LAB 5 // Software

TWO'S COMPLEMENT & ADDERS

INSTRUCTIONS and RUBRIC

CSE 2301 – Fall 2024

INTRODUCTION

Prerequisites

No additional knowledge is required for this lab, but you should be very comfortable with 2's Complement.

Objectives & Background

This is a three-part software lab. Here you will gain some familiarity with one of LogicWorks' special features: the ability to create discrete subcircuit symbols. First you will refresh your memory by building a circuit that adds three unsigned binary values using the 4-bit adder components in LogicWorks. You will then design a small circuit that can make an unsigned binary number negative by converting it to its Two's Complement equivalent. After turning this into a symbol, everything will come together as a large circuit that can add two signed numbers together.

PART 1:

Add three unsigned numbers

YOUR TASK – Create a circuit that adds three unsigned binary numbers. All of your inputs will come from hex keyboards, so they could have any value in the range 0-15. Using the CO (carry-out/overflow) pin on the 4-bit adders you use, you should be able to find a way to display sums of up to 45 with only two adders. Demo.

> HINT: A 4-bit adder produces a 4-bit output, but also an overflow. That overflow is much like a "5th bit" in the sense that it represents a sum greater than or equal to 16. If you added 10 and 10, for example, the adder would display 0100_2 as output (4_{10}) and provide an overflow, which is weighted 16. Therefore, $16 + 4 = 20$. What if you do this twice and overflow both times? How can you determine a sum of greater than or equal to 32? And how would you display it?

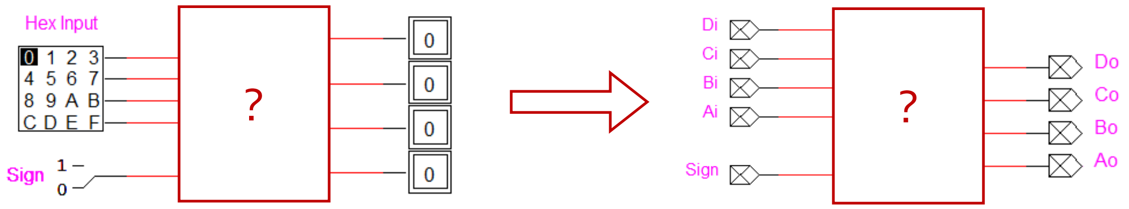
PART 2:

Discrete subcircuit module

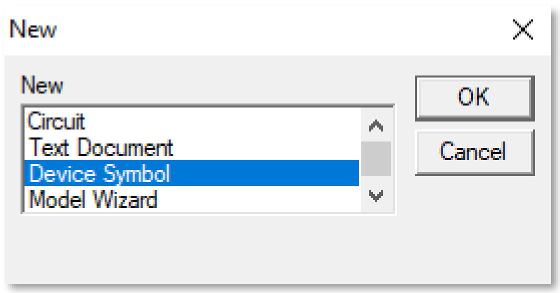
YOUR TASK – Start by designing a small circuit in a new LogicWorks file that takes the input from a hex keyboard and converts it to 2's Complement if it's negative. Indicate the sign via a binary switch. Recall that for a 4-bit value, you can only represent numbers in the range -8 to +7 in 2's Complement, so assume that you never input any greater than a 7 (positive) or 8 (negative) on the hex keyboard.

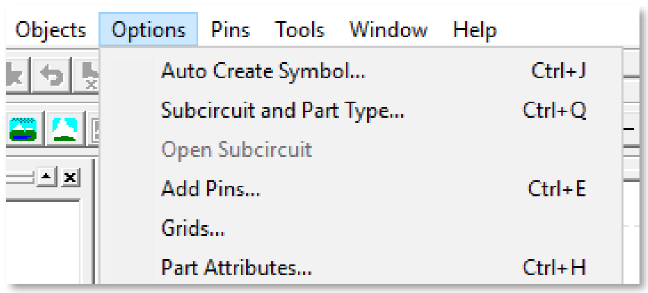
> HINT: How do we convert to 2's complement usually? Also, if you connect a variable input and a binary switch to an XOR, you can use the binary switch to invert the value.

