        .A(A),
        .B(B),
        .Cin(Cin)
    );

    // Test procedure
    initial begin
        // Monitor signals
        $monitor("Time = %0d | A = %b (%0d), B = %b (%0d), Cin = %b, S = %b (%0d), Cout = %b",
                 $time, A, A, B, B, Cin, S, S, Cout);
        Cin = 0;
        // Test cases
         for (A = 0; A <= 3; A = A + 1)begin
            for(B = 0;B <= 3; B= B + 1) begin
                #10;
            end
         end
        // End the simulation
        $finish;
    end
endmodule
```

Figure 6: Screenshot of the test bench used. Implemented a loop to iterate through the possible values of a two bit input.

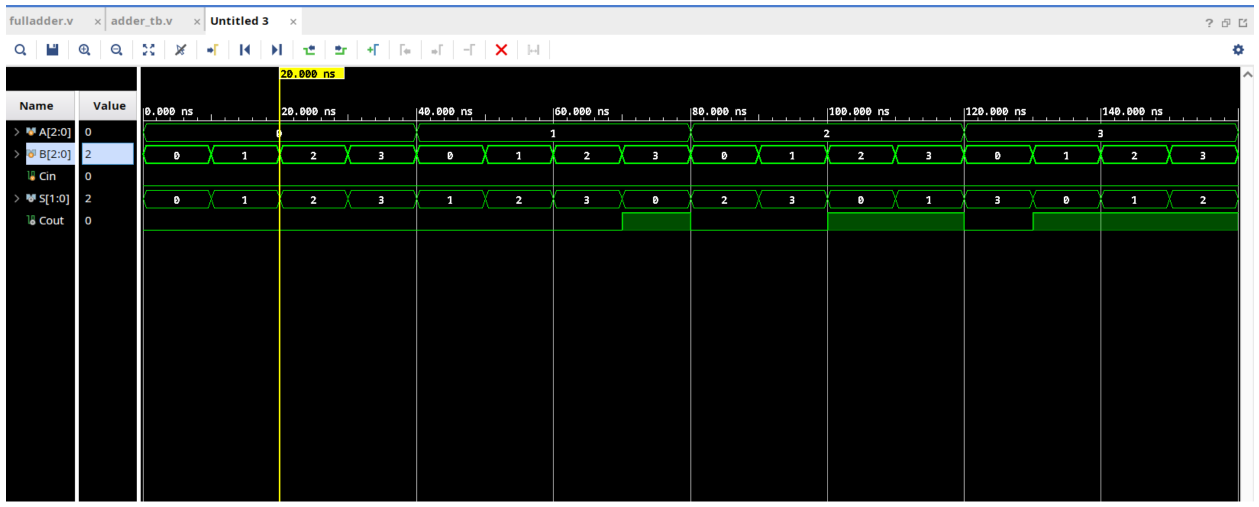
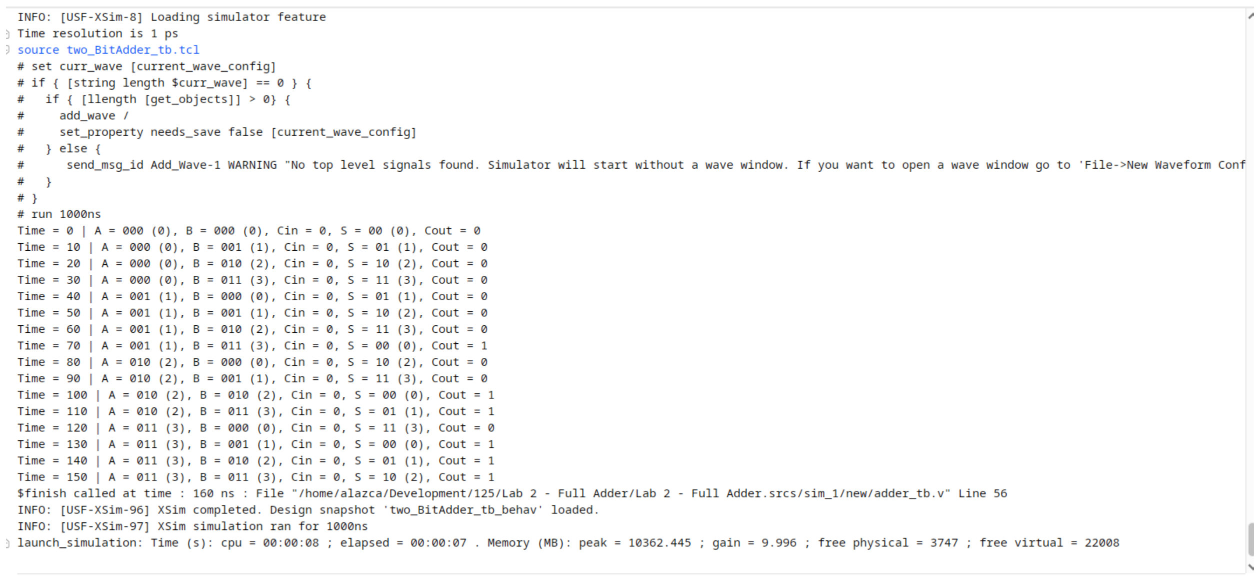


Figure 7: Screenshot of the final waveform. Input A is incremented by each third iteration of the two bit integer.



```
INFO: [USF-XSim-8] Loading simulator feature
Time resolution is 1 ps
source two_BitAdder_tb.tcl
# set curr_wave [current_wave_config]
# if { [string length $curr_wave] == 0 } {
#   if { [llength [get_objects]] > 0} {
#     add_wave /
#     set_property needs_save false [current_wave_config]
#   } else {
#      send_msg_id Add_Wave-1 WARNING "No top level signals found. Simulator will start without a wave window. If you want to open a wave window go to 'File->New Waveform Conf
#   }
# }
# run 1000ns
Time = 0 | A = 000 (0), B = 000 (0), Cin = 0, S = 00 (0), Cout = 0
Time = 10 | A = 000 (0), B = 001 (1), Cin = 0, S = 01 (1), Cout = 0
