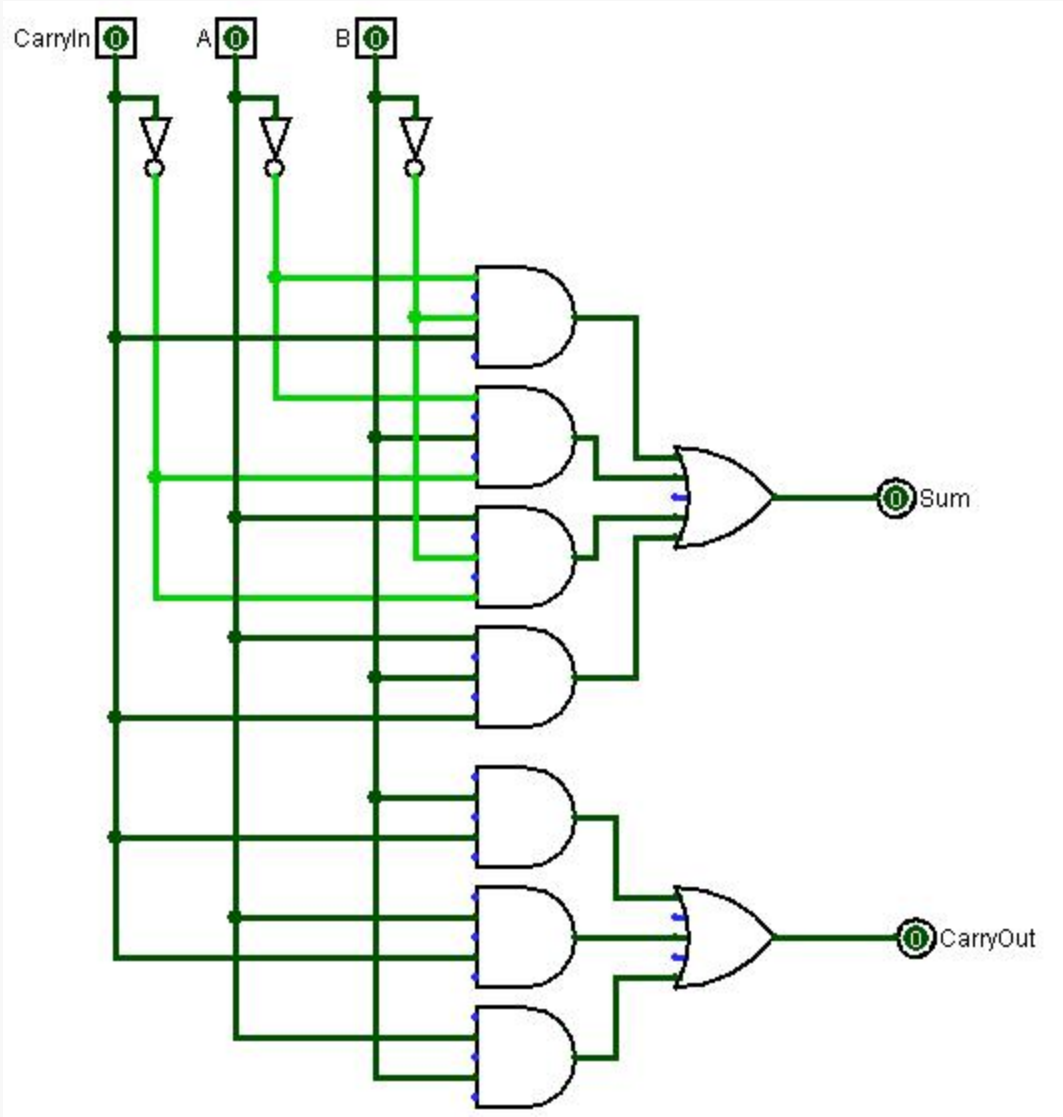
1-bit Full Adder

The expressions for the sum and carry lead to the following unified implementation:

$$Sum = \overline{A} \cdot \overline{B} \cdot C_{in} + \overline{A} \cdot B \cdot \overline{C_{in}}$$
$$+ A \cdot \overline{B} \cdot \overline{C_{in}} + A \cdot B \cdot C_{in}$$

