Lab 05: Two's Complement and Adders

Arturo Salinas-Aguayo
CSE 2301: Principles and Practice of Digital Logic Design
Dr. Mohammad Khan, Section 003L-1248
Electrical and Computer Engineering Department

College of Engineering, University of Connecticut
Coded in LaTeX

Theory

What is 2's Complement?

Two's Complement is a method of encoding signed integers in binary form, where the most significant bit indicates the sign (0 for positive, 1 for negative). The remaining bits represent the magnitude of the number. To convert a positive number to its Two's Complement negative equivalent, you invert all the bits and add 1 to the result.

We use Two's Complement because it simplifies arithmetic operations like addition and subtraction in digital circuits. Unlike signed-magnitude representation, Two's Complement eliminates the need for separate circuitry to handle negative numbers. This makes it efficient and less prone to error in binary arithmetic operations.

Why the Cout Pin Overflow is Misleading

When adding two numbers in Two's Complement, the Cout pin in a full-adder indicates a carry-out. However, this can be misleading when detecting overflow, especially in signed arithmetic. For example, adding -3 and +5 in Two's Complement could result in a carry-out even though no overflow occurred from a signed arithmetic perspective. Instead of relying on the Cout pin, overflow detection should focus on cases where the signs of the two operands are the same but the result has a different sign. In my circuit for Part 3, I detected overflow by using combinational logic to identify when two positive numbers resulted in a negative number, or two negative numbers resulted in a positive number.

Deliverables

Example 1 ***Labeled Chip and Pin Symbols.***
Here are the labeled symbols for both the 2's Complement Converter and the Overflow Detector. All of the pins and chips are labeled according to the following notation: UX where X is a number to denote the chip, followed by a dash and lowercase letter to denote the gate on the chip.

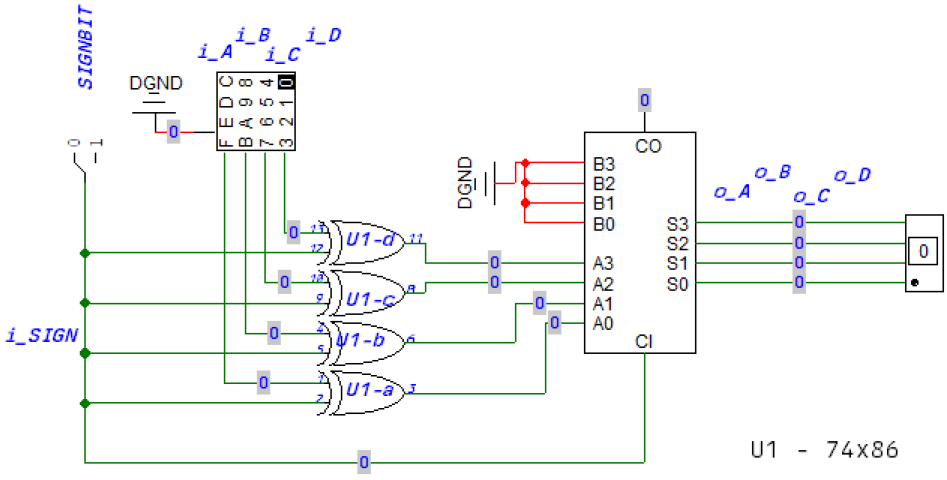


Figure 1: 2's Complement Converter

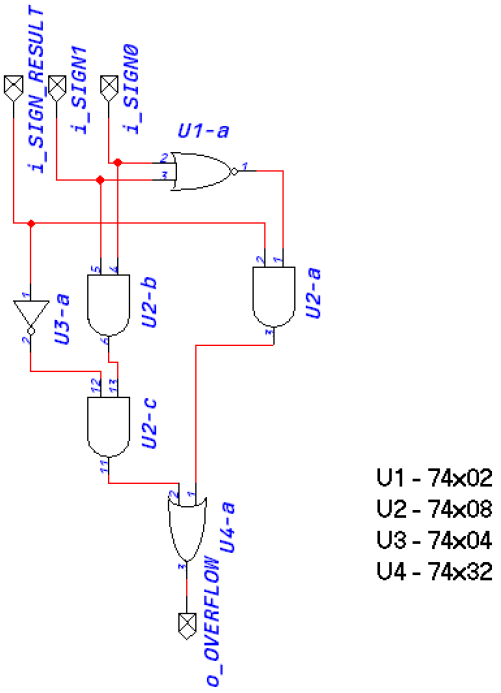


Figure 2: Overflow Detector

Example 2 ***Test Cases***

Below is a table of eight test cases, listing the inputs, and showing the magnitude, sign, and overflow of the outputs. Recall that "Sign of Result" is 1 if the result is negative, and 0 if the result is positive. "Overflow" is 1 if an overflow occurred, and 0 if no overflow occurred.

