

ENGR 378 Digital Systems Design

Exp. 2: Arithmetic Circuit, Behavioral

Submitted by:

Björn Franzén
Kristian Lauszus

Sept. 11, 2014

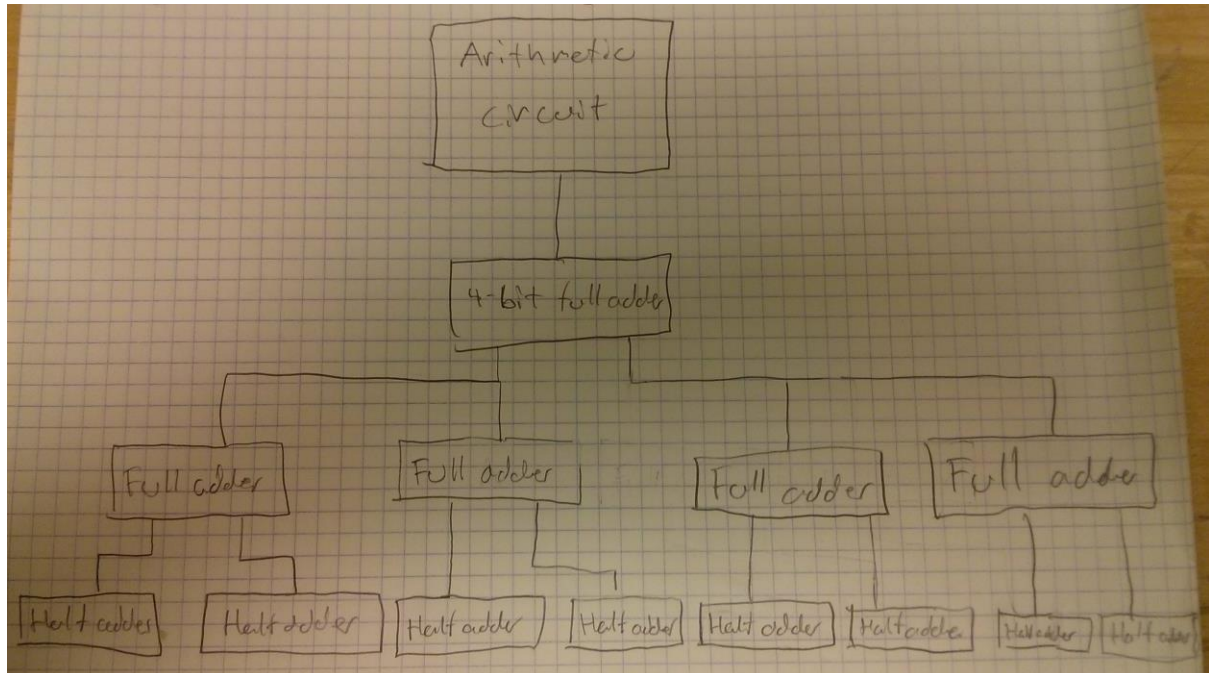
Demonstration.....	____/5
Report: Presentation	____/2
Problem analysis.....	____/2
Design work.....	____/5
Results.....	____/5
Conclusion/Discussion.....	____/1
Total.....	____/ 20

Problem Analysis

We used the truth table stated in the report and realized that it could be implemented with one four-bit adder with its input tied to a mux, implementing different functions according to the table.

Hardware Design

We used the building blocks as in the prelab and broke down the 4-bit adder into smaller blocks according to the block diagram below.



Verilog Modeling

We described the different building blocks as outlined here.

Half-adder

The half-adder is implemented directly from the truth table for it.

```
module half_adder(  
    output cout, sum,  
    input a,b  
);  
    xor M1 (sum, a, b);  
    and M2 (cout, a, b);  
endmodule
```

Full-adder

By cascading two half-adders and with some combinational logic on the output, the following code was generated.

```
module full_adder(  
    output cout, sum,  
    input a,b, cin  
);  
    wire w1, w2, w3;  
  
    half_adder M1(w2, w1, a, b);  
    half_adder M2(w3, sum, cin, w1);  
    or  
        M3(cout, w2, w3);  
endmodule
```

4-bit Adder

Following the same pattern as with the full-adder, this 4-bit version was implemented by cascading 4 full-adders.

```
module full_adder_4bit(  
    output cout,  
    output[3:0] sum,  
    input[3:0] a, b,  
    input cin  
);  
    wire cin1, cin2, cin3;  
  
    full_adder M1(cin1, sum[0], a[0], b[0], cin);  
    full_adder M2(cin2, sum[1], a[1], b[1], cin1);  
    full_adder M3(cin3, sum[2], a[2], b[2], cin2);  
    full_adder M4(cout, sum[3], a[3], b[3], cin3);  
endmodule
```

Mux

The truth table outlined in the lab assignment, as well as the prelab report, was implemented with the following mux. Its outputs are tied to the 4-bit full adder and the mux's inputs have different values for a and b, according to the truth table.