$$Carry = B \cdot C + A \cdot C + A \cdot B$$

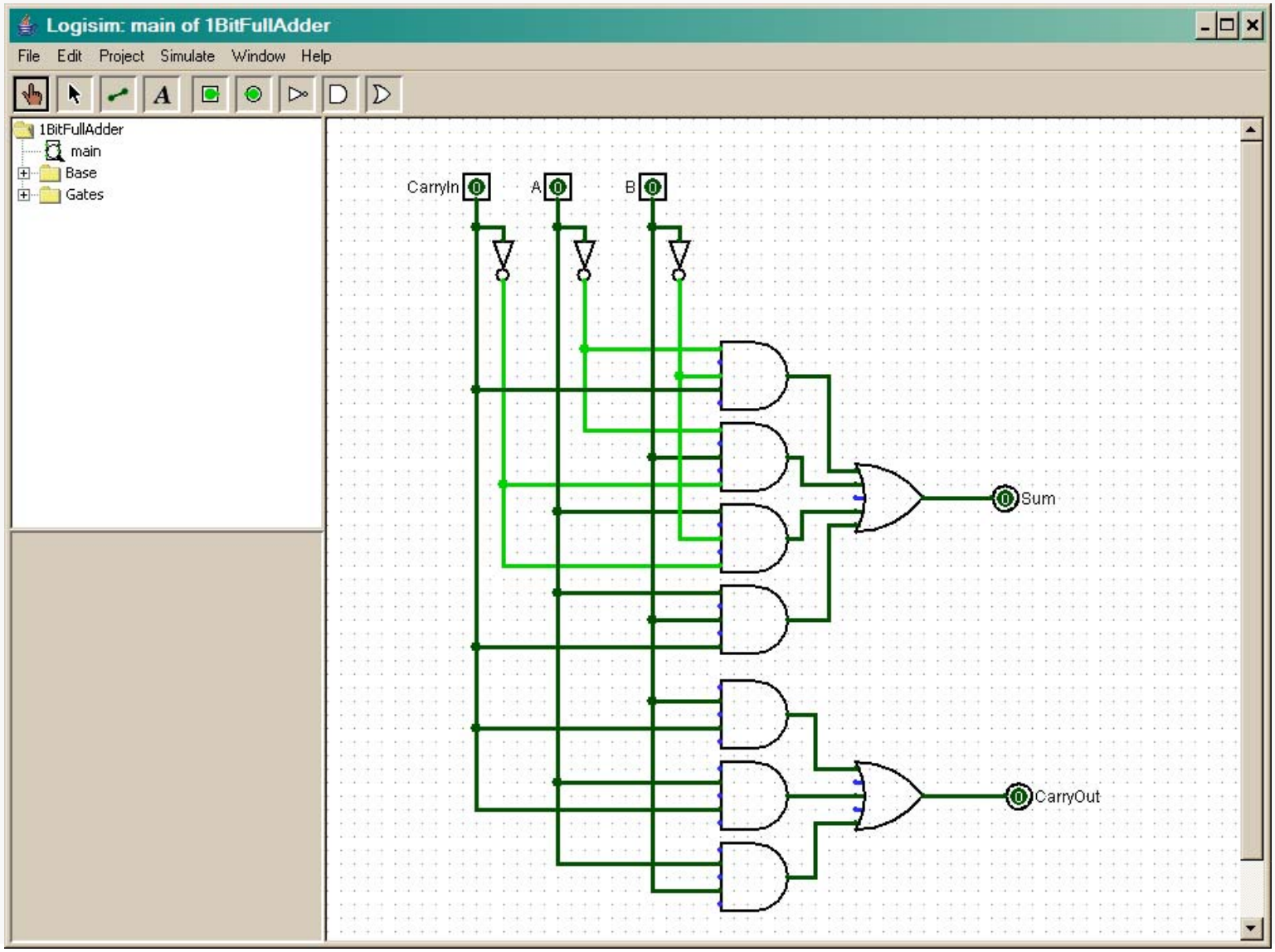


This implementation requires only two levels of logic (ignoring the inverters as customary).

