

Worcester Polytechnic Institute
Electrical and Computer Engineering Department

ECE3801 – Lab 2 (4-bit ALU)

Copyright: Feb, 2005

Written by Ahmad Hatami
Modified by Samant Kakarla

Modified: November 2006
S. Jarvis

Please have all individual components created in this lab available to all parties working together to complete the lab. Make sure everyone also has the components available individually from previous labs. It is a good idea to backup your data on another media apart from the network space you have available.

Objective

In this lab you will design a 4-bit ALU. In this process you will learn how to use MSI circuit blocks and hierarchical structures to design a relatively complex digital system.

Big Picture

Arithmetic Logic Unit (ALU) is an important part of every computer and calculator. An ALU usually has two input operands for data and several inputs for control. Depending upon the value of the control inputs, one of several arithmetic or logical operations is performed on the input operands. You will design a four bit ALU which can perform: Addition, Subtraction, AND, OR, and XOR operations. Your ALU must be able to display the result as a signed or unsigned number. The ALU has the following structure.

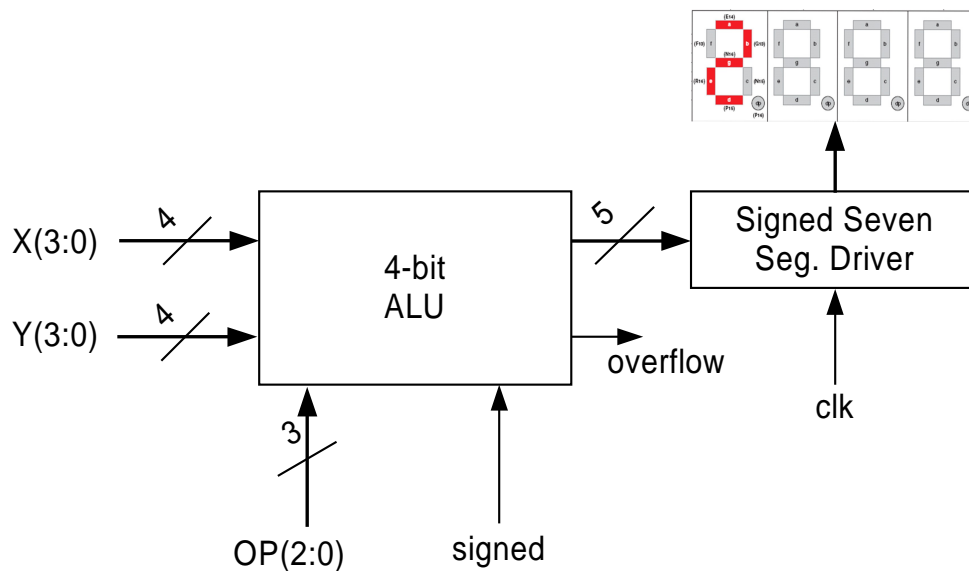


Figure 1

X(3:0): The first 4-bit operand

Y(3:0): The second 4-bit operand

OP(2:0): Three bit control input, known as the opcode, which identifies the operation to be performed. Table 1 defines the opcodes for this ALU, and their associated operations.

Signed: All arithmetic operations performed by your ALU will be performed using two's complement format. However, your 7 segment decoder from Lab 1 was designed unsigned 4-bit binary to hex representations. If the *signed* input signal is high and the result of the operation is a negative 2's comp number, the display

unit should show the appropriate negative result using the sign segment of the display (i.e.,). Otherwise, the unsigned result should appear on the display.

Signed = 0 Unsigned representation
Signed = 1 If the result is negative two's complement representation

Overflow: If the result of the operation generates an overflow this output will be activated.

OP2	OP1	OP0	Function	Ovl
0	0	0	ADD X , Y	Overflow
0	0	1	SUB X , Y	Overflow
0	1	0	SUB Y , X	Overflow
0	1	1	INC X	Overflow
1	0	0	DEC X	Overflow
1	0	1	X and Y	0
1	1	0	X or Y	0
1	1	1	X xor Y	0

Table 1

Design

You should build your design up in pieces as described below. Design each piece individually in the Schematic Editor and **test it** before building the next piece.

Bitwise Logical Operations

The logical operations that your ALU should perform are *x and y* ($x \cdot y$), *x or y* ($x + y$), *x xor y* ($x \oplus y$). All of these are bitwise operations. For example, bit 3 of the output for *x and y* ($x \cdot y$) is the logical AND of $x(3)$ and $y(3)$.

