Scott Sargent and Ezra Davis
Lab #3 - Implementing a 4-bit Four Function ALU
11/24/2013

Introduction:

The arithmetic logic unit (ALU) is a subsystem inside the central processing unit (CPU). This unit is designed to perform integer math and implement bitwise logic functions. In this lab, we had to program a 4-bit ALU to display the outputs of different arithmetic operations (addition, subtraction using two's complement, bitwise AND, and bitwise OR) on a seven-segment display using a hardware description language (HDL) called Verilog. We used a Nexys-3 FPGA for testing of our ALU.

Discussion and Results:

We set up a new project and created a Verilog module called hex2seg to convert 4-bit binary to an output suitable for a 7-segment display. (Appendix A) The truth table for this circuit is in prelabs.

We had already created a hex to 7-segment display in lab #2 as a bonus. We used that code from the previous lab and modified it to meet the requirement for lab #3.

The AN output controls the demultiplexer which switches between the different 7-segment displays on the FPGA. We only wanted to use one of the displays, so AN outputs a constant 1110.

We created a implementation constraints file (code in lab description) and generated the bit file for programming the FPGA. We then programmed the FPGA and tested all different values of the inputs to make sure the display behaved properly.

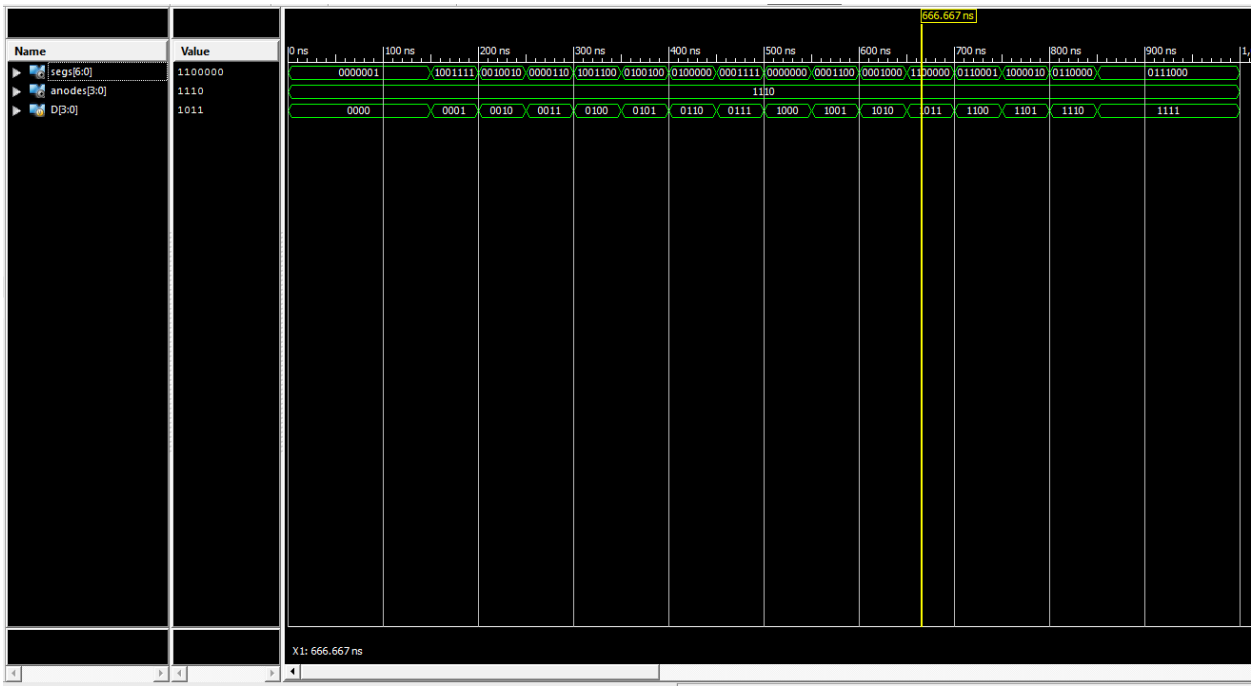
We then imported our 4-bit adder from last lab and implemented a new top-level module (adder4hex) that instantiated both it and hex2seg. The code for this module is in the lab description, as is the implementation constraints file associated with it. We then created a new implementation constraints file, which has code in the lab description document, and removed the other one.

Next, we generated a programming file and programmed the FPGA to verify the circuit.

