Connor Cousineau

2/19/20

ECE 3700

LAB 2: Adders

The objective of this lab is to design a Ripple carry adder and a Look-Ahead Carry Adder. Then to determine if the LACA is in fact faster than the RCA. To do this I will be designing a Full Adder first. To do these I will be using continuous assign statements and the equations found in the FA section of the Code section of this document. A summary of the inputs are as follows. If both inputs are on, then a carry is produced. If A or B and Carry in are on, then a carry is also produced. If A or B or Carry in are on then a sum is produced.

This unit is only capable of producing 1-bit operations. A simple test bench was run to check if the outputs were in fact simple additions. Reaching the conclusion that the FA does work allowed me to move on to the 4-bit Ripple Carry Adder. See Figure 2 for the design. By combining the 4 FA's, the unit is now capable of doing 4-bit math. 0-31 max. Using the test bench, I was able to conclude that this unit was working as intended. The next step was to grab the data which can be found in Figure 4. This unit contains 8 LE's according to the synthesis report. Due to its relatively small size compiling doesn't take much time at all. The longest route of this unit is Cin to Cout at 10.202. This is not the 9.5, but the timings can vary from unit to unit. The last step was to put it on the FPGA and watch the unit do what its supposed to. Flipping the switches on the board caused the respective bit LED's to change based on the input. Experiment one was complete.

The next unit to create was the Look-Ahead Carry Adder. The idea behind this unit is to not have to rely on the previous carry values. To do this, we derived in class how do this with math equations containing only propagates, generates and Carry in. The propagates and generates are represented by the combination of A and B respectively. How I did this can be found in the LACA section of the Code section. The Carry in portion was derived to be decrementing generated followed by incrementing propagates. The last term was all the propagates and Carry in. By assigning all of these terms to continuous assign statements I was able to produce the LACA found in Figure 7. Using the same tech bench with slight modification to the Sum bit vector I was able to determine that this LACA was in fact working as intended. There was a slight issue which caused the third carry not to be correct do to misplaced propagate conditions. This caused some of the arithmetic to be very wrong. I.E. 1+3 =12. Upon fixing this issue the unit works as intended. It was put on the FPGA and was demonstrated to be working properly. The last step was to acquire the timings which can be found in Figure 5. The slowest time on my unit was the Cin to S[3] at 9.778.

There is a major downside to this speedup that we can see in the LE increase. This unit used 12 logic elements whereas the RCA only needed 8. That is roughly 33% more space needed to preform the same operation. This cost can be decided on based on the intended purpose of the unit. If Space is more important a person may choose to use the RCA over the LACA. That being said, the LACA in my timings was about 5% faster. At this scale that improvement is almost negligible, but when this goes much bigger that 5% becomes very noticeable. If performance is the only desire of the designer then they

may choose the LACA over the RCA. The overall learning from this is that performance costs space and depending on which is more important will determine the overall look of the design.

The last thing that was needed to complete this lab was to design the 8-bit LACA. Given that we already designed the 4-bit, it was as simple as just expanding the bit vectors A B and S and hooking them up in series. Running a very large for loop we were able to determine that doing this was in fact correct. This unit is able to do 8-bit math. When doing the synthesis, this unit ends up using 30 LE's 19 4 input, 7 3 input, and 4 2 input. When comparing the LE use of the 8-bit to the 4-bit. There is a noticeable increase from 12 to 30. Roughly 1.4 times the amount. The data gathered leads to the idea that as scale increases. The number of LE will increase near exponentially. This could be and error on my board, or bad user inputs which could push towards more linear scaling. Unfortunately, no timing data was taken on this 8-bit unit and no timing conclusions can be made. The for loop and the time needed to process the shear amount of number that can be added by this unit goes to show that as you add bits, the amount of time needed to compute its truth table goes exponential. 2^x.

