

Basic Digital Circuits

Computer Science 240

Laboratory 6

In lab today, you will experiment with some basic digital circuits you have learned about in lecture, including the exclusive-OR, multiplexer, and decoder. We will also begin to experiment with an Arithmetic Logic Unit (ALU), which you will be learning about in lecture tomorrow. The experiments will give you more practice building physical circuits and using LogicWorks to simulate the circuits.

Exclusive-OR

Exercise 1: A function for an **even parity** bit produces a '0' when the number of '1' bits in the n-bit input is even, and a '1' otherwise.

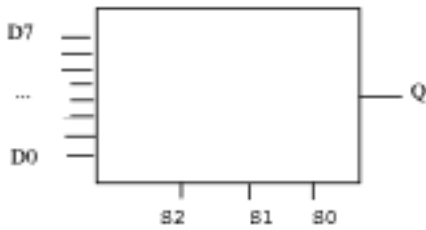
1. Begin by filling in the truth table for P, to produce an even parity bit for inputs ABCD:

A	B	C	D	P_{even}
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

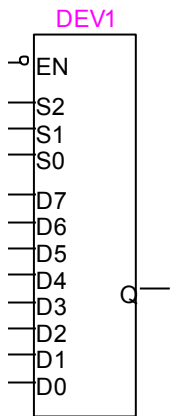
2. An **odd parity** bit function produces a '0' when the number of '1' bits in the n-bit input is odd, and a '1' otherwise. What is the relationship between the even and odd parity functions?
3. An exclusive-OR gate is useful for solving this type of problem. Why?
4. Design an even parity bit circuit using exclusive-OR gates only (you can draw this in LogicWorks and paste it here, as we did in lab last week):
5. Add one exclusive-OR gate to your circuit so that it also produces the odd parity bit (HINT: it is possible for an exclusive-OR gate to act as a NOT gate). Paste your circuit here:
6. In LogicWorks, simulate inputs with binary switches and display P_{even} and P_{odd} using binary probes, and verify that your circuit produces the correct outputs for each input combination.
7. Build your circuit on the protoboard, using a single 74LS86 chip. Demonstrate to the instructor.

Multiplexer

Exercise 2: An 8-bit multiplexer has 8 input lines, 3 control lines, and 1 output line. The 3 control lines S2,S1,S0 select which one of the 8 input lines D0..D7 is gated through to the output line Q:



1. Simulate this device using LogicWorks. You can find an 8-bit multiplexer in the **Simulation Logic** library. The device looks like this:

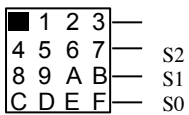


The **EN** input is an enable line. It must be connected for the device to be enabled, or work correctly.

Notice that it has an open bubble on the input to indicate that it is active low (which means that you must connect it to **ground**).

Connect D0 – D8 to **binary switches**. Use a **binary probe** to display your output.

Use the bottom three bits of a **Hex Keyboard w/o STB** to simulate the select lines:



Experiment with the device, and demonstrate to the instructor that you understand the operation of the multiplexer.

2. Although the most common use of a multiplexer is select an input to gate through to the output from among a set of possibilities, it can also be used as a code detector for a function.

This following truth table for a function Q indicates whether or not a 3-bit value S2 S1 S0 is a power of two:

S2	S1	S0	Q
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

On your Protoboard, connect a 74151 (8 x 1 MUX) to produce the function Q. Demonstrate to the instructor that your circuit detects inputs on the select lines which represent a power of 2.

Decoder

Exercise 3: An $n \times 2^n$ decoders takes n inputs and produces 2^n outputs. Only one of the 2^n outputs is selected for any given n -bit input combination. The most common use of a decoder is to activate or select only one of a possible set of outputs

The 74138 is a 3x8 decoder: given 3 inputs, it selects one of 8 outputs to be active.