Time = 20 | A = 000 (0), B = 010 (2), Cin = 0, S = 10 (2), Cout = 0
Time = 30 | A = 000 (0), B = 011 (3), Cin = 0, S = 11 (3), Cout = 0
Time = 40 | A = 001 (1), B = 000 (0), Cin = 0, S = 01 (1), Cout = 0
Time = 50 | A = 001 (1), B = 001 (1), Cin = 0, S = 10 (2), Cout = 0
Time = 60 | A = 001 (1), B = 010 (2), Cin = 0, S = 11 (3), Cout = 0
Time = 70 | A = 001 (1), B = 011 (3), Cin = 0, S = 00 (0), Cout = 1
Time = 80 | A = 010 (2), B = 000 (0), Cin = 0, S = 10 (2), Cout = 0
Time = 90 | A = 010 (2), B = 001 (1), Cin = 0, S = 11 (3), Cout = 0
Time = 100 | A = 010 (2), B = 010 (2), Cin = 0, S = 00 (0), Cout = 1
Time = 110 | A = 010 (2), B = 011 (3), Cin = 0, S = 01 (1), Cout = 1
Time = 120 | A = 011 (3), B = 000 (0), Cin = 0, S = 11 (3), Cout = 0
Time = 130 | A = 011 (3), B = 001 (1), Cin = 0, S = 00 (0), Cout = 1
Time = 140 | A = 011 (3), B = 010 (2), Cin = 0, S = 01 (1), Cout = 1
Time = 150 | A = 011 (3), B = 011 (3), Cin = 0, S = 10 (2), Cout = 1
$finish called at time : 160 ns : File "/home/alazca/Development/125/Lab 2 - Full Adder/Lab 2 - Full Adder.srcs/sim_1/new/adder_tb.v" Line 56
INFO: [USF-XSim-96] XSim completed. Design snapshot 'two_BitAdder_tb_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
launch_simulation: Time (s): cpu = 00:00:08 ; elapsed = 00:00:07 . Memory (MB): peak = 10362.445 ; gain = 9.996 ; free physical = 3747 ; free virtual = 22008
```

Figure 8: Screenshot of the console output indicating the time and output observance at each increment of 10ns.