Overall, the lessons that were learned from this lab are as follows. The ripple carry adder typically will have the longest route from its start to its end. To combat this the concept of calculating the next carry first came to mind. By doing so the unit did in fact gain some speed, but there is the consequence of space. By speeding up the unit the number of LE's went up 33% but gaining speed of 5%. Another thing learned is that designing a small unit and combining the logic in series can save a lot of time. By designing the 4-bit LACA, I was able to bypass the need of designing the 8-bit. Combining the two 4 bits is equivalent in most ways. I was not able to determine if there is a time difference between the two units and designing a one 8-bit unit. The TL;DR of this experiment is that the LACA is fast than the RCA and Larger computation units can be constructed using functional smaller ones.

All of these details can be found in the photos found below. The code is also included at the bottom of this document.

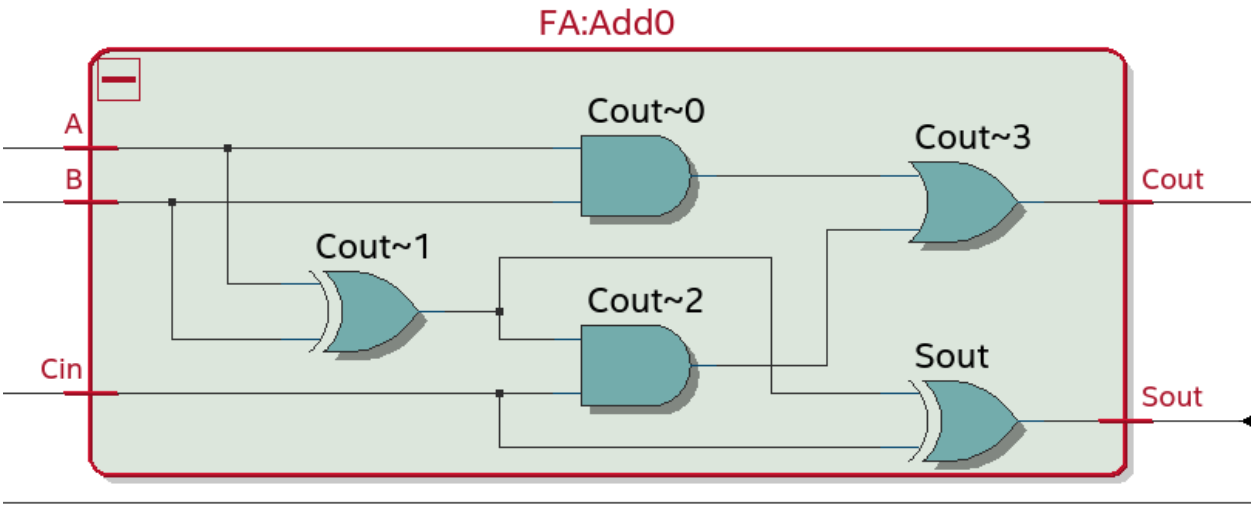

Figure 1. Full Adder

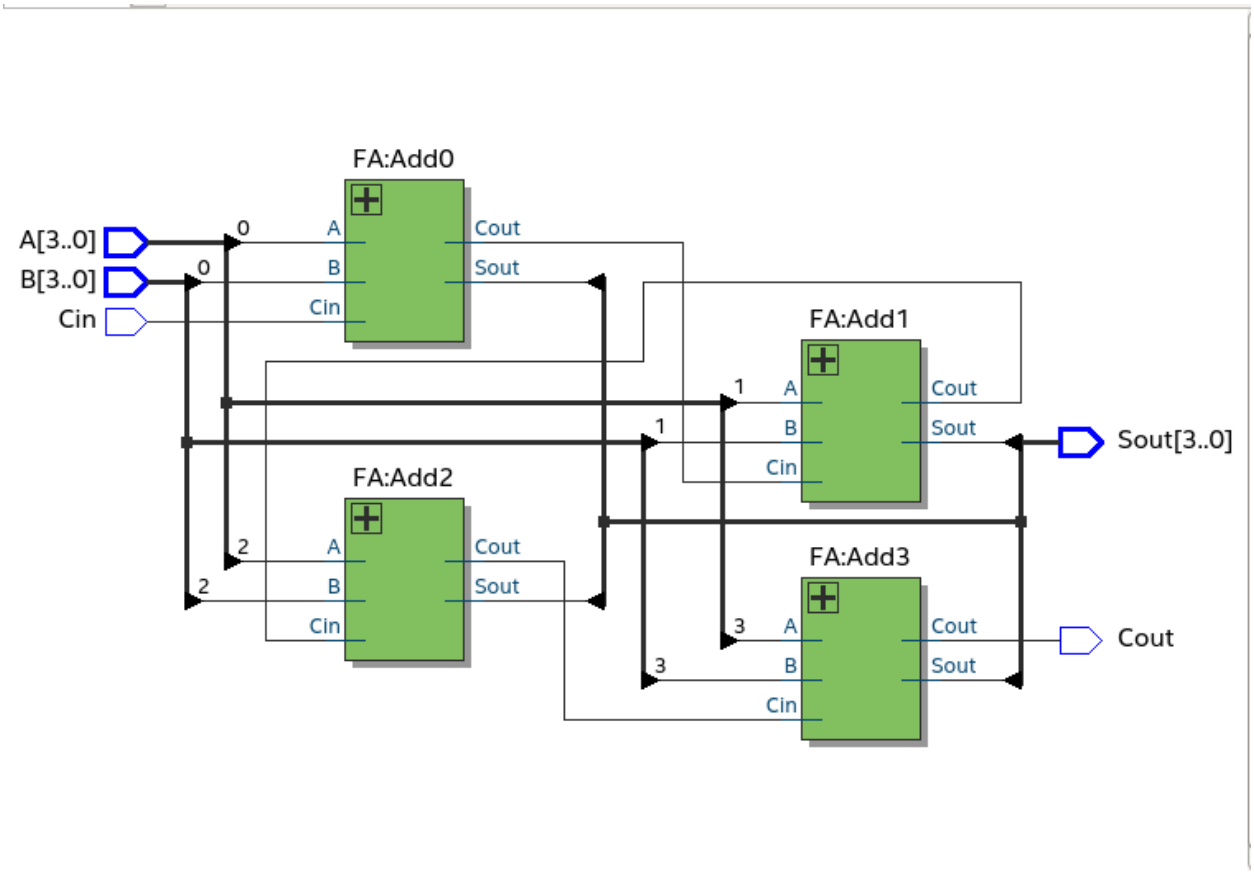

Figure 2. Ripple Carry Adder

	Resource	Usage
1	Estimated Total logic elements	30
2		
3	Total combinational functions	30
4	▼ Logic element usage by number of LUT inputs	
1	-- 4 input functions	19
2	-- 3 input functions	7
3	-- <=2 input functions	4
5		
6	▼ Logic elements by mode	
1	-- normal mode	30
2	-- arithmetic mode	0
7		
8	▼ Total registers	0
1	-- Dedicated logic registers	0
2	-- I/O registers	0
9		
10	I/O pins	26
11		
12	Embedded Multiplier 9-bit elements	0
13		
14	Maximum fan-out node	A[5]~input
15	Maximum fan-out	6
16	Total fan-out	140
17	Average fan-out	1.71