(NOTE: there are **3** enable lines on this device, G1, G2A and G2B. G1 is tied to +5V (**active high**), and G2A and G2B are tied to ground (**active low**).

1. Connect the circuit on the protoboard. Complete the truth table, based on your observations.

C	B	A	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
0	0	0	0	1	1	1	1	1	1	1
0	0	1	1	0	1	1	1	1	1	1
0	1	0	1	1	0	1	1	1	1	1
0	1	1	1	1	1	0	1	1	1	1
1	0	0	1	1	1	1	0	1	1	1
1	0	1	1	1	1	1	1	0	1	1
1	1	0	1	1	1	1	1	1	0	1
1	1	1	1	1	1	1	1	1	1	0

2. The outputs for this device are **active low**. What does that mean? **Active low means that the output has a value of 0 when it is active, instead of 1.**

4-bit Adder

Exercise 4: Use LogicWorks to construct a 4-bit adder. Open the *lab6_4bit.cct*, which contains 4 1-bit adders. Use the hex keyboard and hex display to simulate inputs and outputs (also display the carry-out with a binary probe) and demonstrate to the instructor.

Take a screen shot of your circuit and paste it here.

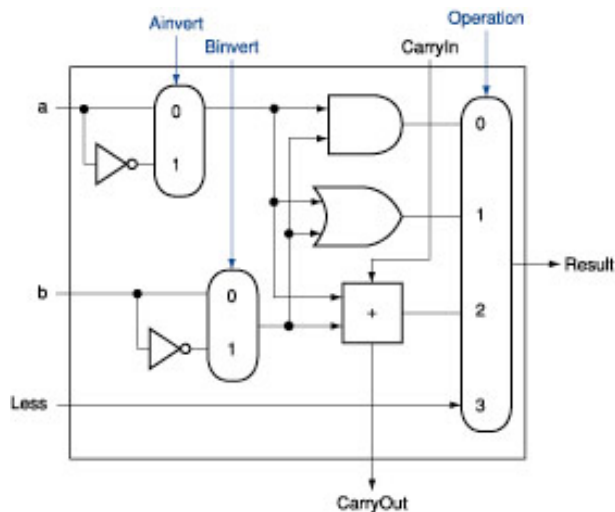
1-bit ALU

Exercise 5: Open the *lab6_aluparts.cct* circuit, which contains 2x1 multiplexers (represented by the smaller oval shaped symbols in the diagram below), a 1-bit full adder (represented by the square with the + symbol), and a 4x1 multiplexer (represented by the large oval shaped symbol). Use these, plus additional logic gates, to implement the 1-bit ALU below.

Use binary switches for the inputs **a** and **b**, **Operation** (2 bits, **Op1** and **Op0**), **Ainvert**, and **Binvert**, **Carryin** (for addition), and **Less** (sets the result on a Set Less Than (SLT) instruction).

Use binary probes for the outputs (**CarryOut** and **Result**).

Label all the inputs and outputs.



This is the truth table for the 1-bit ALU:

Operation		Result
Op1	Op0	
0	0	a AND b
0	1	a OR b
1	0	a + b
1	1	Less (0 or 1, from the result of a SLT)