1. Create a new project and then create a schematic file in your project named *and4bits* and design a circuit with the four bit *and* functionality. (Figure 2)
2. Save your design.
3. Create a test bench waveform and verify your design.
4. After verification, create a new symbol for your design and name your symbol *and4bits*.
5. Now create, test and save schematics for a 4-bit *or*, and a 4-bit *xor* and name them *or4bits*, and *xor4bits* respectively.

Controlled Inversion

For subtracting two numbers we need to find the two's complement of a number as described before. In order to find two's complement of a number we need to find the bitwise complement of a number first, and then add one to the that. Here you are going to design a component that performs the bitwise inversion. Figure 4 shows the symbol for this component. Output of this component is either the bitwise inverted version of its input or the input itself, based on the value of *Ctrl* input.

10. Add a new schematic file to your project and name it *inv4ctrl*.
11. Design a circuit which provides this function.
12. Check your schematic and save the file.
13. Create a test bench waveform to verify the behavior of system.
14. Make sure that your design works properly.
15. Create a symbol for this component and name it *inv4ctrl*.

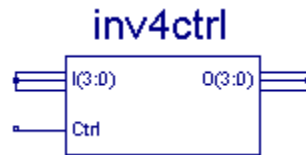


Figure 4

Constant signals

There are times when you want to apply a constant value (0 and 1) to all bits of an input. It will be more convenient to do such if you design a component which provides these signals. This approach has two benefits, first it will reduce the amount of wiring in your design, and secondly if you decide to change the values of these constants in your design, you only need to modify a single component not the whole system. Figure 5 shows the symbol for this component. There are two outputs in this system (*ones = 1111*, *zeros = 0000*).

16. Create a new schematic file to your project and name it *consts*.
17. Design a circuit for this component.
18. Verify your design, and save it.
19. Create a symbol for this component and name it *consts*.

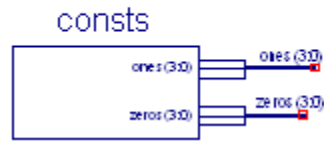


Figure 5

Signed Numbers

In this part you will design a circuit capable of representing a number as signed or unsigned. Figure 6 shows the symbol for this design, and Table 2 shows the function of this component.



Figure 6

signed	A(3:0)	A ₃	Q(3:0)	sign
0	A ₃ A ₂ A ₁ A ₀	X	A ₃ A ₂ A ₁ A ₀	0
1	A ₃ A ₂ A ₁ A ₀	0	A ₃ A ₂ A ₁ A ₀	0
1	A ₃ A ₂ A ₁ A ₀	1	Two's complement of A ₃ A ₂ A ₁ A ₀	1

Table 2

20. Create a new schematic file in your project and name it as *signeddigit*.
21. Design a circuit which provides the function described in Table 2.
22. Check your schematic and save your design.
23. Create a test bench waveform to generate all the combinations of input and verify the outputs of your design.
24. Create a symbol for your design and name it as *signeddigit*.

Seven Segment Display for Signed Numbers

You designed a seven segment driver unit in your first lab which is capable of displaying four hexadecimal numbers simultaneously on the Spartan3 board. Now you will design a new component that displays two signed numbers on the seven segment displays. Your design will have two 4-bit inputs ($X(3:0)$, and $Y(3:0)$), two single bit inputs ($sign_x$, and $sign_y$); which represent whether you want to consider X and Y inputs as signed number or not; a clk and $Reset$. The outputs of the system are the familiar signals that you have used to drive the seven segments as before.

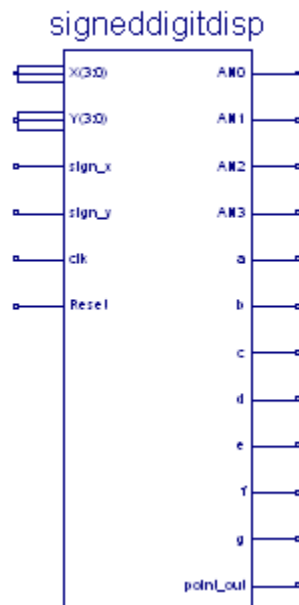


Figure 7

25. Create a new schematic file in your project and name it as *signeddigitdisp*.
26. Design this component using your newly *signeddigit* and other parts you have designed before.
27. Check the schematic and save the file.
28. Create a symbol for new design and save it.
29. Since this is an important unit in your project, you are going to download this design into the board and make sure your design functions properly.
30. Create a new schematic file similar to Figure 8.
31. Check your schematic and save your file.
32. Add a constraint file for this design. (Table 3)

33. Save your constraint file.
34. Generate the programming file for this design, and download that into the board.
35. Make sure that the circuit works properly and show it to the TA for verification.