Is there an alternative design that requires fewer AND/OR gates? If so, how many levels does it require?

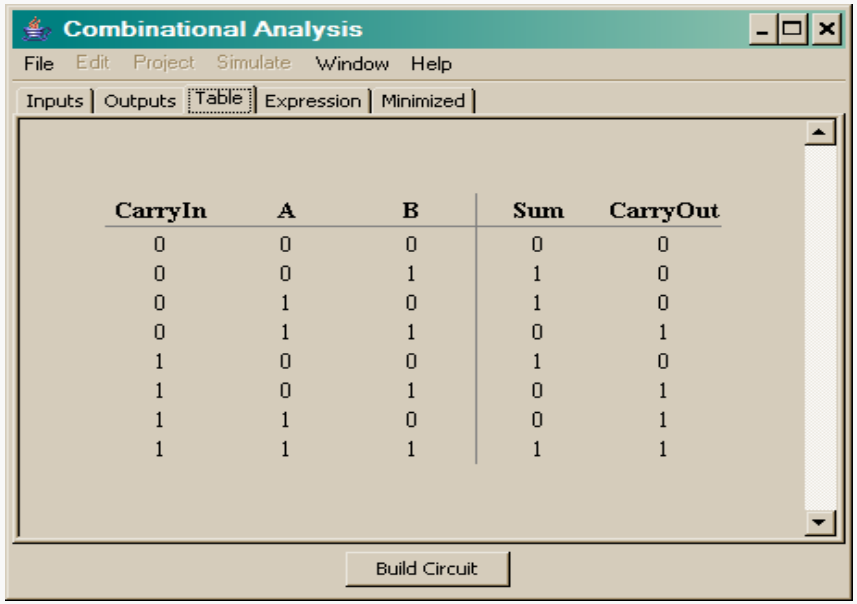
LogiSim Implementation

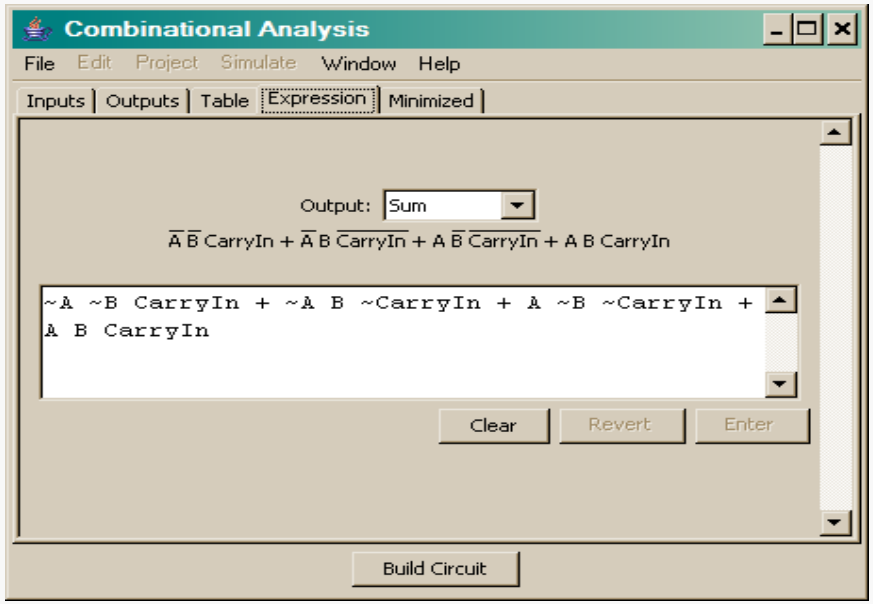
The previous circuit is easily implemented in LogiSim.



LogiSim Analysis Tool

The Project menu contains an option to analyze the circuit.

