CS 3410 Design Document

Michael Aspinwall

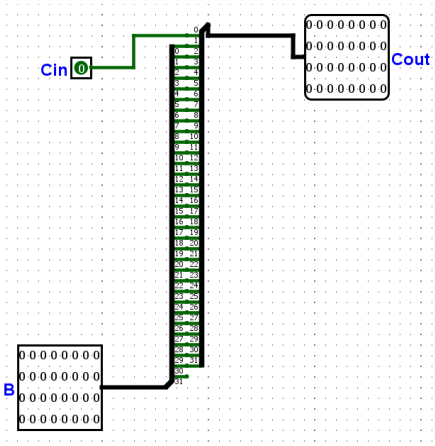
2/10/2019

1) Overview

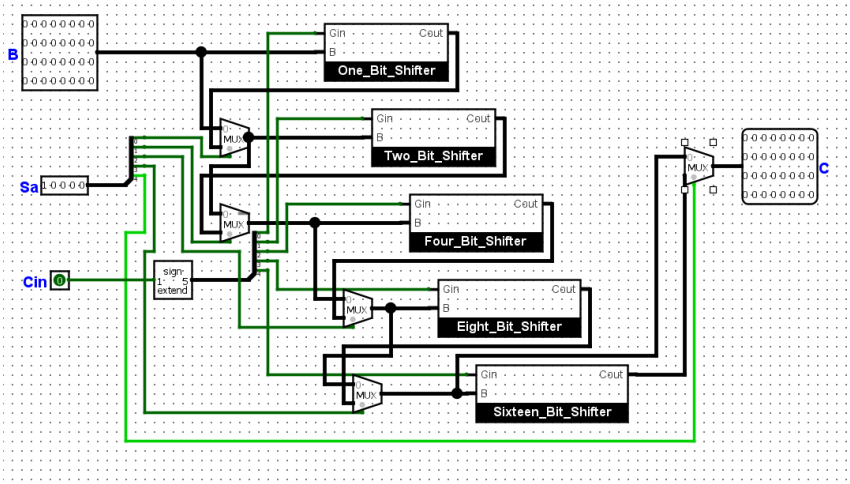
This assignment required the creation of an arithmetic logic unit that could handle 32-bit integers. The difficulty of the task was broken down into sub-circuits. The largest frame was the Alu itself. The next two largest circuits are the shifter and adder. Each of those were constructed from smaller adders and shifters that chained together to make the 32bit version.

2) 32-bit shifter

My implementation started with a one-bit shifter.



Using a similar structure but differing the number of offed bits I constructed 2, 4, 8, and 16 shifters. At the highest-level Sa was used to select between running B through each of these.



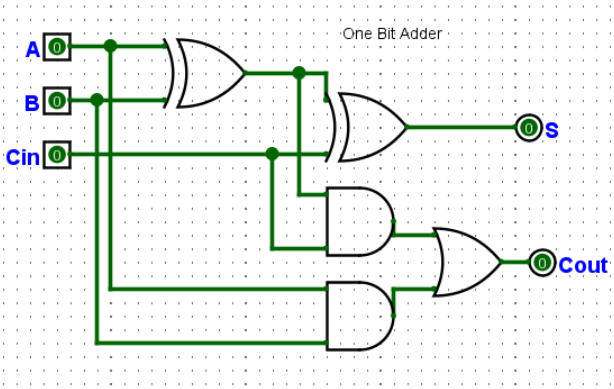
Using the 1, 2, 4, 8, and 16 bits of Sa I Mux'd between an "untouched" B and the shifted version. I replaced the shifted bits with the input to Cin.

Design justification

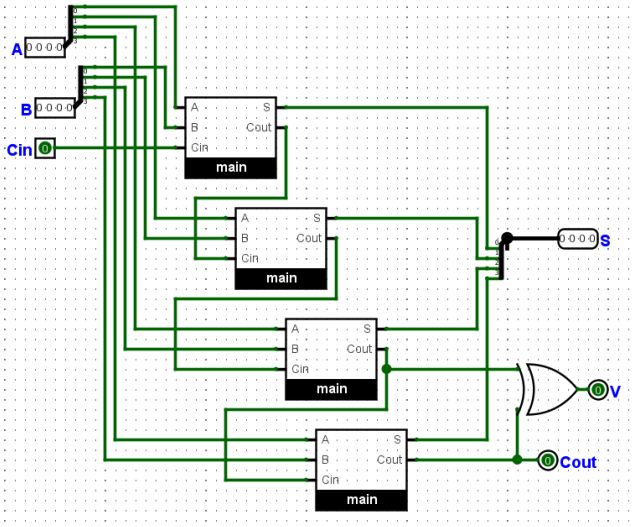
Using the multiplexers was perhaps not the most efficient system for selection. But as a trade of for gate amount my solution gains clarity. The mux's clearly show the control of the circuit.

