Colt Thomas

Lab 5 – Arithmetic Logic Unit

February 11, 2014

<u>Preparation</u>

1.) To prepare for the lab, we need to design the Full Adder circuit with truth tables, k-maps and equations. These are listed below:

-truth table for ALU:

A	B	Cin	S	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

-Kmap for ALU:

S	Cin		
AB		0	1
	00	0	1
	01	1	0
	11	0	1
	10	1	0

Cout	Cin		
AB		0	1
	00	0	0
	01	0	1
	11	1	1
	10	0	1

-by looking at these tables and maps we come up with the equations for S and Cout:

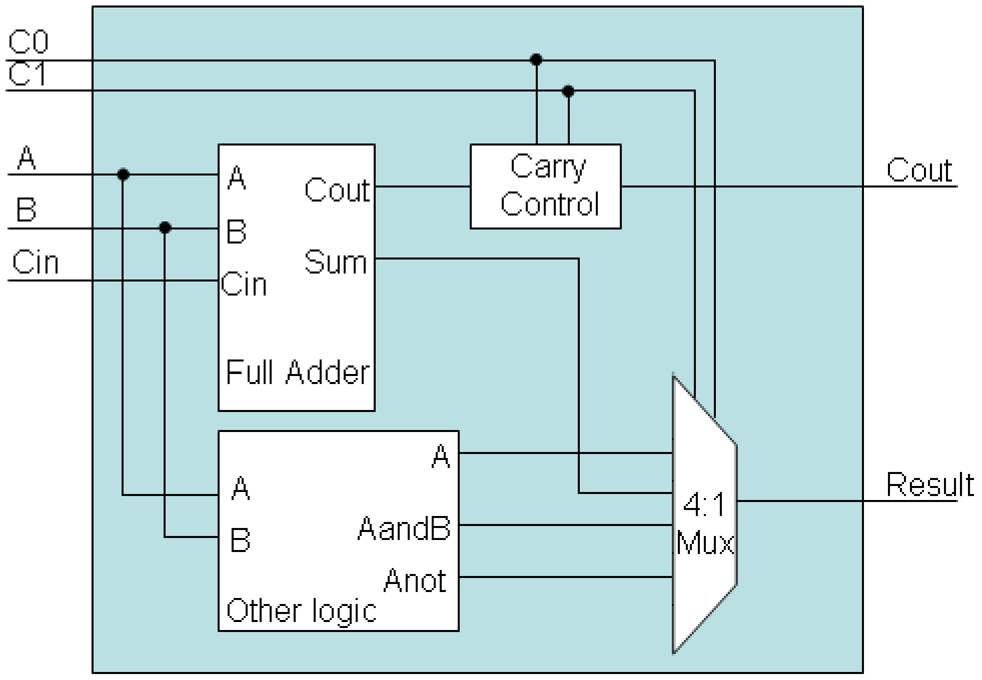
S = A (+) B (+) Cin ← the (+) is an XOR

Cout = BCin +AB + ACin

We have inputs of a one bit number A, one bit number B and a Carry in (Cin)

The outputs are Sum and Carry-Out.

2.) I am now going to learn how to work the Verilog code for the Full Adder module. Instead of doing a schematic for the lab, we will implement the following design for the full adder:



Note that this is to only be implemented in Verilog code!

Procedure

Below we have all the code and simulation for the required components for our 4bit ALU

