On our to-do list from the previous section is figuring out how to build AND and OR gates with many inputs.

These will be needed when creating circuit implementations using a sum-of-products equation as our template.

Let's assume our gate library only has 2-input gates and figure how to build wider gates using the 2-input gates as building blocks.

We'll work on creating 3- and 4-input gates, but the approach we use can be generalized to create AND and OR gates of any desired width.

The approach shown here relies on the associative property of the AND operator.

This means we can perform an N-way AND by doing pair-wise ANDs in any convenient order.

The OR and XOR operations are also associative, so the same approach will work for designing wide OR and XOR circuits from the corresponding 2-input gate.

Simply substitute 2-input OR gates or 2-input XOR gates for the 2-input AND gates shown below and you're good to go!

Let's start by designing a circuit that computes the AND of three inputs A, B, and C.

In the circuit shown here, we first compute (A AND B), then AND that result with C.

Using the same strategy, we can build a 4-input AND gate from three 2-input AND gates.

Essentially we're building a chain of AND gates, which implement an N-way AND using N-1 2-input AND gates.

We can also associate the four inputs a different way: computing (A AND B) in parallel with (C AND D), then combining those two results using a third AND gate.

Using this approach, we're building a tree of AND gates.

Which approach is best: chains or trees?

First we have to decide what we mean by "best".

When designing circuits we're interested in cost, which depends on the number of components, and performance, which we characterize by the propagation delay of the circuit.

Both strategies require the same number of components since the total number of pair-wise ANDs is the same in

both cases.

So it’s a tie when considering costs.

Now consider propagation delay.

The chain circuit in the middle has a tPD of 3 gate delays, and we can see that the tPD for an N-input chain will be N-1 gate delays.

The propagation delay of chains grows linearly with the number of inputs.

The tree circuit on the bottom has a tPD of 2 gates, smaller than the chain.

The propagation delay of trees grows logarithmically with the number of inputs.

Specifically, the propagation delay of tree circuits built using 2-input gates grows as log2(N).

When N is large, tree circuits can have dramatically better propagation delay than chain circuits.

The propagation delay is an upper bound on the worst-case delay from inputs to outputs and is a good measure of performance assuming that all inputs arrive at the same time.

But in large circuits, A, B, C and D might arrive at different times depending on the tPD of the circuit generating each one.

Suppose input D arrives considerably after the other inputs.

If we used the tree circuit to compute the AND of all four inputs, the additional delay in computing Z is two gate delays after the arrival of D.

However, if we use the chain circuit, the additional delay in computing Z might be as little as one gate delay.

The moral of this story: it’s hard to know which implementation of a subcircuit, like the 4-input AND shown here, will yield the smallest overall tPD unless we know the tPD of the circuits that compute the values for the input signals.

In designing CMOS circuits, the individual gates are naturally inverting, so instead of using AND and OR gates, for the best performance we want to the use the NAND and NOR gates shown here.

NAND and NOR gates can be implemented as a single CMOS gate involving one pullup circuit and one pulldown circuit.

AND and OR gates require two CMOS gates in their implementation, e.g., a NAND gate followed by an INVERTER.

We'll talk about how to build sum-of-products circuitry using NANDs and NORs in the next section.

Note that NAND and NOR operations are not associative: NAND(A,B,C) is not equal to NAND(NAND(A,B),C).

So we can't build a NAND gate with many inputs by building a tree of 2-input NANDs.

We'll talk about this in the next section too!

We've mentioned the exclusive-or operation, sometimes called XOR, several times.

This logic function is very useful when building circuitry for arithmetic or parity calculations.

As you'll see in Lab 2, implementing a 2-input XOR gate will take many more NFETs and PFETs than required for a 2-input NAND or NOR.

We know we can come up with a sum-of-products expression for any truth table and hence build a circuit implementation using INVERTERs, AND gates, and OR gates.

It turns out we can build circuits with the same functionality using only 2-INPUT NAND gates.

We say the the 2-INPUT NAND is a universal gate.

Here we show how to implement the sum-of-products building blocks using just 2-input NAND gates.

In a minute we'll show a more direct implementation for sum-of-products using only NANDs, but these little schematics are a proof-of-concept showing that NAND-only equivalent circuits exist.

2-INPUT NOR gates are also universal, as shown by these little schematics.

Inverting logic takes a little getting used to, but its the key to designing low-cost high-performance circuits in CMOS.