Figure 3. 8bit Resources

	Input Port	Output Port	RR	RF	FR	FF ▲
1	Cin	Cout	10.104			10.202
2	A[0]	Cout	10.065			10.141
3	B[0]	Cout	10.033			10.018
4	Cin	Sout[3]	9.819	9.653	10.087	9.880
5	A[0]	Sout[3]	9.780	9.614	10.026	9.819
6	B[0]	Sout[3]	9.748	9.582	9.903	9.696
7	Cin	Sout[2]	9.431	9.213	9.672	9.445
8	A[0]	Sout[2]	9.392	9.174	9.611	9.384
9	B[0]	Sout[2]	9.360	9.142	9.488	9.261
10	B[1]	Cout	9.131			9.112
11	A[1]	Cout	8.979			8.994
12	B[2]	Cout	8.921			8.909
13	Cin	Sout[1]	8.787	8.631	9.025	8.834
14	A[2]	Cout	8.781			8.793
15	B[1]	Sout[3]	8.846	8.680	8.997	8.790
16	A[0]	Sout[1]	8.748	8.592	8.964	8.773
17	A[1]	Sout[3]	8.694	8.528	8.879	8.672
18	B[0]	Sout[1]	8.716	8.560	8.841	8.650
19	B[2]	Sout[3]	8.636	8.470	8.794	8.587
20	A[2]	Sout[3]	8.496	8.330	8.678	8.471
21	B[1]	Sout[2]	8.458	8.240	8.582	8.355
22	Cin	Sout[0]	8.405	8.219	8.555	8.328
23	A[0]	Sout[0]	8.370	8.193	8.528	8.284
24	A[1]	Sout[2]	8.306	8.088	8.464	8.237
25	B[0]	Sout[0]	8.317	8.093	8.413	8.180
26	B[2]	Sout[2]	8.249	8.031	8.331	8.156
27	A[2]	Sout[2]	8.119	7.937	8.237	8.052
28	B[3]	Cout	8.137			8.032
29	A[3]	Cout	8.037			8.009
30	B[3]	Sout[3]	7.825	7.621	7.945	7.732
31	B[1]	Sout[1]	7.801	7.598	7.924	7.712
32	A[3]	Sout[3]	7.757	7.600	7.925	7.701
33	A[1]	Sout[1]	7.649	7.446	7.755	7.595

Figure 4. Ripple Carry Adder Timer

	Input Port	Output Port	RR	RF	FR	FF
1	Cin	S[3]	9.816	9.568	9.983	9.778
2	B[0]	S[3]	9.759	9.511	9.889	9.684
3	A[0]	S[3]	9.638	9.390	9.824	9.619
4	Cin	Cout	9.569			9.556
5	B[0]	Cout	9.512			9.462
6	A[0]	Cout	9.391			9.397
7	B[1]	S[3]	9.267	9.019	9.386	9.181
8	B[2]	S[3]	9.210	9.000	9.371	9.120
9	B[1]	S[2]	9.113	8.960	9.339	9.119
10	A[1]	S[3]	9.135	8.887	9.295	9.090
11	A[1]	S[2]	8.981	8.828	9.248	9.028
12	Cin	S[1]	9.060	8.888	9.229	9.022
13	B[1]	Cout	9.020			8.959
14	B[2]	Cout	8.942			8.929
15	B[0]	S[1]	9.003	8.831	9.135	8.928
16	A[2]	S[3]	8.991	8.781	9.155	8.904
17	A[1]	Cout	8.888			8.868
18	A[0]	S[1]	8.882	8.710	9.070	8.863
19	Cin	S[2]	8.769	8.616	8.962	8.742
20	A[2]	Cout	8.717			8.711
21	B[0]	S[2]	8.729	8.576	8.869	8.649
22	A[0]	S[2]	8.608	8.455	8.798	8.578
23	Cin	S[0]	8.437	8.268	8.575	8.371
24	B[3]	S[3]	8.391	8.143	8.514	8.257
25	B[0]	S[0]	8.367	8.151	8.473	8.248
26	A[3]	S[3]	8.315	8.114	8.484	8.216
27	A[0]	S[0]	8.246	8.030	8.354	8.181
28	B[3]	Cout	8.143			8.035
29	A[3]	Cout	8.056			8.015
30	B[1]	S[1]	8.080	7.861	8.204	7.976
31	B[2]	S[2]	7.988	7.788	8.075	7.918
32	A[1]	S[1]	7.946	7.727	8.061	7.885
33	A[2]	S[2]	7.853	7.691	7.982	7.782

