Here's another approach to improving the latency of our adder, this time focusing just on the carry logic.

Early on in the course, we learned that by going from a chain of logic gates to a tree of logic gates, we could go from a linear latency to a logarithmic latency.

Let's try to do that here.

We'll start by rewriting the equations for the carry-out from the full adder module.

The final form of the rewritten equation has two terms.

The G, or generate, term is true when the inputs will cause the module to generate a carry-out right away, without having to wait for the carry-in to arrive.

The P, or propagate, term is true if the module will generate a carry-out only if there's a carry-in.

So there only two ways to get a carry-out from the module: it's either generated by the current module or the carry-in is propagated from the previous module.

Actually, it's usual to change the logic for the P term from "A OR B" to "A XOR B".

This doesn't change the truth table for the carry-out but will allow us to express the sum output as "P XOR carry-in".

Here's the schematic for the reorganized full adder module.

The little sum-of-products circuit for the carry-out can be implemented using 3 2-input NAND gates, which is a bit more compact than the implementation for the three product terms we suggested in Lab 2.

Time to update your full adder circuit!

Now consider two adjacent adder modules in a larger adder circuit: we'll use the label H to refer to the high-order module and the label L to refer to the low-order module.

We can use the generate and propagate information from each of the modules to develop equations for the carry-out from the pair of modules treated as a single block.

We'll generate a carry-out from the block when a carry-out is generated by the H module, or when a carry-out is generated by the L module and propagated by the H module.

And we'll propagate the carry-in through the block only if the L module propagates its carry-in to the intermediate

carry-out and H module propagates that to the final carry-out.

So we have two simple equations requiring only a couple of logic gates to implement.

Let's use these equations to build a generate-propagate (GP) module and hook it to the H and L modules as shown.

The G and P outputs of the GP module tell us under what conditions we'll get a carry-out from the two individual modules treated as a single, larger block.

We can use additional layers of GP modules to build a tree of logic that computes the generate and propagate logic for adders with any number of inputs.

For an adder with N inputs, the tree will contain a total of N-1 GP modules and have a latency that's order log(N).

In the next step, we'll see how to use the generate and propagate information to quickly compute the carry-in for each of the original full adder modules.

Once we're given the carry-in C_0 for the low-order bit, we can hierarchically compute the carry-in for each full adder module.

Given the carry-in to a block of adders, we simply pass it along as the carry-in to the low-half of the block.

The carry-in for the high-half of the block is computed the using the generate and propagate information from the low-half of the block.

We can use these equations to build a C module and arrange the C modules in a tree as shown to use the C_0 carry-in to hierarchically compute the carry-in to each layer of successively smaller blocks, until we finally reach the full adder modules.

For example, these equations show how C4 is computed from C0, and C6 is computed from C4.

Again the total propagation delay from the arrival of the C_0 input to the carry-ins for each full adder is order log(N).

Notice that the G_L and P_L inputs to a particular C module are the same as two of the inputs to the GP module in the same position in the GP tree.

We can combine the GP module and C module to form a single carry-lookahead module that passes generate and propagate information up the tree and carry-in information down the tree.

The schematic at the top shows how to wire up the tree of carry-lookahead modules.

And now we get to the payoff for all this hard work!

The combined propagation delay to hierarchically compute the generate and propagate information on the way up and the carry-in information on the way down is order log(N), which is then the latency for the entire adder since computing the sum outputs only takes one additional XOR delay.

This is a considerable improvement over the order N latency of the ripple-carry adder.

A final design note: we no longer need the carry-out circuitry in the full adder module, so it can be removed.

Variations on this generate-propagate strategy form the basis for the fastest-known adder circuits.

If you'd like to learn more, look up "Kogge-Stone adders" on Wikipedia.