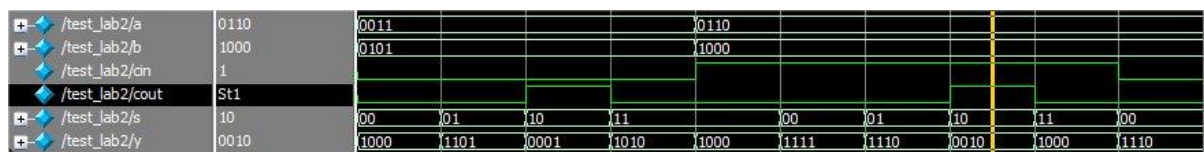
```
module Lab2(  
    output[3:0] F,  
    output cout,  
    input [3:0] a,b,  
    input [1:0] s,  
    input cin  
);  
    wire[3:0] y0, y1;  
  
    assign y0 = !s[1] ? a : !s[0] ? ~a : 0;  
    assign y1 = s[0] ? ~b : b;  
  
    full_adder_4bit M1(cout, F, y0, y1, cin);  
endmodule
```

Results/Verification

To verify our design we implemented a test bench which can be found under “HDL Source Code”. Our approach was basically to test 2 randomly chosen inputs for a, b and Cin with all the values for select. Figure 1 shows the output from our test bench. At the point where the cursor is at, for an example, should yield the output ‘A+B+Cin which we verified by summing the bits according to Table 1. This was done for all input values.

‘A	1001
+B	1000
+Cin	1
Sum	10010

Table 1- Summation of 'A+B+Cin.



/test_lab2/a	0110	0011					0110							
/test_lab2/b	1000	0101					1000							
/test_lab2/cin	1													
/test_lab2/cout	St1													
/test_lab2/s	10	00	01	10	11		00	01	10	11	00			
/test_lab2/y	0010	1000	1101	0001	1010	1000	1111	1110	0010	1000	1110			

Figure 1 - Results from the simulation.

Conclusions and Discussion

We didn't have any difficulties with implementing the design in Verilog, especially since the adder was already outlined in the course literature. The real challenge was to get modelsim to recognizing our project structure, especially the simulation source code.

Below is a high-level solution of the assignment which is obviously far more efficient in terms development time and readability, as we let the synthesizer take care of implementing the needed adders instead of creating them manually.

```
module Lab2(  
    output [3:0] y,  
    output cout,  
    input [3:0] a,b,  
    input [1:0] s,  
    input cin,  
  
    );  
    assign {cout,y} = (s==2'b00) ? (a+b+cin) :  
                    (s==2'b01) ? (a+(~{1'b1,b})+cin) :  
                    (s==2'b10) ? ((~{1'b1,a})+b+cin) :  
                    (s==2'b11) ? ((~{1'b1,b})+cin) :  
                                0;  
endmodule
```

Work Breakdown

As the lab was fairly easy we didn't divide the work amongst us so it was a joint effort.