Figure 5. LACA

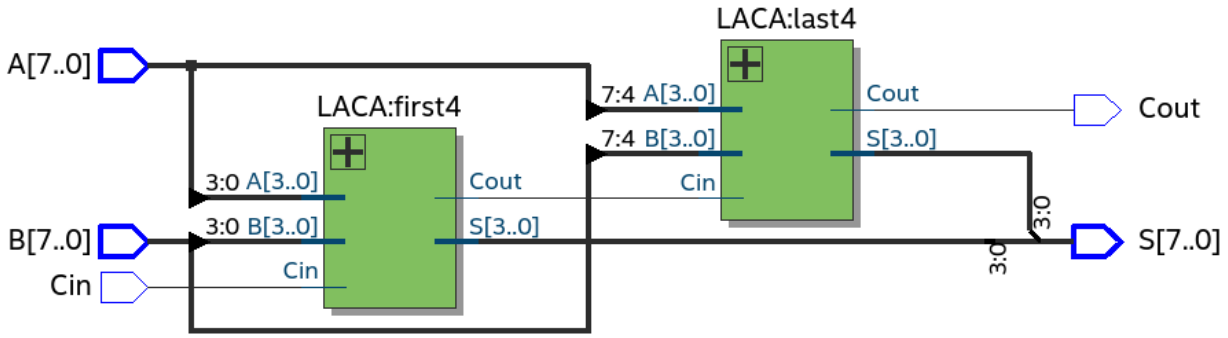

Figure 6. 8 Bit LACA

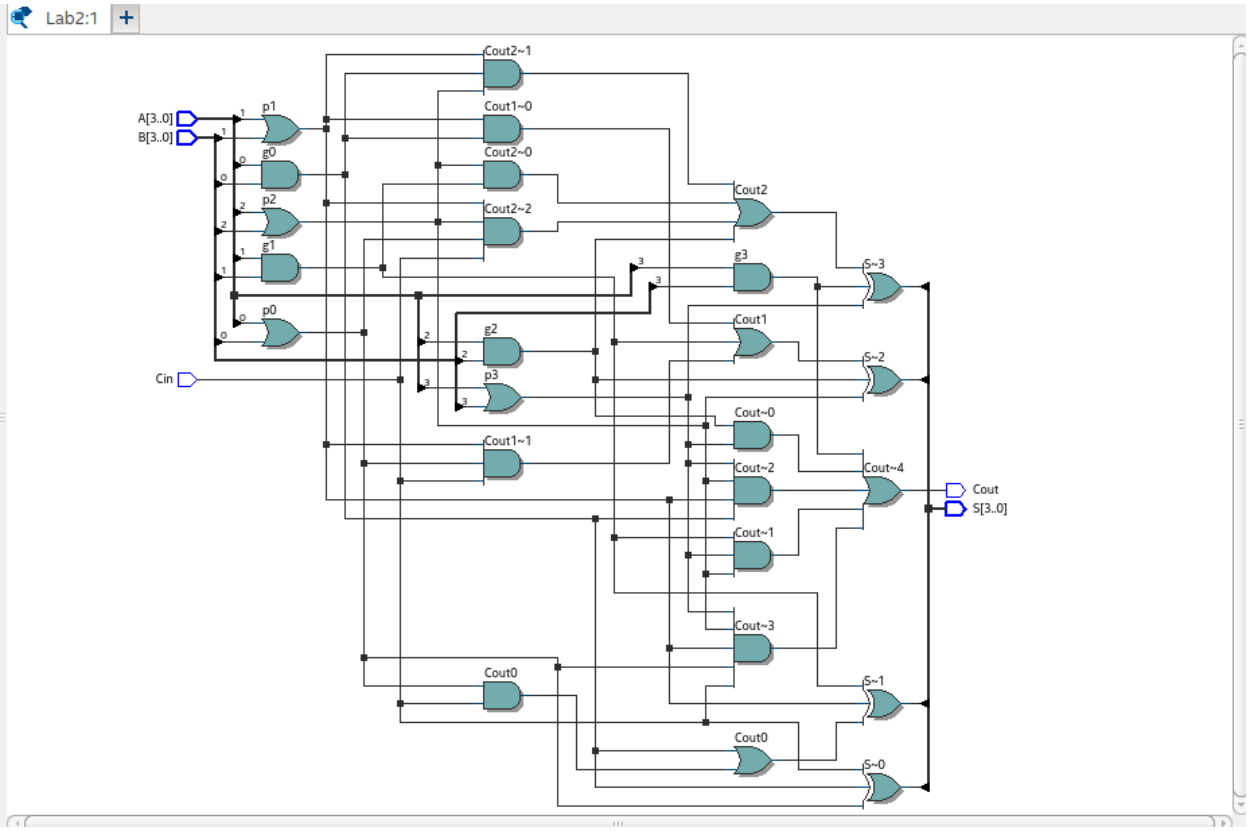

Figure 7. LACA

Code

```
-------------------------------------------------------------------------------------------------------------------------------------------
LACA--------------------------------------------------------------------------------
`timescale 1ps / 1ps
module LACA(A,B,Cin,Cout,S);

input [3:0]A,B;
input Cin;
wire g0,g1,g2,g3,p0,p1,p2,p3,Cout0,Cout1,Cout2;
output Cout;
output[3:0]S;

assign g0 = A[0]&B[0];
assign g1 = A[1]&B[1];
assign g2 = A[2]&B[2];
assign g3 = A[3]&B[3];

assign p0 = A[0]|B[0];
assign p1 = A[1]|B[1];
assign p2 = A[2]|B[2];
assign p3 = A[3]|B[3];

assign Cout0 = g0|(p0 & Cin);
assign Cout1 = g1|(g0 & p1)|(p0 & p1 & Cin);
assign Cout2 = g2|(g1 & p2)|(g0 & p2 & p1)|(p2 & p1 & p0 & Cin);
assign Cout = g3|(g2 & p3)|(g1 & p3 & p2)|(g0 & p3 & p2 & p1)|(p3 & p2 & p1 & p0 & Cin);

assign S[0]=g0^p0^Cin;
assign S[1]=g1^p1^Cout0;
```

```
assign S[2]=g2^p2^Cout1;
assign S[3]=g3^p3^Cout2;

endmodule

LACA8bit----------
module LACA8bit(A,B,Cin,Cout,S);