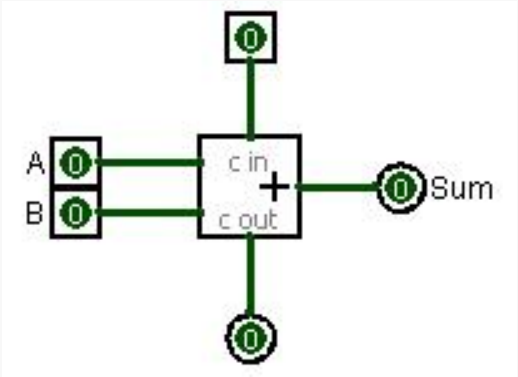




You may find these useful in verifying the correctness of a circuit.

1-bit Full Adder as a Module

When building more complex circuits, it is useful to consider sub-circuits as individual, "black-box" modules. For example:

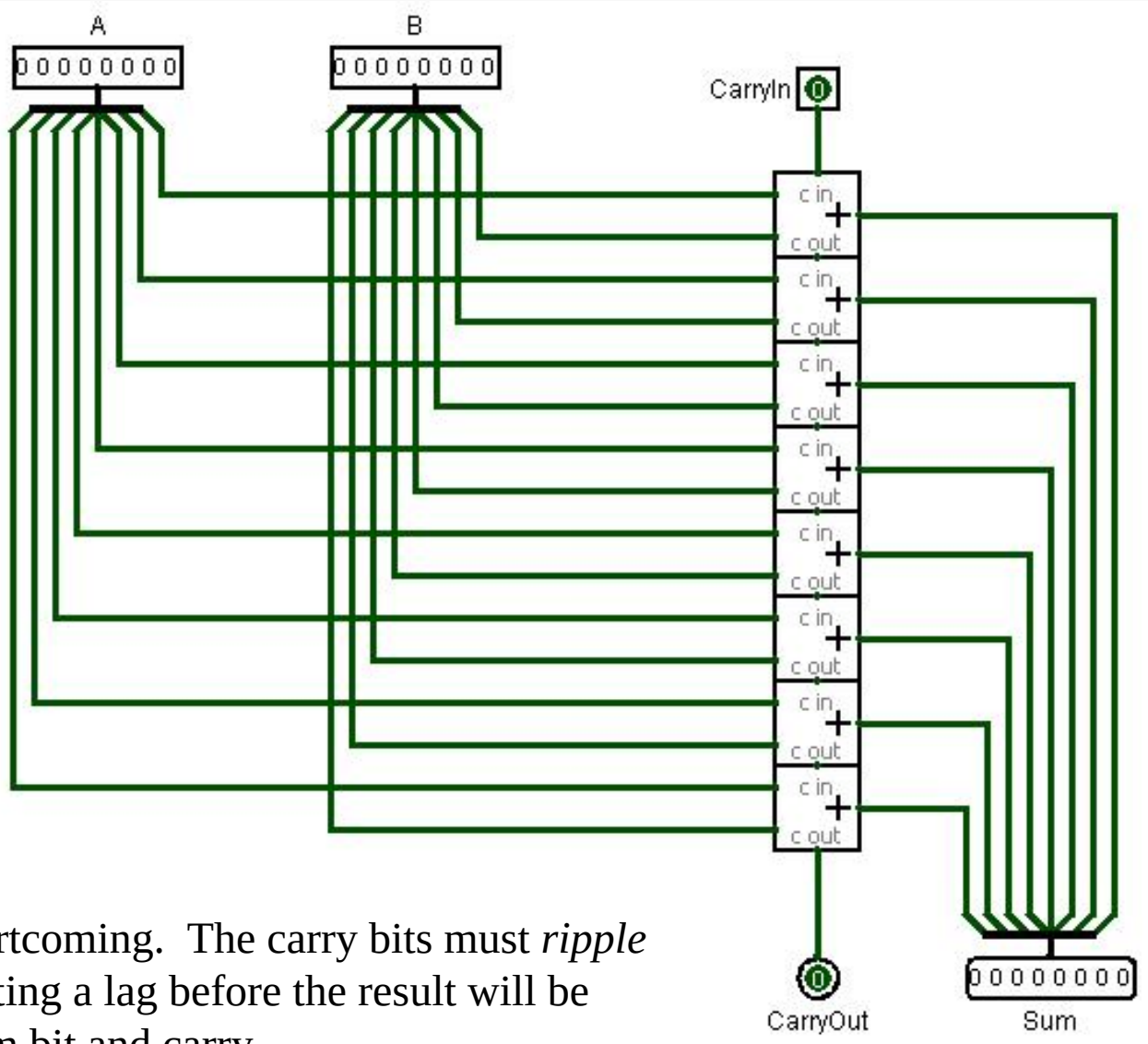


$$Sum = \overline{A} \cdot \overline{B} \cdot C_{in} + \overline{A} \cdot B \cdot \overline{C_{in}} + A \cdot \overline{B} \cdot \overline{C_{in}} + A \cdot B \cdot C_{in}$$