Full Adder

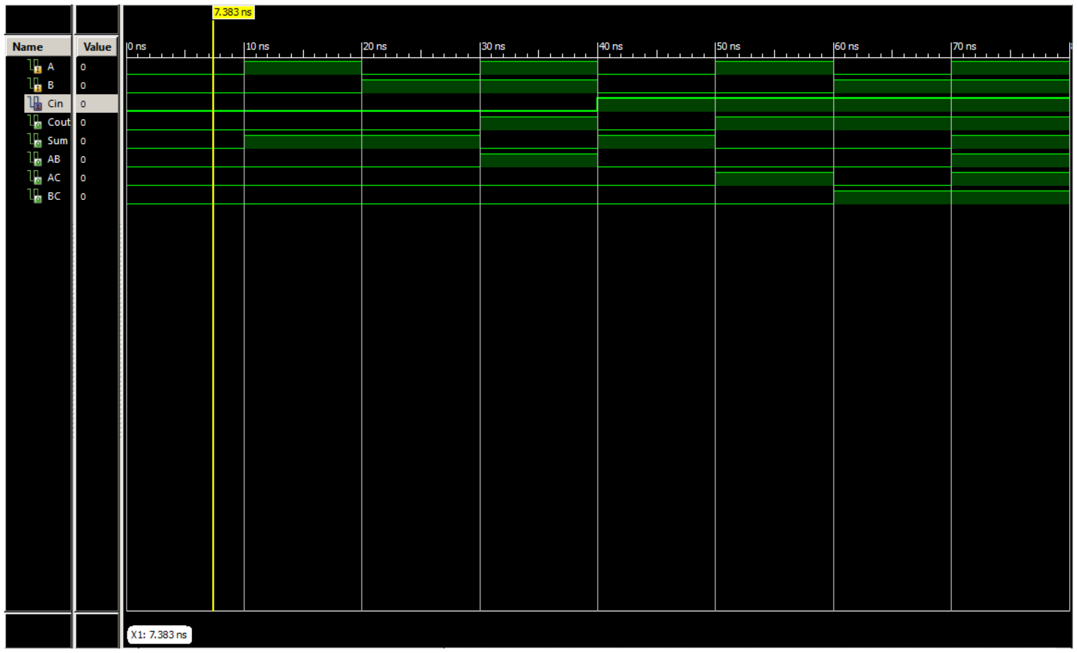
-Code for the Full Adder:

```
module FullAdder1(
   input [3:0] A,
   input [3:0] B,
   input [3:0] Cin,
   output [3:0] Cout,
   output [3:0] Sum
   );
            XOR3(Sum , A , B , Cin);
            AND2(AB , A , B);
            AND2(AC , A , Cin);
            AND2(BC , B , Cin);
            OR3(Cout , AB , AC , BC);


endmodule
```

-TCL file for the Full Adder:

```
#add all signals to the waveform viewer
wave add / -radix hex


#define how the data input signals will behave when you run the simulation
#the command "isim force add (signal) 0 -time 0 -value 1 -time 20ns -repeat 40ns" means that (signal)
#will have a value of 0 from time 0, then change to 1 at time 20ns, and then repeat that cycle every 40ns
isim force add A 0 -time 0 -value 1 -time 10ns -repeat 20ns
isim force add B 0 -time 0 -value 1 -time 20ns -repeat 40ns
isim force add Cin 0 -time 0 -value 1 -time 40ns -repeat 80ns



#Nothing will change in the waveform viewer until you run the simulation for some period of time.
run 80ns
```

-Simulation:



The simulation works; I did find out however that my initial truth table was incorrect and fixed it.

<u>4:1 MUX</u>

-Code for the 2:1 Mux and the 4:1 mux:

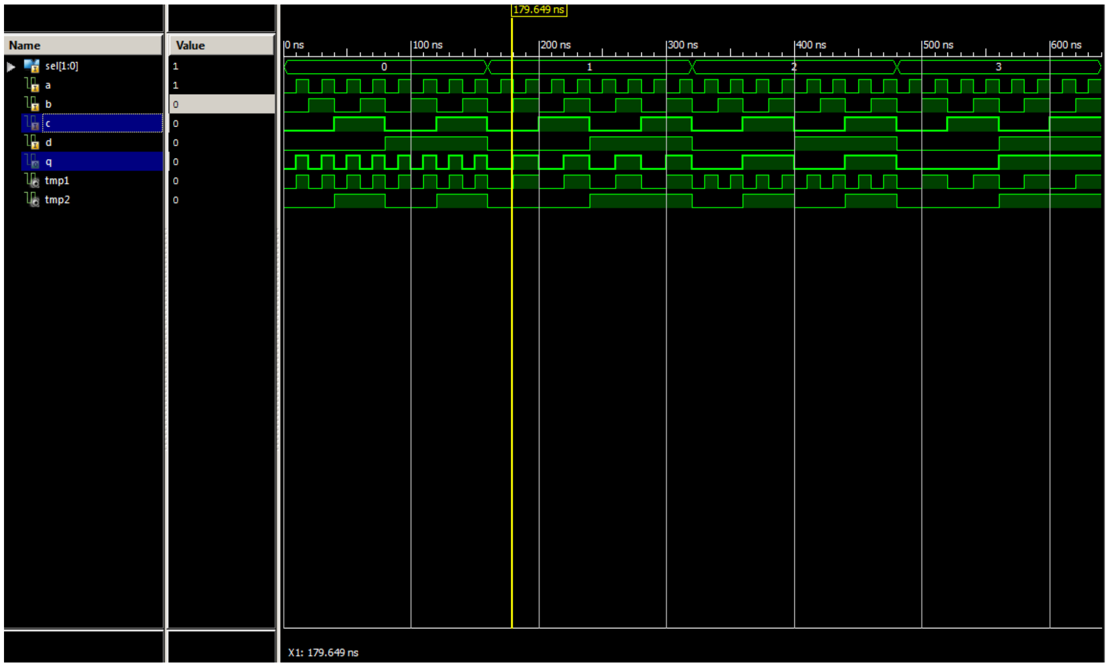
```
module mux21(q, sel, a, b);
            input sel, a, b;
            output q;
            wire selbar, a1, a2;

            not(selbar, sel);
            and(a1, selbar, a);
            and(a2, sel, b);
            or(q, a1, a2);

endmodule
```

```
module mux41(q, sel, a, b, c, d);
        input[1:0] sel;
        input a, b, c, d;
        output q;
        wire tmp1, tmp2;

        mux21 M0(tmp1, sel[0], a, b);
        mux21 M1(tmp2, sel[0], c, d);
        mux21 M2(q, sel[1], tmp1, tmp2);

endmodule
```