Prelabs

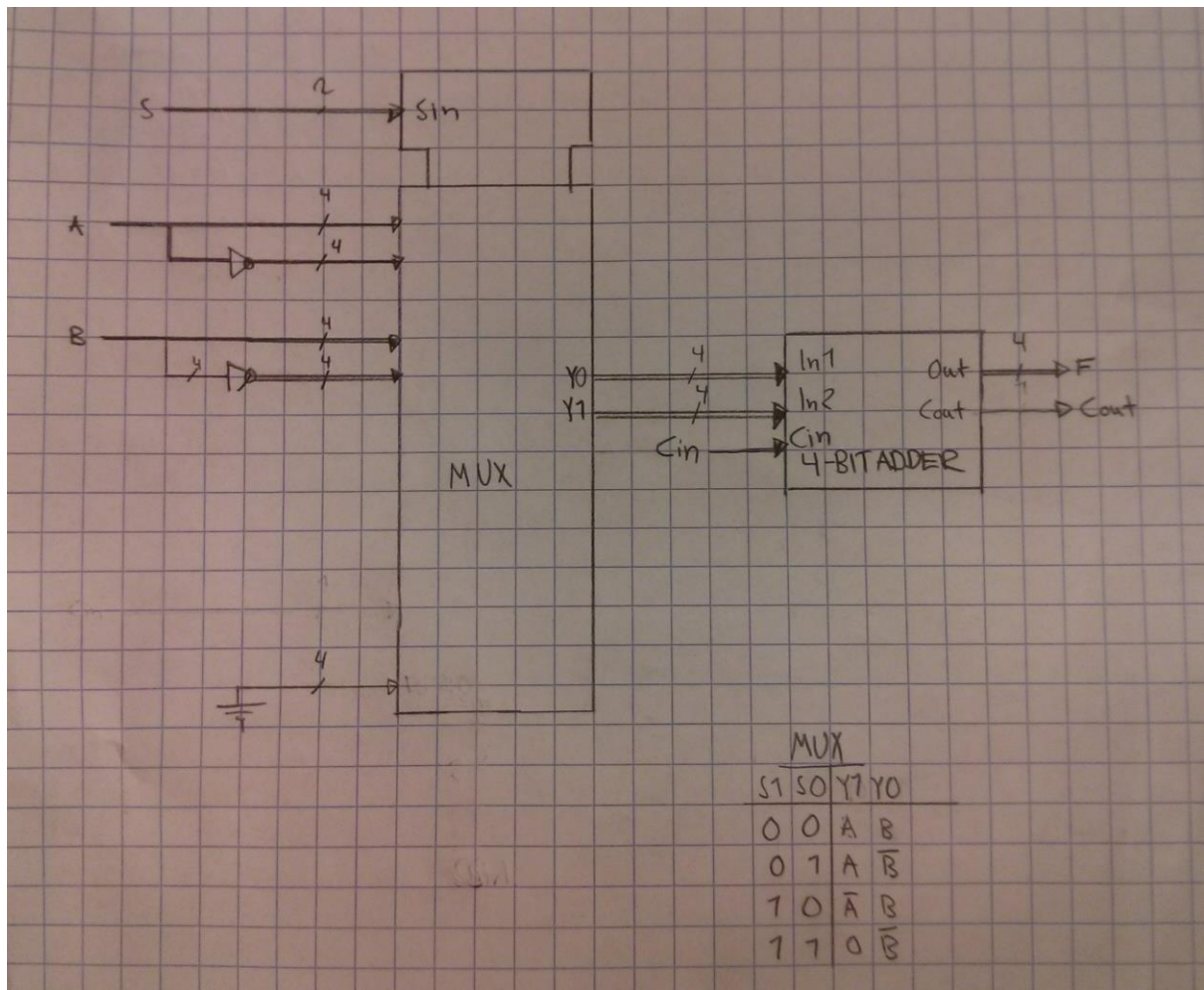


Figure 2 – Our prelab report.

HDL Source Code

```
module half_adder(
    output cout, sum,
    input a,b
);
    xor M1 (sum, a, b);
    and M2 (cout, a, b);
endmodule

module full_adder(
    output cout, sum,
    input a,b, cin
);
    wire w1, w2, w3;

    half_adder M1(w2, w1, a, b);
    half_adder M2(w3, sum, cin, w1);
    or      M3(cout, w2, w3);
endmodule

module full_adder_4bit(
    output cout,
    output[3:0] sum,
    input[3:0] a, b,
    input cin
);
    wire cin1, cin2, cin3;

    full_adder M1(cin1, sum[0], a[0], b[0], cin);
    full_adder M2(cin2, sum[1], a[1], b[1], cin1);
    full_adder M3(cin3, sum[2], a[2], b[2], cin2);
    full_adder M4(cout, sum[3], a[3], b[3], cin3);
endmodule

module Lab2(
    output[3:0] F,
    output cout,
    input [3:0] a,b,
    input [1:0] s,
    input cin
);
    wire[3:0] y0, y1;

    assign y0 = !s[1] ? a : !s[0] ? ~a : 0;
    assign y1 = s[0] ? ~b : b;

    full_adder_4bit M1(cout, F, y0, y1, cin);
endmodule

module test_lab2;
    wire [3:0] y;
    wire cout;
    reg [3:0] a,b;
    reg [1:0] s;
    reg cin;

    Lab2 M1(y,cout,a,b,s,cin);

    initial begin
        a = 0; b = 0; s = 0; cin = 0;
    end

    always begin
        #100 $stop;
    end
end
```

```

always begin
    a = 3; b = 5; cin = 0;
    s=0;
    #10 s=1;
    #10 s=2;
    #10 s=3;

    #10 a = 6; b = 8; cin = 1;
    #10 s=0;
    #10 s=1;
    #10 s=2;
    #10 s=3;
end
endmodule

```

Alternative HDL Source Code

```

module Lab2(
    output [3:0] y,
    output cout,
    input [3:0] a,b,
    input [1:0] s,
    input cin,

);
    assign {cout,y} = (s==2'b00) ? (a+b+cin) :
                      (s==2'b01) ? (a+(~{1'b1,b})+cin) :
                      (s==2'b10) ? ((~{1'b1,a})+b+cin) :
                      (s==2'b11) ? ((~{1'b1,b})+cin) :
                                0;
endmodule

```