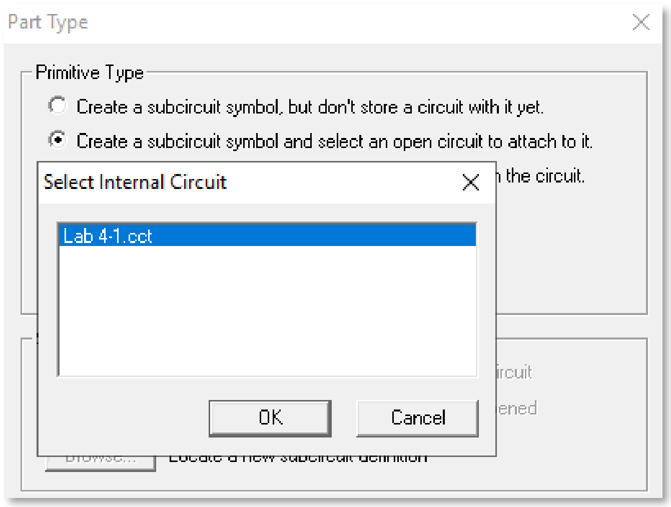
1. Once you have a design in mind, you're going to convert it into a discrete subcircuit. Your circuit should look something like the image below. You'll first need to replace the input and output part of the design with "port in" and "port out" components respectively, and then name them, DCBA in, and DCBA out, for example.



2. Next, you'll need to create a new Device Symbol. Once you've done so, a new interface will appear named "Part1". Go to Options → Subcircuit and Part Type, then select the second option and target file, which in your case will be the open circuit file.




Objects
Options
Pins
Tools
Window
Help
Auto Create Symbol... Ctrl+J
Subcircuit and Part Type... Ctrl+Q
Open Subcircuit
Add Pins... Ctrl+E
Grids...
Part Attributes... Ctrl+H


Part Type

Primitive Type

( ) Create a subcircuit symbol, but don't store a circuit with it yet.

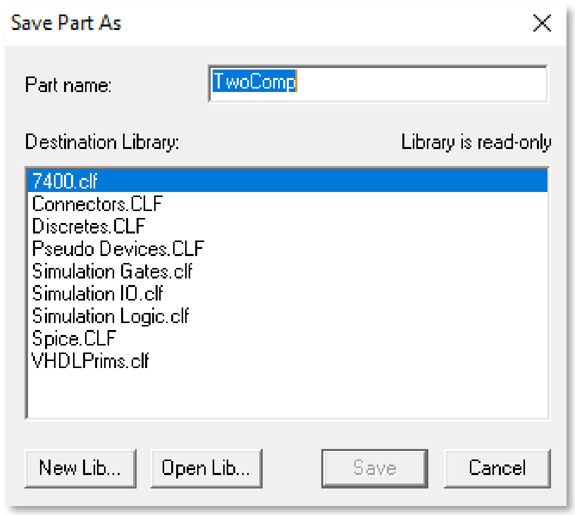
(●) Create a subcircuit symbol and select an open circuit to attach to it.

Select Internal Circuit

Lab 4-1.cct

3. Now go to Options → Auto-Create Symbol. It should list all of your pins. If not, note that there are some range-selection behaviors, which is why you might see something A1-3 if you numbered your ports. Name the part something relevant, like "TwoComp" and

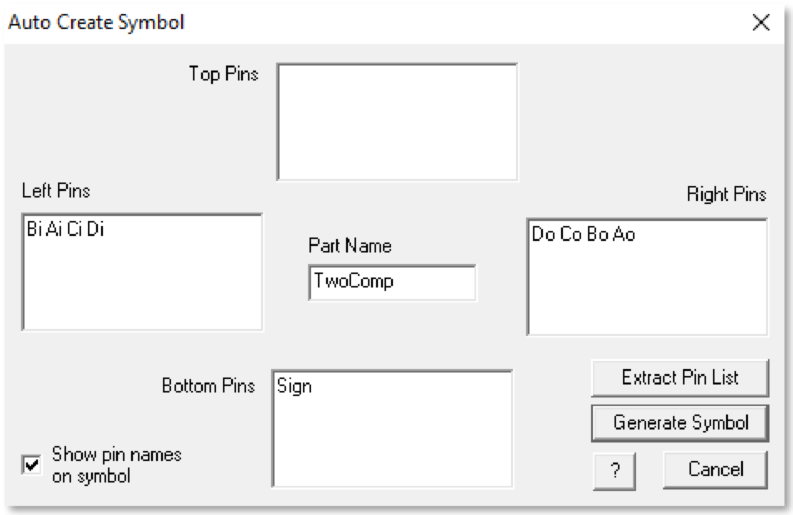
generate the symbol. It will be helpful to ensure than the inputs/outputs are MSB to LSB order.