-TCL file for the MUX

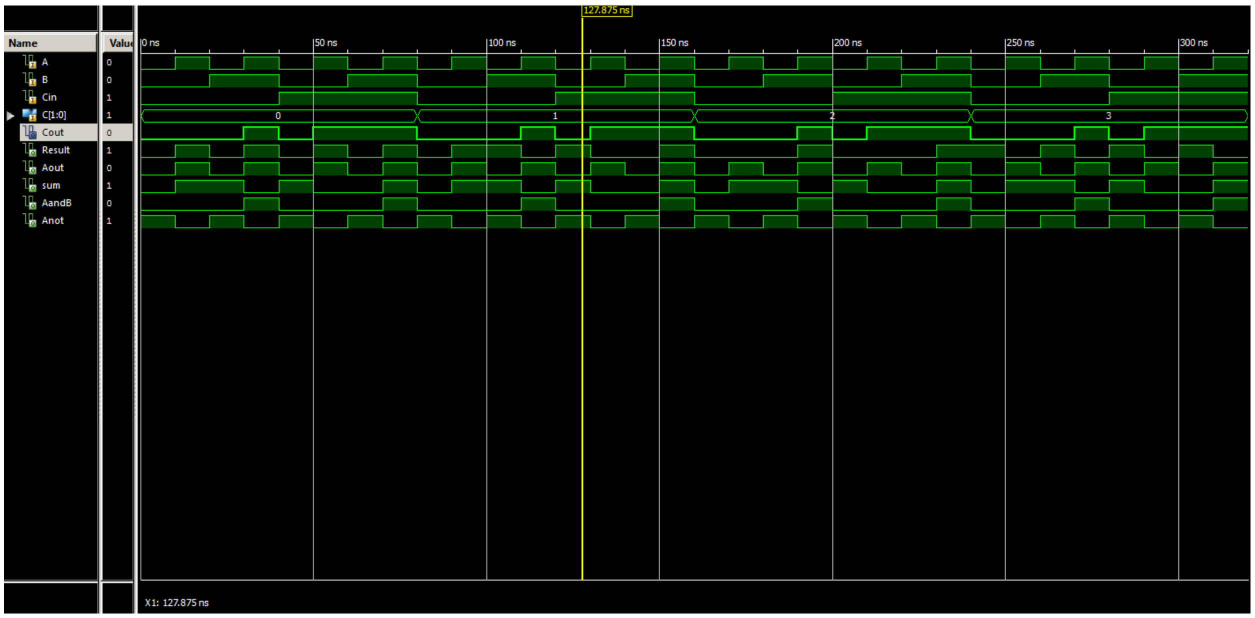
```
#add all signals to the waveform viewer
wave add / -radix hex


#define how the data input signals will behave when you run the simulation
#the command "isim force add (signal) 0 -time 0 -value 1 -time 20ns -repeat 40ns" means that (signal)
#will have a value of 0 from time 0, then change to 1 at time 20ns, and then repeat that cycle every 40ns
isim force add a 0 -time 0 -value 1 -time 10ns -repeat 20ns
isim force add b 0 -time 0 -value 1 -time 20ns -repeat 40ns
isim force add c 0 -time 0 -value 1 -time 40ns -repeat 80ns
isim force add d 0 -time 0 -value 1 -time 80ns -repeat 160ns


isim force add sel 00
run 160ns


isim force add sel 01
run 160ns
isim force add sel 10
run 160ns
isim force add sel 11
run 160ns
```