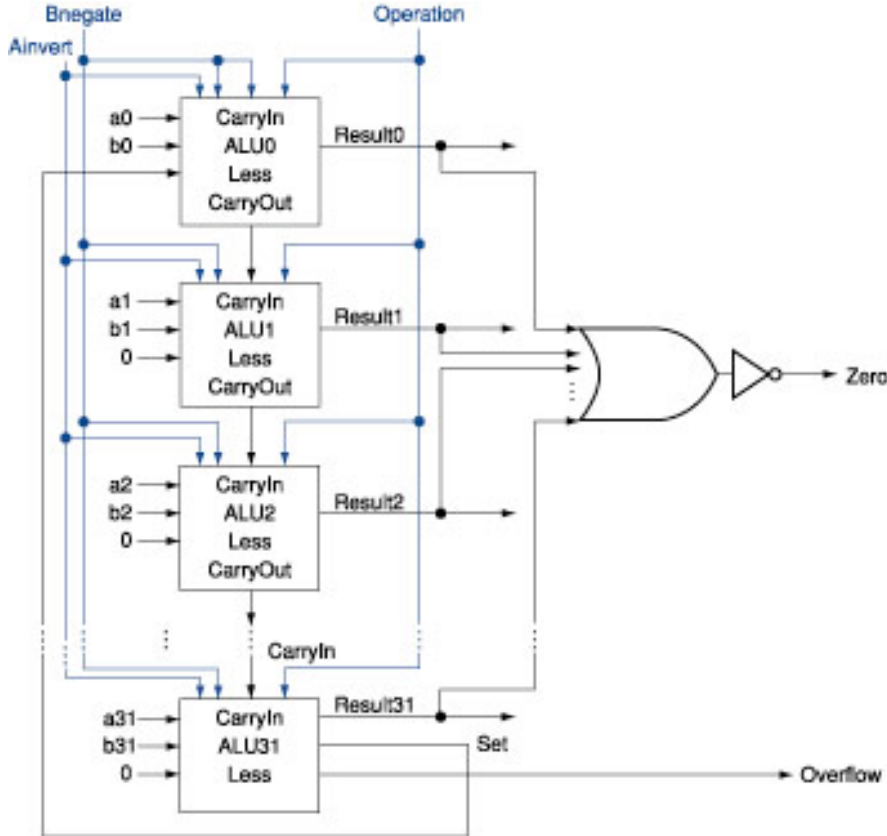
Select values for **a**, **b**, **Ainvert**, **Binvert**, and **Less**, and check that the result is as expected for the operation specified by the two **Operation** bits **Op1** and **Op0**. Select a set of tests to demonstrate that your circuit works correctly. List the tests below, and demonstrate to the instructor:

x indicates don't care: for the 1-bit ALU, the less bit is either 1 or 0, simulated by user

Op1	Op0	a	b	Ainvert	Binvert	Carry in	Less	Result	Carryout
0	0	0	0	0	0	0	x	0	0
0	0	1	1	0	0	0	x	1	1
0	0	0	1	1	0	0	x	1	1
0	1	0	0	0	0	0	x	0	0
0	1	1	0	0	0	0	x	1	0
0	1	0	1	1	0	0	x	1	1
1	0	0	0	0	0	0	x	0	0
1	0	1	0	0	0	0	x	1	0
1	0	1	1	0	0	0	x	0	1
1	1	1	1	0	1	1	x	x	1
1	1	0	1	0	1	1	x	x	0

4-bit ALU

Exercise 6: Similar to the way you made an n-bit adder, you can make a n-bit ALU by using n 1-bit ALUs, connected together as described in the text and lecture:



Open the circuit *lab6alu.cct* .

The following is the truth table for the 4-bit ALU circuit:

Ainvert	Binvert	Op1	Op0	Function
0	0	0	0	a AND b
0	0	0	1	a OR b
0	0	1	0	a + b (add)
0	1	1	0	a - b
0	1	1	1	set on less than

Set the Ainvert, Binvert, Op1, and Op2 bits to perform the following operations to test your circuit. Use the indicated test inputs for A and B. Give the values of Output, Carry-out, Zero, and Overflow for each test:

Function	Test Inputs	Output	Carry-out	Zero	Overflow
A + B	A = 3, B = 7	A	0	0	1
A + B	A = F, B = E	D	1	0	0
A AND B	A = 1, B = F	1	1	0	0
A AND B	A = 7, B = 8	0	0	1	0
A OR B	A = 3, B = C	F	0	0	0
A NOR B	A = 6, B = 9	0	0	1	1
SLT	A = 3, B = 9	0	0	1	1
SLT	A = C, B = 7	1	1	0	1

Demonstrate and explain the results to the instructor.