$$Carry = B \cdot C + A \cdot C + A \cdot B$$

Chaining an 8-bit Adder

An 8-bit adder build by chaining 1-bit adders:



This has one serious shortcoming. The carry bits must *ripple* from top to bottom, creating a lag before the result will be obtained for the final sum bit and carry.

Computing the Carry Bits

Any Boolean function can be computed using two levels of logic gates (not counting inverters)… why?

Therefore, it should be possible to compute all of the carry bits needed by the preceding 8-bit adder using only two levels of logic gates, and that should provide a way to speed up the computation of the final result…

Let's consider the problem of adding two 8-bit operands, A and B, and label the bits of each as:

$$a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0 + b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0$$

Similarly, label the relevant carry bits as:

$$c_8 c_7 c_6 c_5 c_4 c_3 c_2 c_1 c_0$$

Normally 0!

And finally, label the sum bits as:

$$s_7 s_6 s_5 s_4 s_3 s_2 s_1 s_0$$

Computing the Carry Bits

We can derive some interesting (Boolean) results for the carry bits:

$$c_1 = b_0 \cdot c_0 + a_0 \cdot c_0 + a_0 \cdot b_0$$

$$c_2 = b_1 \cdot c_1 + a_1 \cdot c_1 + a_1 \cdot b_1$$

But… this shows c_2 as dependent on first knowing c_1… which is the ripple carry logic.

However, if we substitute we can show:

$$\begin{aligned} c_2 &= b_1 \cdot c_1 + a_1 \cdot c_1 + a_1 \cdot b_1 \\ &= b_1 \cdot (b_0 \cdot c_0 + a_0 \cdot c_0 + a_0 \cdot b_0) + a_1 \cdot (b_0 \cdot c_0 + a_0 \cdot c_0 + a_0 \cdot b_0) + a_1 \cdot b_1 \\ &= a_1 \cdot b_0 \cdot c_0 + a_1 \cdot a_0 \cdot c_0 + a_1 \cdot a_0 \cdot b_0 + b_1 \cdot b_0 \cdot c_0 + b_1 \cdot a_0 \cdot c_0 + b_1 \cdot a_0 \cdot b_0 + a_1 \cdot b_1 \end{aligned}$$

And that gives us c_2 in two levels of logic gates… but imagine how complex the formulas would be for the higher-order carry bits…

Abstraction

Take the original (generalized) carry equation and factor it:

$$\begin{aligned} c_{i+1} &= b_i \cdot c_i + a_i \cdot c_i + a_i \cdot b_i \\ &= a_i \cdot b_i + (a_i + b_i) \cdot c_i \end{aligned}$$

Then, if we rewrite the formula for c_2, we have:

$$\begin{aligned} c_2 &= a_1 \cdot b_1 + (a_1 + b_1) \cdot (a_0 \cdot b_0 + (a_0 + b_0) \cdot c_0) \\ &= g_1 + p_1 \cdot (g_0 + p_0 \cdot c_0) \end{aligned}$$

where:

$$g_1 = a_1 \cdot b_1 \text{ and } p_1 = a_1 + b_1$$

We call the latter terms the *generate* and *propagate* factors.

Generate and Propagate

We can then express the general carry bit formula: $c_{i+1} = g_i + p_i \cdot c_i$

The formula indicates that if $g_i = 1$ then the adder generates a carry-out no matter what the value of the carry-in c_i might be.

Moreover, if $p_i = 1$, then if there is a carry-in, the adder propagates a carry-out to the next position no matter what the generate bit g_i might be.