-Simulation of the MUX



1 bit ALU

-Simulation for ALU



-TCL for the ALU:

```
#add all signals to the waveform viewer
wave add / -radix hex
```

```
#define how the data input signals will behave when you run the simulation
#the command "isim force add (signal) 0 -time 0 -value 1 -time 20ns -repeat 40ns" means that (signal)
#will have a value of 0 from time 0, then change to 1 at time 20ns, and then repeat that cycle every 40ns
isim force add A 0 -time 0 -value 1 -time 10ns -repeat 20ns
isim force add B 0 -time 0 -value 1 -time 20ns -repeat 40ns
isim force add Cin 0 -time 0 -value 1 -time 40ns -repeat 80ns


isim force add C 00
run 80ns

isim force add C 01
run 80ns
isim force add C 10
run 80ns
isim force add C 11
run 80ns
```

-Verilog code for ALU

```
module ALU(
   input A,
   input B,
   input Cin,
   input [1:0]C,
   output Cout,
   output Result
   );
                wire sum , Aout , AandB , Anot;

FullAdder1 FA(A , B , Cin , Cout , sum);
OtherLogic OL (A , B , AandB , Anot , Aout);
mux41 MO (Result , C, Aout , sum , AandB, Anot);

endmodule
```

<u>4bit ALU</u>

-4bit ALU Verilog code

```
module ALU4bit(
   input [3:0] A,
   input [3:0] B,
   input Cin,
   input [1:0] Ctrl,
   output [3:0] R,
   output Cout
                );


                wire Cy0 ,Cy1 , Cy2 ;
ALU ALU1(A[0],B[0],Cin ,Ctrl, Cy0 , R[0]);
ALU ALU2(A[1],B[1],Cy0 ,Ctrl, Cy1 , R[1]);
ALU ALU3(A[2],B[2],Cy1 ,Ctrl, Cy2 , R[2]);
ALU ALU4(A[3],B[3],Cy2 ,Ctrl, Cout , R[3]);

Endmodule
```

-4bit ALU TCL file

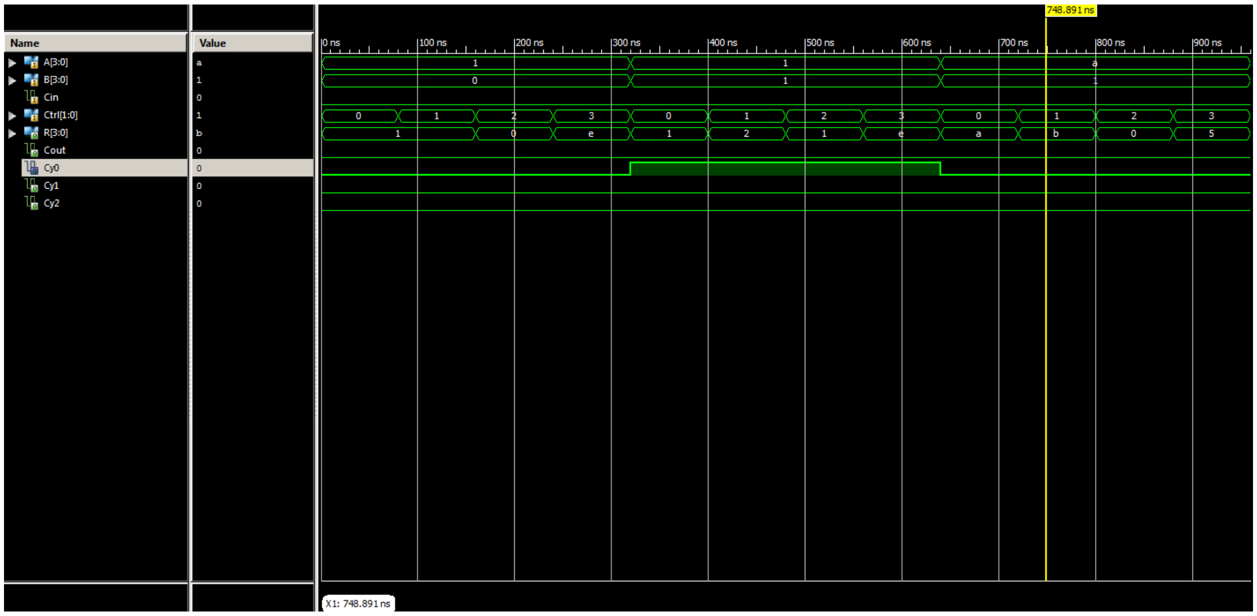
Here is the TCL code for the above addition:

```
#add all signals to the waveform viewer
wave add / -radix hex
```

```
        #Cin is always 0
        isim force add Cin 0

        #addition of 0001 and 0000
        isim force add A 0001
        isim force add B 0000

        isim force add Ctrl 00
        run 80ns

        isim force add Ctrl 01
        run 80ns
        isim force add Ctrl 10
        run 80ns
        isim force add Ctrl 11
        run 80ns

        #addition of 0001 and 0001

        isim force add A 0001
        isim force add B 0001

        isim force add Ctrl 00
        run 80ns

        isim force add Ctrl 01
        run 80ns
        isim force add Ctrl 10
        run 80ns
        isim force add Ctrl 11
        run 80ns

        #addition of 1010 and 0001

        isim force add A 1010
        isim force add B 0001

        isim force add Ctrl 00
        run 80ns

        isim force add Ctrl 01
        run 80ns
        isim force add Ctrl 10
        run 80ns
        isim force add Ctrl 11
        run 80ns

#all addition checks in the simulation
```

