SW 2 - 4-Bit Adder (Two's Complement)

Jonathan Schwartz

Section 003L

TA: Conner Rickermann

Date submitted: 2/25/2021

**Objective**

The objective of this lab was to add two signed 4-bit numbers using two’s complement and detect whether there was an overflow error anywhere in the computation. The range of the numbers is [-8,7], and the output must also be in this range. We use two’s complement because it is more effective for accounting for negative numbers given that there is no -0, and the computation is still simple.

**Background and Theory**

Adding two numbers in binary without two’s complement can be difficult because in one’s complement there is a negative 0 number and -8 is excluded. With signed magnitude, it is difficult to add two numbers of opposite signs. In 4-bit two’s complement circuits, the DCBA bits are weighted -8, 4, 2, and 1 respectively. For this circuit, adding with two’s complement requires converting the two signed magnitude numbers into two’s complement, then adding them, then converting back to signed magnitude. At each step, it is important to determine if the result is in the domain of the circuit, for example, inputs must be between [-8,7], inclusive, and the sum of the numbers must be in this range as well.

The first overflow condition, during the signed magnitude to two’s complement conversion, would be true if the original number should be negative but the output has a 0 in the “D” bit or vice versa, so this must be checked. The other condition would be if the resulting sign of the sum of the numbers is opposite what the original numbers are ***if*** the original numbers had the same sign. If any of these conditions are met, the bit representing an error will hold the value of 1, otherwise, if nothing goes wrong, it will hold the value of a 0.

**Results**

Conversion from signed magnitude to two’s complement

|  | D | C | B | A |  | D2 | C2 | B2 | A2 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input: | 1 | 1 | 1 | 1 | Output: | 1 | 0 | 0 | 0 |
| Input: | 1 | 1 | 1 | 0 | Output: | 1 | 0 | 0 | 1 |
| Input: | 1 | 1 | 0 | 1 | Output: | 1 | 0 | 1 | 0 |
| Input: | 1 | 1 | 0 | 0 | Output: | 1 | 0 | 1 | 1 |
| Input: | 1 | 0 | 1 | 1 | Output: | 1 | 1 | 0 | 1 |
| Input: | 1 | 0 | 1 | 0 | Output: | 1 | 1 | 1 | 0 |
| Input: | 1 | 0 | 0 | 1 | Output: | 1 | 1 | 1 | 1 |
| Input: | 0 | 0 | 0 | 0 | Output: | 0 | 0 | 0 | 0 |
| Input: | 0 | 0 | 0 | 1 | Output: | 0 | 0 | 0 | 1 |
| Input: | 0 | 0 | 1 | 0 | Output: | 0 | 0 | 1 | 0 |
| Input: | 0 | 0 | 1 | 1 | Output: | 0 | 0 | 1 | 1 |
| Input: | 0 | 1 | 0 | 0 | Output: | 0 | 1 | 0 | 0 |
| Input: | 0 | 1 | 0 | 1 | Output: | 0 | 1 | 0 | 1 |
| Input: | 0 | 1 | 1 | 0 | Output: | 0 | 1 | 1 | 0 |
| Input: | 0 | 1 | 1 | 1 | Output: | 0 | 1 | 1 | 1 |

In the actual circuit, to accomplish the conversion, we add two numbers, so there is an error thrown when a number outside the range [-8,7] is inputted or -0 is inputted. This error is picked up in the error bit initially and reported at the end.

Then, once the numbers are added together in two’s complement form, the same test is run to see if the number is out of bounds by the method described in the theory section.

For any input below -8 or above 7, or equal to -0, the error bit will produce a 1. Any input that results in a number below -8 or greater than 7 also produces a 1 in the error bit.

For results that don’t error, the signed magnitude output will have the sign bit, as well as a 4-bit representation of the magnitude (-8 has a magnitude that requires the fourth bit), and every output works fully.

**Discussion and Conclusion**

Throughout the lab, I had two major issues after implementation. First, there should not be a ground on the carry of a 4-bit adder, because then it will not return the correct values. The second issue was how to gather whether the sum was out of the range of the circuit. I decided to use an XNOR gate in conjunction with an AND gate and an OR gate to compare the sign of the inputs and the output. I have a working circuit that returns whether there was an error as well as the correct results, but the error bit does not prevent an output altogether, so the circuit would still transfer incorrect data unless this error bit is accounted for wherever this added is used.

**Questions**

Since you cannot, in practice, test all digital circuits exhaustively, how would decide which tests to make?

You would definitely test whether you get an expected output for non-edge cases such as 1+1 or 2+(-3), then you would test edge cases for each input, such as -8+(-1), or 7+1 and -0 as an input or a number below -8 or above 7 as an input. It wouldn’t be essential to test 1+2 and 2+1, but it would be essential to test the edge cases in each input.