So we can show that:

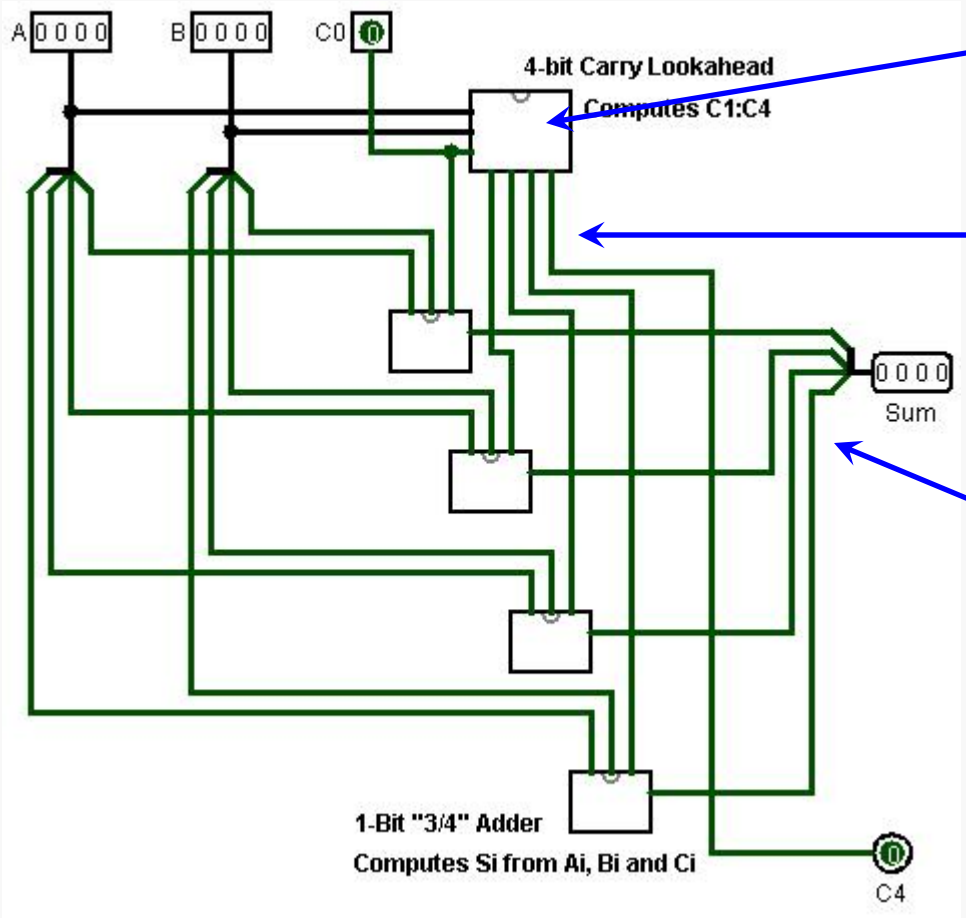
$$c_1 = g_0 + p_0 \cdot c_0$$
$$c_2 = g_1 + p_1 \cdot c_1 = g_1 + p_1 \cdot g_0 + p_1 \cdot p_0 \cdot c_0$$
$$c_3 = g_2 + p_2 \cdot c_2 = g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_0$$
$$c_4 = g_3 + p_3 \cdot c_3 = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 c_0$$

Now, note that we can calculate all the g's and p's in one level of logic from the bits of the two operands, so we can calculate these four carry bits in just three levels of logic.

But… this will still get very complex if we try to find higher-order carry bits…

4-bit Fast-Carry Adder

We can build a 4-bit adder using the fast-carry logic shown on the previous slide:



Computes the four carry bits in three levels of logic…

…carry bits are then fed to the four adders…

…which then compute the sum bits

A Higher-level Abstraction

The formulas grow too complex to make this approach ideal for building a practical (wider) adder… however, we can consider cascading four of the 4-bit fast-carry adders to implement a 16-bit adder.

To do this, we must consider the carry bits that must be generated for each of the 4-bit adders.

We can adapt the approach used above to create a higher-level fast-carry logic unit to generate those carry bits quickly as well.

See Appendix C.6 in P&H for the details.