-4bit ALU Simulation:

I tested some basic addition. A screenshot is below:



-The .bit file that I used for the 4bit adder:

```
## Leds
NET R[0]  LOC = "J14"; # Bank = 1, Pin name = IO_L14N_1/A3/RHCLK7, Type = RHCLK/DUAL, Sch name = JD10/LD0
NET R[1]  LOC = "J15"; # Bank = 1, Pin name = IO_L14P_1/A4/RHCLK6, Type = RHCLK/DUAL, Sch name = JD9/LD1
NET R[2]  LOC = "K15"; # Bank = 1, Pin name = IO_L12P_1/A8/RHCLK2, Type = RHCLK/DUAL, Sch name = JD8/LD2
NET R[3]  LOC = "K14"; # Bank = 1, Pin name = IO_L12N_1/A7/RHCLK3/TRDY1, Type = RHCLK/DUAL, Sch name = JD7/LD3
#NET "Led<4>"  LOC = "E17"; # Bank = 1, Pin name = IO, Type = I/O, Sch name = LD4
#NET "Led<5>"  LOC = "P15"; # Bank = 1, Pin name = IO, Type = I/O, Sch name = LD5
#NET "Led<6>"  LOC = "F4";  # Bank = 3, Pin name = IO, Type = I/O, Sch name = LD6
NET Cout  LOC = "R4";  # Bank = 3, Pin name = IO/VREF_3, Type = VREF, Sch name = LD7

## Switches
NET B[0] LOC = "G18"; # Bank = 1, Pin name = IP, Type = INPUT, Sch name = SW0
NET B[1] LOC = "H18"; # Bank = 1, Pin name = IP/VREF_1, Type = VREF, Sch name = SW1
NET B[2] LOC = "K18"; # Bank = 1, Pin name = IP, Type = INPUT, Sch name = SW2
NET B[3] LOC = "K17"; # Bank = 1, Pin name = IP, Type = INPUT, Sch name = SW3
NET A[0] LOC = "L14"; # Bank = 1, Pin name = IP, Type = INPUT, Sch name = SW4
NET A[1] LOC = "L13"; # Bank = 1, Pin name = IP, Type = INPUT, Sch name = SW5
NET A[2] LOC = "N17"; # Bank = 1, Pin name = IP, Type = INPUT, Sch name = SW6
NET A[3] LOC = "R17"; # Bank = 1, Pin name = IP, Type = INPUT, Sch name = SW7

## Buttons
#NET "btn<0>" LOC = "B18"; # Bank = 1, Pin name = IP, Type = INPUT, Sch name = BTN0
#NET "btn<1>" LOC = "D18"; # Bank = 1, Pin name = IP/VREF_1, Type = VREF, Sch name = BTN1
NET Ctrl[0] LOC = "E18"; # Bank = 1, Pin name = IP, Type = INPUT, Sch name = BTN2
NET Ctrl[1] LOC = "H13"; # Bank = 1, Pin name = IP, Type = INPUT, Sch name = BTN3
```

Anomalies

-In general, this lab went well. I had some programming errors throughout the lab, but none that were really serious. I also had to figure out what I was doing with the .tcl files. My two biggest issues were to get my module inputs and outputs aligned. I had my parameters mixed up until I realized that it was just like making classes in C++. I also put the "wire" declaration within the inputs. This didn't affect my simulation, but it did affect my .bit file when I tried to upload it. I had to go to all my modules to make sure that parameters were correct. I then tested and passed off. I am glad that I got my design right from the beginning; this significantly reduced my debugging time. It helped that we did the circuit design for the last homework assignment.