Lab 3
16-Bit Full Adder

Group G
Chang, Philbert (013179257)
Hua, Russell (013184015)
Thai, Paul (014760252)

ECE 3300L.01
Professor Aly

07/05/2021

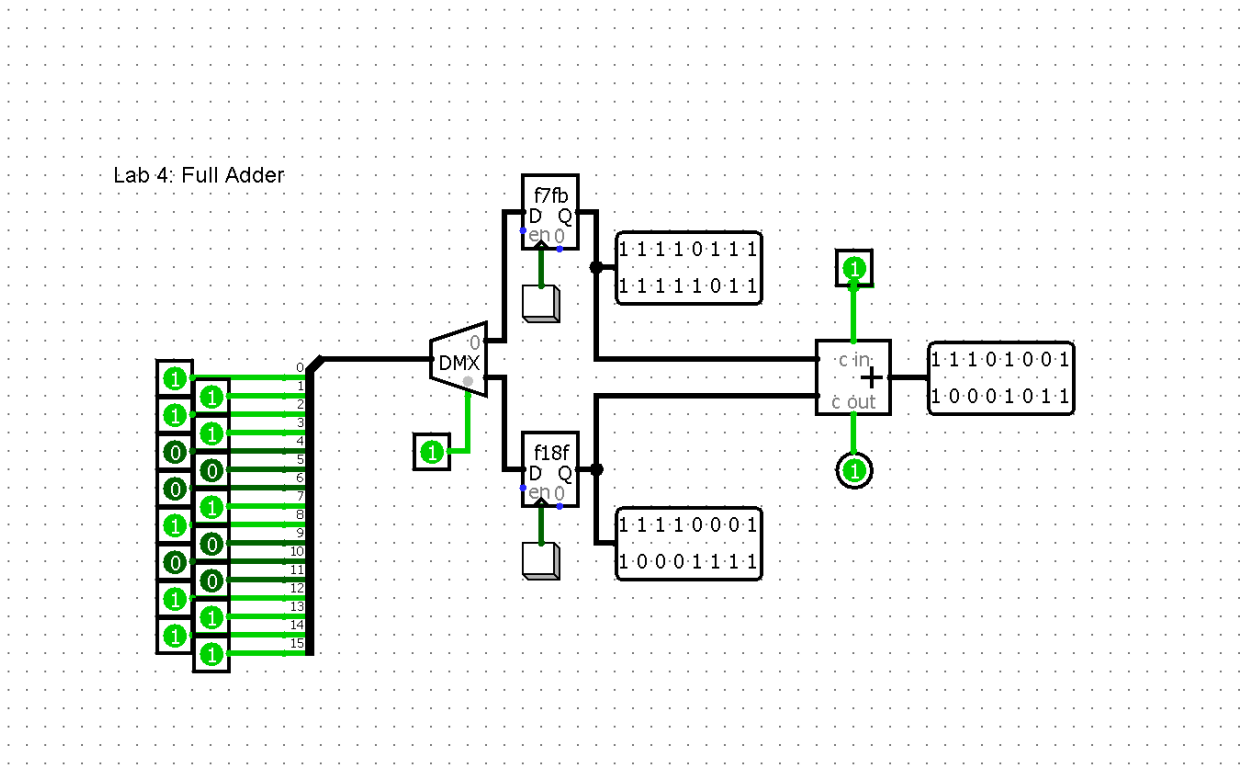
Problem

Create a 16-bit full adder, complete with carry in and carry out bits using 16 switches and 2 buttons on the FPGA.

Code Detail

The foundation of this 16-bit full adder program is a 1-bit full adder module created using structural modeling. The demultiplexer is created using behavioral modeling and the two buttons are programmed to control either select or carry in. The 16-bit full adder is created by using a loop that iterates 16 times and assigns the appropriate outputs of the current adder to the next adder. Clicking the select button enables the second input to be selected in the switches, thus creating a full adder.

Schematic



Results

```
23  module full_adder (
24      input X, input Y, input Cin,
25      output Sum, output Cout
26  );
27      wire w0,w1,w2;
28
29      xor xor0(w0,X,Y);
30      xor xor1(Sum,w0,Cin);
31      and and0(w1,X,Y);
32      and and1(w2,w0,Cin);
33      or   or0(Cout,w2,w1);
34
35  endmodule
```

Figure 1: Full Adder Code

```
23  module full_adder_16 #(parameter SIZE = 16)(
24      input CInput, input [SIZE - 1:0] InputA_n, input [SIZE - 1:0] InputB_n, output [SIZE - 1:0] SumOutput_n, output COutput_n
25      );
26
27
28      wire [SIZE:0] w;
29      assign w[0] = CInput;
30
31      genvar i;
32
33      for (i = 0; i<SIZE; i = i + 1)
34      begin
35          full_adder FA_IT (.X(InputA_n[i]), .Y(InputB_n[i]), .Cin(w[i]), .Sum(SumOutput_n[i]), .Cout(w[i+1]));
36      end
37      assign COutput_n = w[SIZE];
38  endmodule
```