Save Part As

Part name: TwoComp

Destination Library: Library is read-only

- 7400.clf
- Connectors.CLF
- Discretes.CLF
- Pseudo Devices.CLF
- Simulation Gates.clf
- Simulation IO.clf
- Simulation Logic.clf
- Spice.CLF
- VHDLPrims.clf


Auto Create Symbol

Top Pins:

Left Pins: Bi Ai Ci Di

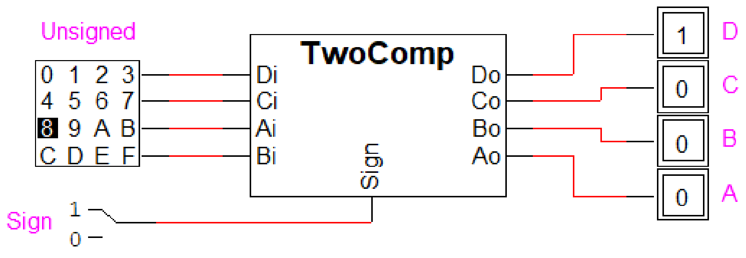
Part Name: TwoComp

Right Pins: Do Co Bo Ao

Bottom Pins: Sign

[✓] Show pin names on symbol

4. Save the symbol with Save As. You'll need to save it to your own library, so create a New Lib and save it to your drive so that it doesn't get erased. Last, select the library you just created, and press save. The library should be loaded automatically, but if it isn't, go to File → Libraries → Open Lib and open the library you just created.



You can now place the component in a file directly and use it whensoever you please without having to rebuild it, copy/paste everywhere, or take up an inordinate amount of space.

PART 3:

Two's Complement Adder

YOUR TASK – Design a circuit that adds two 2's Complement numbers. You will have 4 inputs, including a hex keyboard and binary switch for each signed value, as in part B, and 3 outputs: a hex display for the magnitude of the output, a binary probe to indicate its sign, and a binary probe that detects whether or not overflow occurred. Build another symbol, as in part 2, to detect overflow. Do not test exhaustively.

> HINT: You will need to use the symbol we just created 3 times. Recognize that converting *from* 2's Complement is the same as converting *to* it. When attempting to define overflow, do not

> use the CO pin from the adders – you will need to find the cases where two positive numbers are added and a negative is the result or two negative numbers are added and a positive is the result, and then use combinational logic to solve the problem. What's happening here?

SCORING

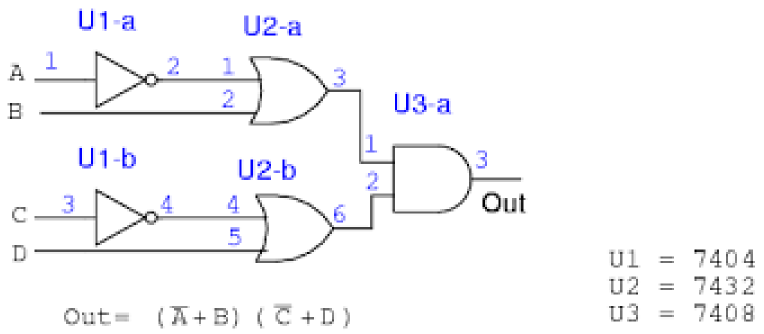
For this lab, you will need to demo both of your circuits (from part 1 and part 3), probably separately. Make sure that you can still access the LogicWorks files remotely and save your data in a secure location to reduce frustration, especially with respect to the symbols you'll be creating.