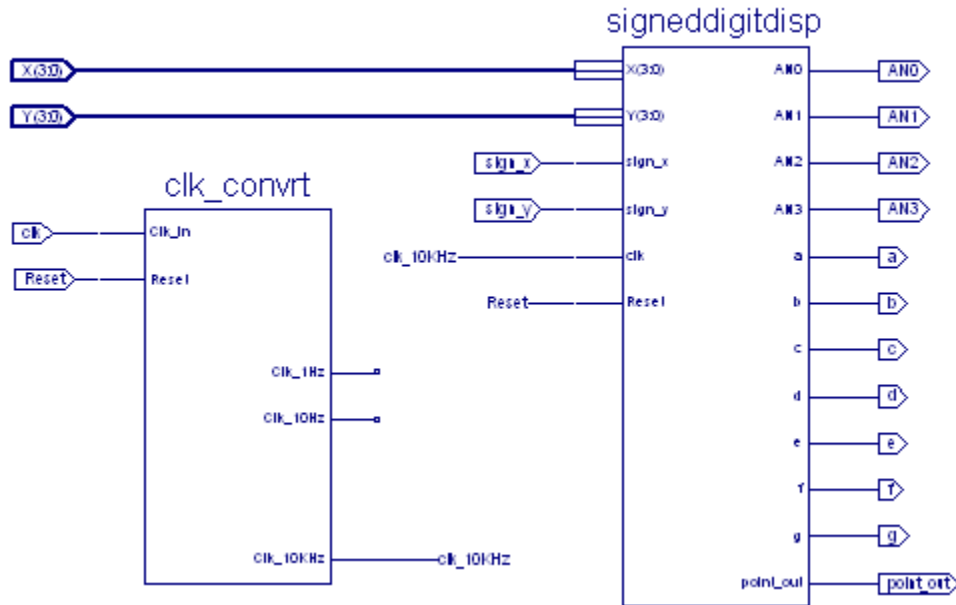


Figure 8

Adder Revisited

As mentioned before, the first five instructions of your ALU can be accomplished using an adder with some control logic. Here we have provided a template for your design that you can use (Figure 9).

Some notes about this template:

- There are two 4-bit inputs ($A(3:0)$, and $B(3:0)$) representing two operands) on the data path.
- There are three single bit inputs ($op2$, $op1$, and $op0$ representing the operation to be performed on the operands). (Control-path).
- The three-to-eight decoder activates one of its outputs based on the selected operation. Create this decode in VHDL.
- Using the two multiplexers you can decide which operand should go through the inversion and which should not.

- If you are using multiplexers with enable inputs pass them as an additional input of the component.
- Your main task is to design the black box in a way to generate the appropriate controls for *S1*, *S2*, *Ctrl*, and *Cin* signals. Create this black box in VHDL. In fact you can create both the decoder and the black box as one module and that would minimize the design a bit.
- Your black box contains only combinational logic elements and may not need all the outputs provided by the decoder.

```

NET "a" LOC = "E14";
NET "b" LOC = "G13";
NET "c" LOC = "N15";
NET "d" LOC = "P15";
NET "e" LOC = "R16";
NET "f" LOC = "F13";
NET "g" LOC = "N16";
NET "point_out" LOC = "P16";
NET "Reset" LOC = "L14";
NET "AN0" LOC = "D14";
NET "AN1" LOC = "G14";
NET "AN2" LOC = "F14";
NET "AN3" LOC = "E13";
NET "Ck" LOC = "T9";
NET "X<0>" LOC = "F12";
NET "X<1>" LOC = "G12";
NET "X<2>" LOC = "H14";
NET "X<3>" LOC = "H13";

```

Table 3

36. Create a new schematic file in your project and enter your design for the adder unit we just discussed.
37. Check your schematic and save your file.
38. Create a test bench waveform and test your design.
39. Create a symbol for your design.
40. Create a new schematic for testing your new unit.

41. Use as a reference for your test circuit.
42. Save your design.
43. Create a user constraint file similar to Table 4.
44. Generate the programming file and download the bit stream into the board.
45. Make sure that your circuit works properly. You should be able to do: ADD, SUB (a,b), SUB (b,a), INC, and DEC operations.
46. Show your functional system to the TA.