We then created a two's complement circuit to convert an input to the four-bit adder.

The 4-bit adder was an essential part of creating the 2's complement. The first part was to take the inverse of the value. The second and final part was to add binary one to the inverse value.

We then created a Verilog test fixture and ran an exhaustive test on all the inputs.



Exhaustive test of the 4-bit 2's complement conversion circuit

The next step was to implement the bitwise OR and AND modules. These were very straightforward. The code for the OR and AND modules are in appendices D and E, respectively.

Now we put all the different operations together using a 4 to 1 by 1 multiplexer. The "op" input controls which operation is returned. 00 for addition, 01 for subtraction, 10 for AND, and 11 for OR.

After downloading and testing the current circuit on the FPGA, we added an output for overflow and another indicating whether the number produced was positive or negative (code in Appendix B).

The Overflow output assumes that addition is unsigned and subtraction is signed using 2's complement. The bitwise operations have no overflow.

The negative output is switched off unless there is a negative number produced by subtraction and subtraction is the currently displayed operation.

Conclusion:

We were able to successfully create a 4-bit ALU.

All the tests were successfully implemented. We learned how to reuse circuits across projects and that the 7-segment display is active low.

Appendix A:

```
module Hex2Seg(
       input [3:0] D,
       output [6:0] segs,
       output [3:0] anodes
       );

 assign segs =  (D==4'b0000)?7'b0000001 : ( //0
      (D==4'b0001)?7'b1001111 : ( //1
      (D==4'b0010)?7'b0010010 : ( //2
      (D==4'b0011)?7'b0000110 : ( //3
      (D==4'b0100)?7'b1001100 : ( //4
      (D==4'b0101)?7'b0100100 : ( //5
      (D==4'b0110)?7'b0100000 : ( //6
      (D==4'b0111)?7'b0001111 : ( //7
      (D==4'b1000)?7'b0000000 : ( //8
      (D==4'b1001)?7'b0001100 : ( //9
      (D==4'b1010)?7'b0001000 : ( //A
      (D==4'b1011)?7'b1100000 : ( //B
      (D==4'b1100)?7'b0110001 : ( //C
      (D==4'b1101)?7'b1000010 : ( //D
      (D==4'b1110)?7'b0110000 : 7'b0111000 //E F
      ))))))))))))));
 assign anodes = 4'b1110;

endmodule
```

Appendix B:

```
module ALU4bit(
       input [3:0] A,
       input [3:0] B,
       input [1:0] op,
       output [3:0] Result,
       output negSign,
       output [6:0] segs7,
       output [3:0] anodes,
  output OverFlow
       );

  wire [3:0] summ, diff, my4and, my4or, Bneg, Barg, magg;
  wire ovf;

  comp2 U1(B,Bneg);

  assign Barg = (op=='b01) ? Bneg : B;

  fourAdd U2(A,Barg,summ,ovf);

  and4bits U3(A,B,my4and);
```

```
  or4bits U4(A,B,my4or);

  comp2 U5(summ, magg);

  assign diff = (negSign == 1'b1) ? magg : summ;
 assign Result = (op=='b00) ? summ : (
       (op=='b01) ? diff : (
       (op=='b10) ? my4and : my4or //if op=='b11
       ));

 Hex2Seg U6(Result, segs7, anodes);
 assign negSign = (op==2'b01 && summ[3]==1'b1) ? 1 : 0;

assign OverFlow = //Addition: (unsigned)
                          (op==2'b00) ? (  ( ((~summ[3])& B[3]) |
                                             ((~summ[3])& A[3]) )  |
                                             (A[3] & B[3])  ) : (
                          //Subtraction: (signed)
                          (op==2'b01) ? ovf :
                          1'b0); //Logical operations don't have Overflow
endmodule
```

Appendix C:

```
module Add4hex(
       input [3:0] A,
       input [3:0] B,
       output [6:0] segs7,
       output [3:0] anodes
       );
 wire Overflow;
 wire [3:0] Sum;

 fourAdd U1(A,B,Sum,Overflow);
 Hex2Seg U2(Sum,segs7,anodes);
endmodule
```

Appendix D:

```
module or4bits(
    input [3:0] A,
    input [3:0] B,
    output [3:0] orAB
    );

       assign orAB = A | B;

endmodule
```

Appendix E:

```
module and4bits(
    input [3:0] A,
    input [3:0] B,
    output [3:0] andAB
    );

       assign andAB = A & B ;

endmodule
```