Demo Requirements [5 pts]:

- ✓ Complete the circuit for three unsigned binary numbers from part one. **Demo (2 pts).**
- ✓ Follow the instructions to develop the 2's Complement Converter as a symbol.
- ✓ Create a new circuit that adds two 2's Complement numbers together and then displays the result in unsigned binary. Develop another symbol that detects overflow from this setup. **Demo (3 pts).**

Report Requirements [4 pts]:

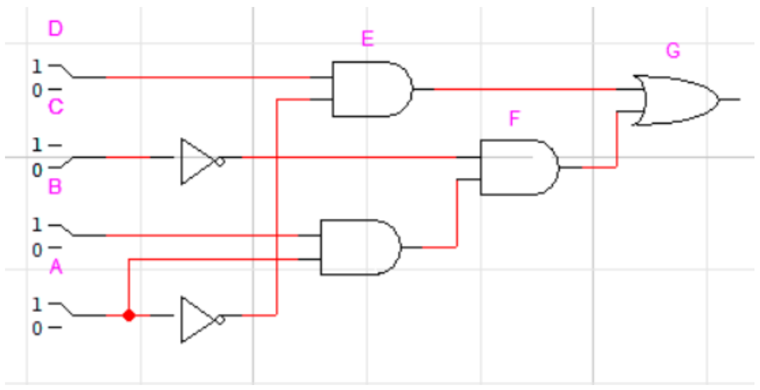
- ✓ [1.2] Theory:
  - o [0.5] What is 2's Complement? How does it work and why do we use it instead of signed binary for digital logic applications?
  - o [0.7] When adding two numbers like -3 and +5 together in 2's Complement, the full-adder will display overflow through the Cout pin. Why is this misleading? How did you actually detect overflow in your circuit (part 3)?
- ✓ [2.2] Deliverables:
  - o [1.0] Return to both of your symbols (namely, the 2's Complement Converter and the Overflow Detector) and label all of the pins and chips that you use. An example is given below. Use UX where X is a number to denote which chip is used, followed by a dash and lowercase letter to denote which gate on the chip is being used. Take pictures of these annotations and include them in your report. Ask TAs for further details if needed.
  - o [0.8] A section including eight tests of your LogicWorks implementation with columns indicating inputs, and the magnitude, sign, and overflow of the output. Draw this in the form of a table. Include one picture of your LogicWorks circuit.
  - o [0.4] How did you select your test cases, since you cannot test exhaustively?
- ✓ [0.6] Discussion section. Should conform to standard lab report guidelines.



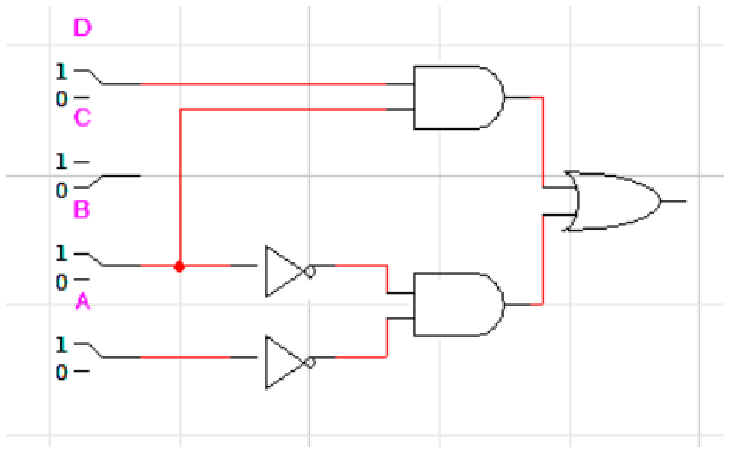
Pin and chip labelling of a simple combinational logic circuit.

Practice Questions [1 pt]:

- ✓ [0.6] Question 1:
  Use the circuit shown below. Assume that the inverters have a delay of 5 ns and the AND/OR gates have a delay of 10 ns. Initially, C = 0 and A = B = D = 1. At time t = 5 ns, A changes to 0. Draw a timing diagram and identify determine if any transients occur.



- ✓ [0.4] Question 2:
  Determine the minterm required to eliminate the static hazard in the circuit shown below. Then modify the circuit to eliminate the hazard by including the gates to create that minterm expression.


D
C
B
A
1
0
1
0
1
0
1
0