Figure 2: 16-Bit Full Adder Code

```
23  module dmux(
24      input dsel, input [15:0] in, output reg [15:0] out0, output reg [15:0] out1
25      );
26      always @(dsel or in)
27          begin
28          case (dsel)
29              0: out0 = in;
30              1: out1 = in;
31          endcase
32      end
33  endmodule
```

Figure 3: Demultiplexer Code

```
22  module button(
23      input clk,
24      output reg led
25      );
26      initial begin
27          led = 1'b0;
28      end
29      always @ (posedge clk)
30          if (clk)
31              led <= ~led;
32  endmodule
```

Figure 4: Button Code

```
module FA_merge(
            input [15:0] In,
            input Sel,
            input CIn,
            output [15:0] SumOut,
            output COut_16,
            output CIn_Status
            );

    wire [15:0] temp0;
    wire [15:0] temp1;
    wire bSel;
    wire bIn;

    button b (.clk(Sel), .led(bSel));
    button c (.clk(CIn), .led(bIn));

    dmux I(.dsel(bSel), .in(In), .out0(temp0), .out1(temp1));
    assign CIn_Status = bIn;
    full_adder_16 J (.CInput(bIn), .InputA_n(temp0), .InputB_n(temp1), .SumOutput_n(SumOut), .COutput_n(COut_16));

endmodule
```

Figure 5: Working 16-Bit Full Adder with Select and C-in Button Code

```
module FA_merge_tb();

    reg [15:0] In_tb;
    reg Sel_tb;
    reg CIn_tb;

    wire [15:0] SumOut_tb;
    wire COut_tb;
    wire CIn_Status_tb;
    FA_merge it_works(.In(In_tb), .Sel(Sel_tb), .CIn(CIn_tb), .SumOut(SumOut_tb), .COut_16(COut_tb), .CIn_Status(CIn_Status_tb));

    initial begin

        Sel_tb = 0;
        In_tb = 16'd0;
        #10
        Sel_tb = 1;
        In_tb = 16'd1;
        CIn_tb = 1;
        // CIN is 1
        #10

        #10
        Sel_tb = 0;
        In_tb = 16'd10;
        #10
        Sel_tb = 1;
        In_tb = 16'd1;
        CIn_tb = 0;
        // CIN is 1

```

Figure 6: Full Adder Testbench Code [P1]

```
54          #10
55          Sel_tb = 0;
56          In_tb = 16'd6000;
57          #10
58          Sel_tb = 1;
59          CIn_tb = 1;
60          In_tb = 16'd1;
61          // CIN is 0
62
63          #10
64          Sel_tb = 0;
65          In_tb = 16'd0;
66          #10
67          Sel_tb = 1;
68          In_tb = 16'd1;
69          CIn_tb = 0;
70          Sel_tb = 0;
71          // CIN is 0
72          #100
73              $finish;
74
75      end
76
77  endmodule
```

Figure 7: Full Adder Testbench Code [P2]

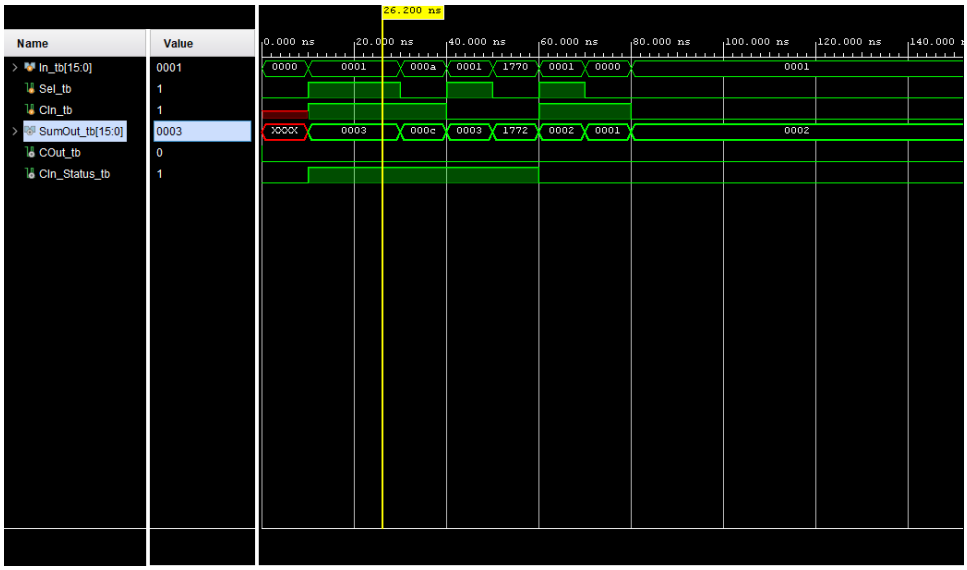


Figure 8: Full Adder Behavioral Simulation

Name	Slice LUTs (32600)	Slice Registers (65200)	Slice (8150)	LUT as Logic (32600)	Bonded IOB (210)	BUFGCTRL (32)
FA_merge	26	34	25	26	36	3
b (button)	1	1	1	1	0	1
c (button_0)	1	1	1	1	0	0
I (dmux)	24	32	23	24	0	0