Input A	Input B	Hex Output	Sign of Result	Overflow
4	6	6	1	1
C	A	6	0	0
3	D	0	0	0
7	7	2	1	1
-7	8	1	0	0
-6	A	4	0	0
-4	-4	8	1	0

Table 1: Test Cases Table

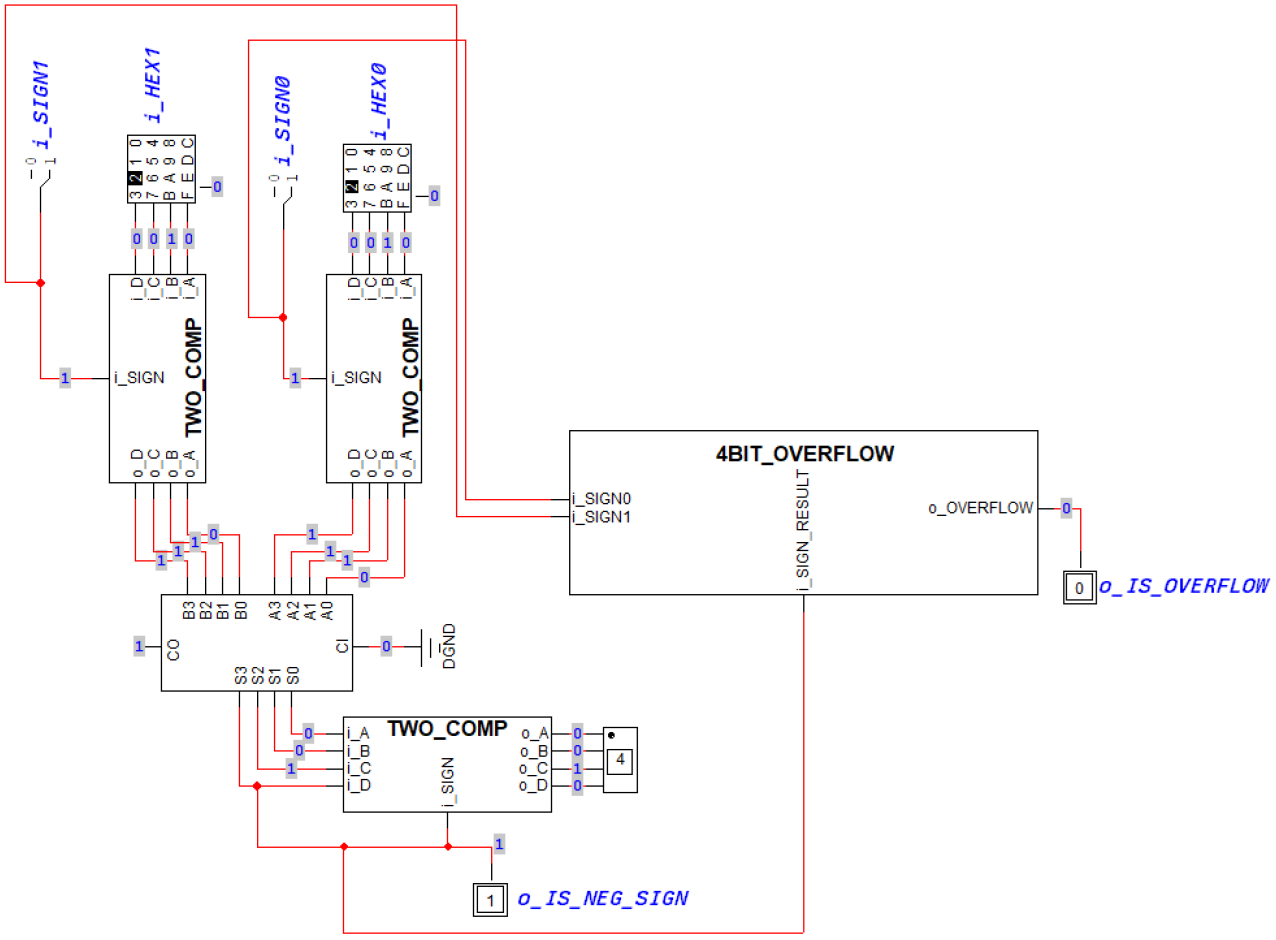


Figure 3: Overflow Detector Circuit

Test Case Selection
The test cases were selected to cover a range of inputs, including both positive and negative numbers, as well as cases with and without overflow. Since exhaustive testing is not feasible, I focused on cases that would trigger different logic paths, such as adding two large positive numbers, two large negative numbers, and combinations that should produce no overflow. Essentially, if I expected a negative and got a positive number, I knew something was awry and overflowed.

Discussion

This lab helped reinforce my understanding of Two's Complement arithmetic and the importance of accurate overflow detection in digital circuits. The process of creating reusable symbols in LogicWorks was particularly valuable, as it streamlined the development of complex circuits. Detecting overflow through combinational logic instead of relying on the Cout pin was a key takeaway, as it highlighted the differences between binary and signed arithmetic. Overall, this lab improved my circuit design skills and my ability to implement Two's Complement in a practical setting.