3) 32-bit full adder

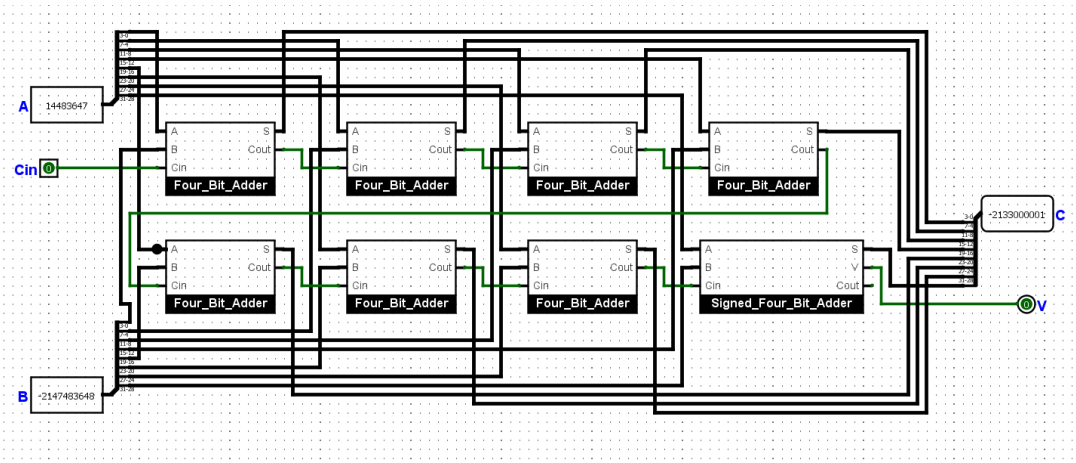
My implementation started with a one-bit adder.



A one-bit full adder was designed to support the addition of two 1-bit inputs with a carry bit. This circuit was necessary for the construction of the more advanced full 32-bit adder. Using this as a building block I chained four together to create a 4-bit adder. These adders were unsigned that i.e. didn't not give off the information necessary to determine if overflow had occurred.



To remedy this, I created a signed 4-bit adder which outputs the Xor of the most two final Cout's. Putting this sub-circuit at the end of the chain I was able to detect overflow.

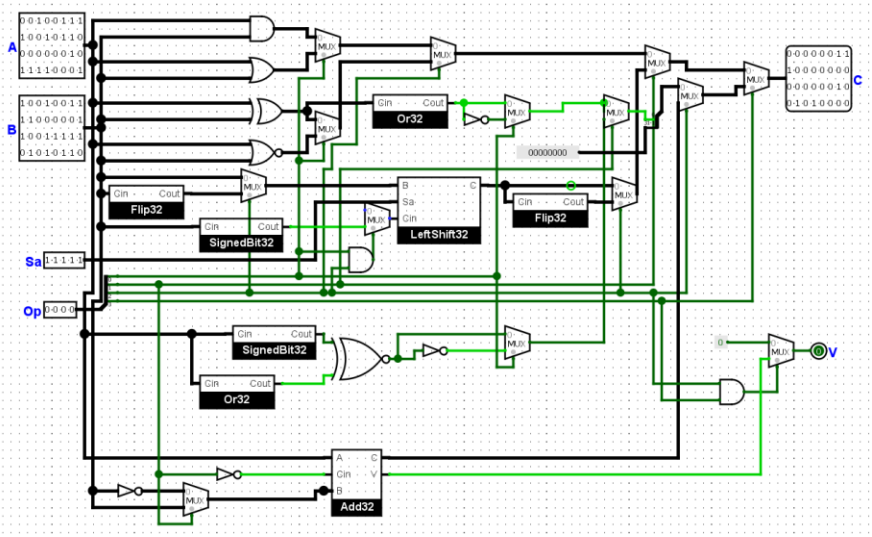


Design justification

My circuit implements the optimal logical expression for both the resulting bit and the carry-out bit. The wiring for the largest 32-bit level could have been wired more clearly had I created an 8-bit sub-circuit, but the two would are logically equivalent so this is not so important.

4) 32-bit ALU

Beginning this aspect of the problem I struggled with the control flow using Op.



My implementation took advantage of the separation of similar operations in order of least significant bit to most. For example, I distinguished between and/or by muxing between their outputs based on the least significant bit. In addition, I had to create three additional sub-circuits Or32, SignedBit32 and Flip32. Or32 or'd together 32 bits leading to one output. This allowed me to detect for any nonzero words. SignedBit32 outputs the most significant bit of an input. Flip32 flips the stream of the word input.