Figure 9: Resource Utilization

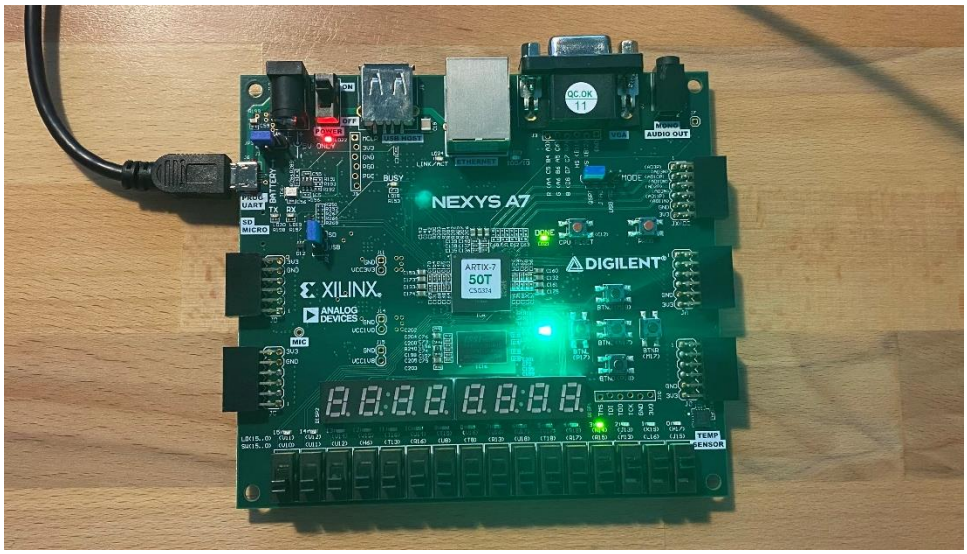


Figure 10: Full Adder with Cin and First 16-bit Input

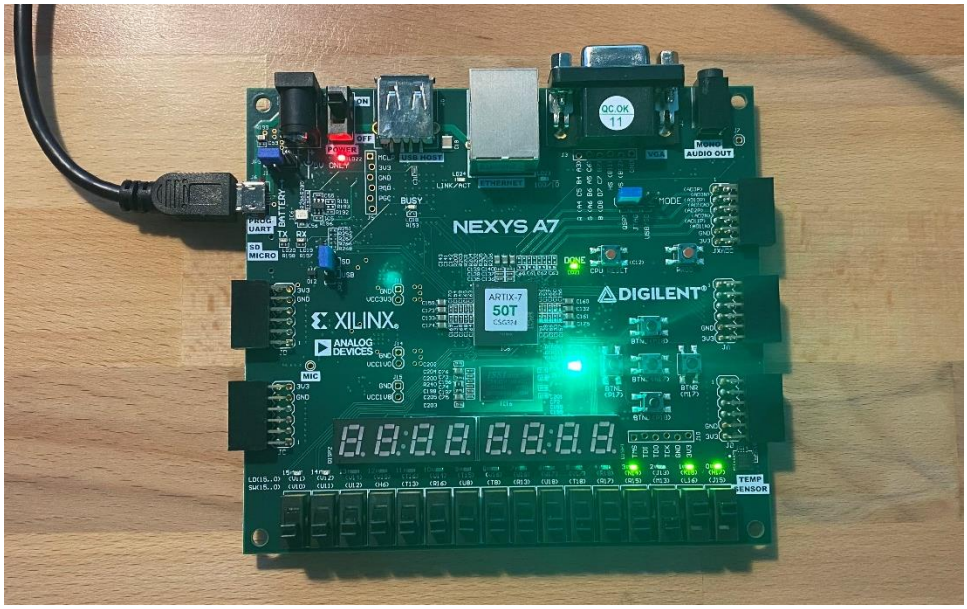


Figure 11: Full Adder with Cin and Second 16-bit Input (After Pressing Sel Sw)

Discussion

The behavioral simulation of the full adder worked as expected, although it was hard to test different combinations as all the buttons and inputs had to be predefined. The implementation of the program on the FPGA worked flawlessly, except for multiple inputs from the buttons due to missing a debouncing circuit. Most of the time, pressing once on the Cin button turns it on and pressing again turns it off. The result of the full adder shown in the LEDs are also updated immediately. The carry out LED also worked without a problem.

Vivado also displayed warnings about suboptimal routing of the button. We aren't sure why this occurs – perhaps this is related to using an input – button in our case - as a clock (inverting the state of output). The warning was ignored by adding a line to ignore the error in the constraints file and didn't seem to greatly affect the implementation of the 16-bit adder.

Conclusion

Overall, the implementation of the program worked as expected. Our program benefited from using multiple types of modeling, as structural modeling was easier for a full adder, and behavioral was easier for a demultiplexer. To improve the implementation, using some debouncing methods, such as creating a circuit or delaying accepting button presses that are too fast would make the inputs easier to use. Additionally, it's possible that using a T-flip-flop would fix the routing warning/error because the functionality is different from a clock.

Work Distribution

Russell: Worked on code and testbench.

Philbert: Worked on report and demo.

Paul: Worked on code and testbench.