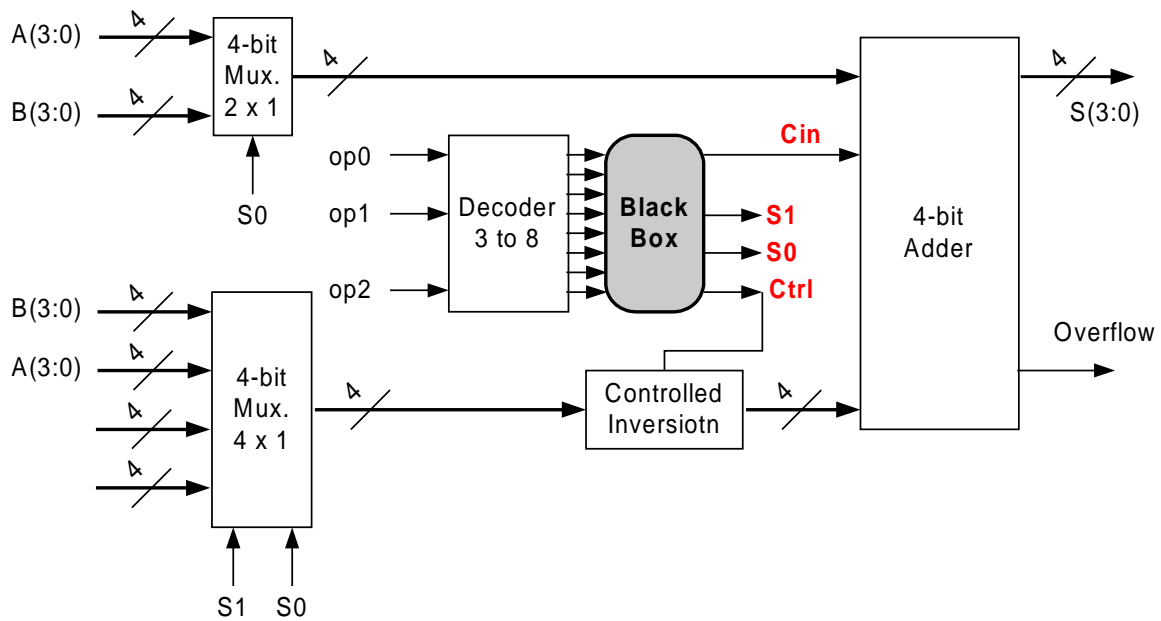


Figure 9

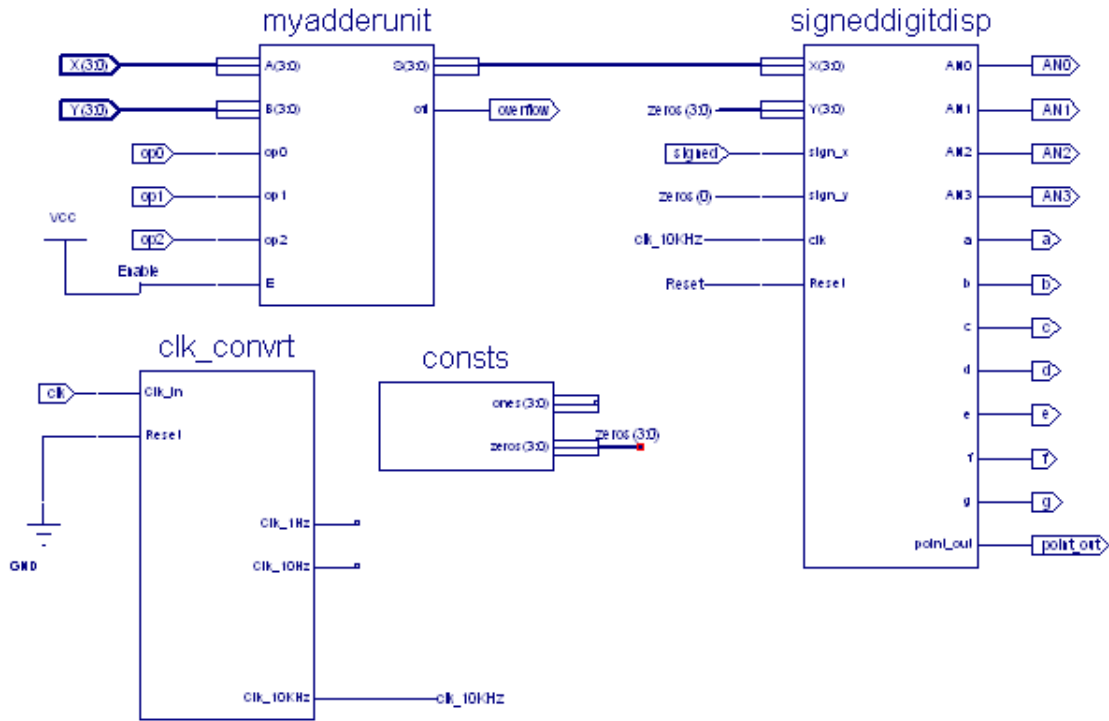


Figure 10

NET "a" LOC = "E14";
NET "b" LOC = "G13";
NET "c" LOC = "N15";
NET "d" LOC = "P15";
NET "e" LOC = "R16";
NET "f" LOC = "F13";
NET "g" LOC = "N16";
NET "point_out" LOC = "P16";
NET "AN0" LOC = "D14";
NET "AN1" LOC = "G14";
NET "AN2" LOC = "F14";
NET "AN3" LOC = "E13";
NET "Clk" LOC = "T9";
NET "Y<0>" LOC = "F12";
NET "Y<1>" LOC = "G12";
NET "Y<2>" LOC = "H14";
NET "Y<3>" LOC = "H13";
NET "X<0>" LOC = "J14";

Table 4

Putting Your ALU Together

Your ALU should do as many operations in parallel on the input data as there is hardware available. For example, all the logical operations can be done at the same time. One of the ADD/SUB instructions can also be done in parallel. Figure 11 shows the general architecture that you can use. You need to design the appropriate functions in the black box. Also note that you have to make sure that the overflow output must be deactivated for logical operations.

47. Create a new high level schematic file in your project and name it as *myAlu*.
48. Use as a baseline and complete the design. Note that since we did not have any push buttons left on the board, we eliminated the reset input.