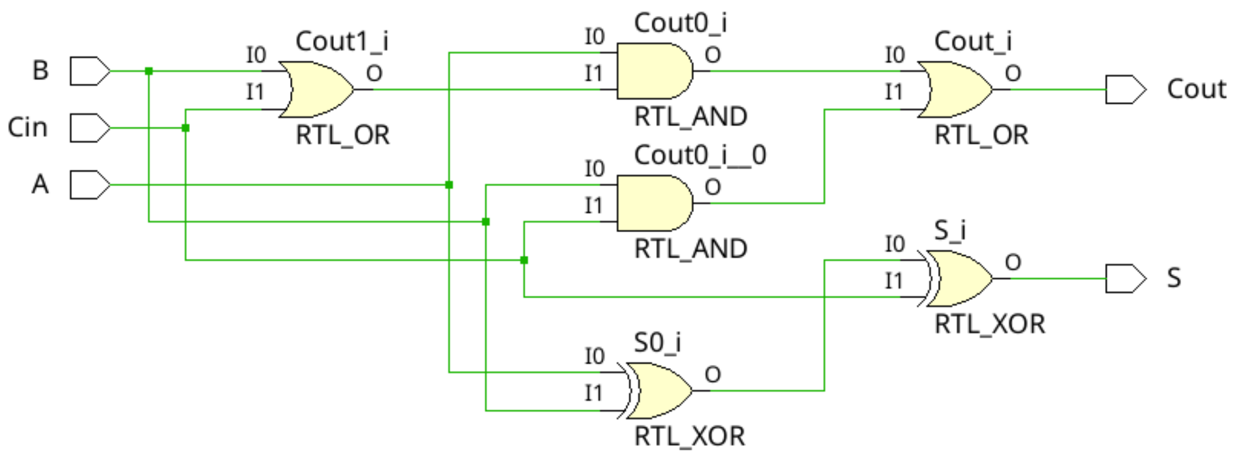


FIgure 9: Schematic of the single full adder module.

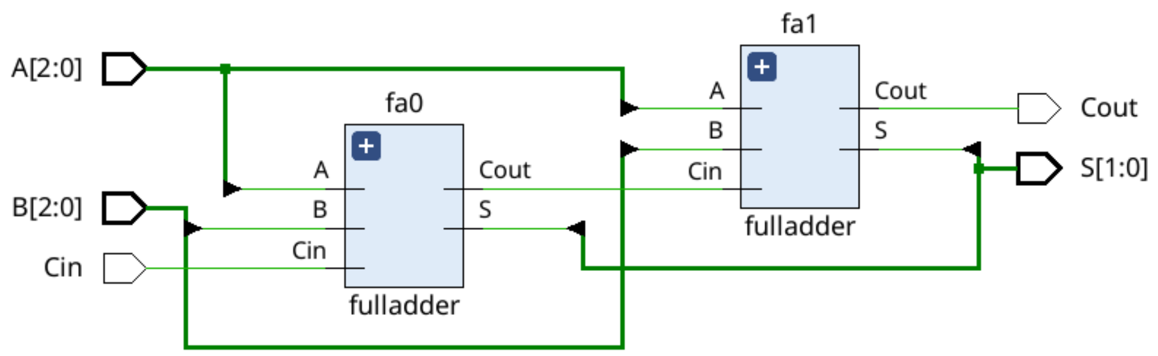


Figure 10: Schematic of two full adder modules representing a two bit adder.

IV - Summary

I implemented a full adder in a data flow methodology. I verified the reduced version from the canonical form for both Summing and Carryout circuits. I simulated signals for two bit adders and recorded the schematic. Simulation was successful. The schematic was slightly different from the original concept.