There were some difficulties in learning the software, but after spending some time reading the reference material (which is sadly incomplete due to the age of the software), some great strides were made in implementing combinational circuitry. Looking forward to the VHDL chapter, I changed my inputs to better make sense when dealing with behaviorial or RTL descriptions of these circuits in the future. That alongisde the ability to specify pins brings the software closer to the hardware implementation.

This also sets us up for the next lab which is a basic ALU implementation, by using the overflow detector and the 2's complement converter, we can now implement the adder and the subtractor in the ALU.

Questions

Example 3 ***Timing Diagram for a Combinational Circuit.***

1. *Initial State (before $t = 5\,ns$): - Inputs are stable: $A = 1$, $B = 1$, $C = 0$, $D = 1$. - Outputs are stable based on these inputs.*

2. *At $t = 5\,ns$: - A changes from 1 to 0. - The change in A propagates through the circuit, starting with the inverters and continuing to the AND/OR gates.*

3. *Inverter Delay: - A', the inverted form of A, takes 5 ns to reflect the change, becoming 1 at $t = 10\,ns$.*

4. *AND/OR Gate Delay: - Gates dependent on A' or A (such as signals H, E, and F) will take an additional 10 ns to reflect the change. Thus, the outputs of these gates begin to change at $t = 15\,ns$.*

- ***Transient Behavior:*** *A transient (or glitch) occurs when the output temporarily toggles due to the varying delays in the circuit.*
- ***Transients Observed:*** *The timing diagram shows transients for signals H, E, F, and G between $t = 5\,ns$ and $t = 15\,ns$, caused by the delayed propagation of the input changes through the inverters and gates.*

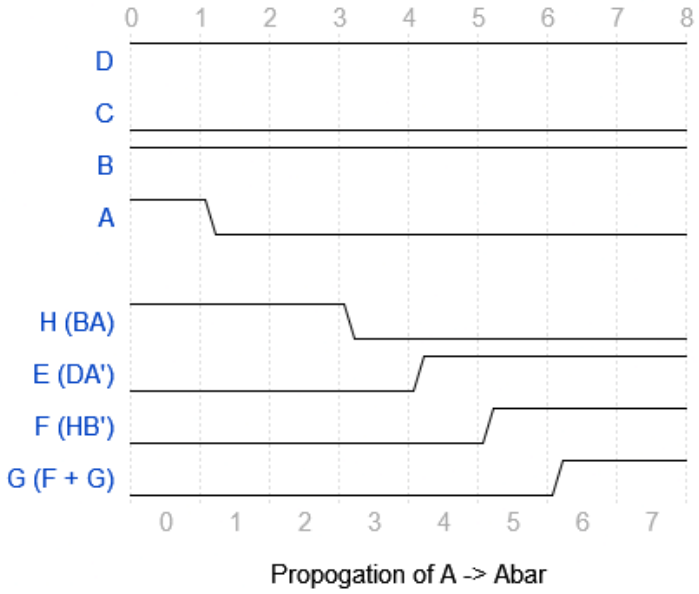


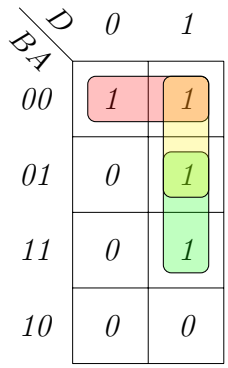
Figure 4: Timing Diagram

Question 2

Example 4 ***Static 1-Hazard Elimination.***
Consider the Boolean function $F = (\overline{A}\overline{B}) + BD$, which is a two-level AND-OR combinational circuit. A static 1-hazard occurs if, in response to a single input change, the output momentarily goes to 0 when it should remain constant at 1. To eliminate this hazard, we follow these steps.

Step 1: Identifying the Static 1-Hazard. *The static 1-hazard in the circuit can be detected by examining adjacent minterms on a Karnaugh map (K-map) for the function F. Adjacent 1's that are not covered by a single loop indicate the presence of a hazard. In this case, the minterms $\overline{A}\overline{B}$ and BD do not cover all the transitions between the states where the output should remain constant.*

The K-map for the function $F = \sum m(0, 1, 3, 7)$ is shown below. The red and green implicant circles represent the two groups of minterms that cover the function. The static 1-hazard occurs between the minterms $(\overline{A}\overline{B})$ and BD. The yellow implicant is the missing minterm that causes the hazard. To fix the static-1 hazard, we need to add this minterm to the expression.



Step 2: Adding a Minterm to Eliminate the Hazard.

By analyzing the K-map, we see that the yellow implicant circle overlaps with the other two circles. that the minterm AB is missing. Adding this minterm ensures that the circuit transitions smoothly without introducing any glitches. The new expression becomes:

$$F = (\overline{A}\overline{B}) + BD + AB$$

This eliminates the static 1-hazard by covering the transition between $(\overline{A}\overline{B})$ *and* BD.