49. Check the schematic and save the file.
50. Create a user constraints file for your design similar to Table 5.
51. Generate the bit stream file and download it into the board.
52. Make sure that system does all operations as mentioned in Table 1.
53. Show your working system to the TA.

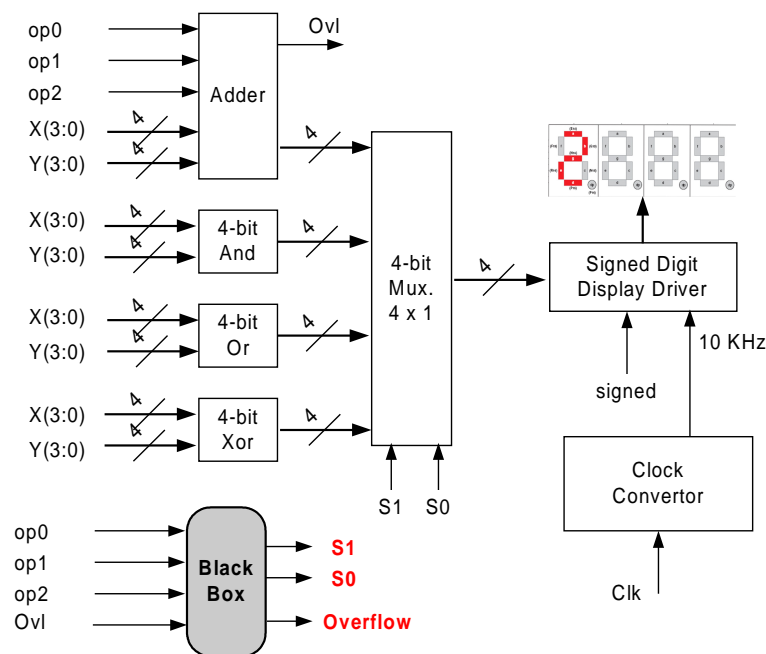


Figure 11

```
NET "a" LOC = "E14";
NET "b" LOC = "G13";
NET "c" LOC = "N15";
NET "d" LOC = "P15";
NET "e" LOC = "R16";
NET "f" LOC = "F13";
NET "g" LOC = "N16";
NET "point_out" LOC = "P16";
NET "AN0" LOC = "D14";
NET "AN1" LOC = "G14";
NET "AN2" LOC = "F14";
NET "AN3" LOC = "E13";
NET "Clk" LOC = "T9";
NET "Y<0>" LOC = "F12";
NET "Y<1>" LOC = "G12";
NET "Y<2>" LOC = "H14";
NET "Y<3>" LOC = "H13";
NET "X<0>" LOC = "J14";
NET "X<1>" LOC = "J13";
```

Table 5

ALU Sign-Off Sheet

Make sure lab instructors initialize each part. Attach this page to the end of your report.

Your Name:

Lab Partner:

Date Performed:

Demonstrated that the circuit works correctly (Total = 75 pts for all part):

2's complement adder (w/ signed display) works: _____

All non-logical operations work : _____

- Adding
- Subtraction
- Increment
- Decrement

- ALU (including logical operations) works: _____

Report = 25 pts