input [7:0]A,B;
input Cin;
wire Cout0;
output Cout;
output[7:0]S;

LACA first4(A[3:0],B[3:0],Cin,Cout0,S[3:0]);
LACA last4(A[7:4],B[7:4],Cout0,Cout,S[7:4]);

endmodule




RCA----------
module RCA(A,B,Cin,Cout,Sout);

input [3:0]A,B;
input Cin;
wire Cout0,Cout1,Cout2,Cout3;
output Cout;
```

```
output[3:0]Sout;

FA Add0(A[0], B[0], Cin, Cout0,Sout[0]);
FA Add1(A[1], B[1], Cout0, Cout1, Sout[1]);
FA Add2(A[2], B[2], Cout1, Cout2, Sout[2]);
FA Add3(A[3], B[3], Cout2, Cout, Sout[3]);

endmodule

module FA(A,B,Cin,Cout,Sout);

input A,B,Cin;
output Cout,Sout;

assign Cout = A&B | Cin&(A^B);
assign Sout = A^B^Cin;

endmodule
```

Test bench----------

```
`timescale 1ps / 1ps
module RCATB;

reg [7:0] A,B;
reg Cin;

wire Cout;
wire [7:0]S;
```

```
integer i;

LACA8bit uut (
.A(A),
.B(B),
.Cin(Cin),
.Cout(Cout),
.S(S)
);

initial begin
        for(i = 0; i < 2**16; i = i+1)begin
        {Cin, A , B} = i;
        #5;
        $display("%d %d %d => %d %d", A, B, Cin, Cout, S);
        end
end
endmodule
```

Was used for all three, Just changed the name of the UUT and the number of bits from 9->16