Let's see if we can improve the throughput of the original combinational multiplier design.

We'll use our patented pipelining process to divide the processing into stages with the expectation of achieving a smaller clock period and higher throughput.

The number to beat is approximately 1 output every 2N, where N is the number of bits in each of the operands.

Our first step is to draw a contour across all the outputs.

This creates a 1-pipeline, which gets us started but doesn't improve the throughput.

Let's add another contour, dividing the computations about in half.

If we're on the right track, we hope to see some improvement in the throughput.

And indeed we do: the throughput has doubled.

Yet both the before and after throughputs are order 1/N.

Is there any hope of a dramatically better throughput?

The necessary insight is that as long as an entire row is inside a single pipeline stage, the latency of the stage will be order N since we have to leave time for the N-bit ripple-carry add to complete.

There are several ways to tackle this problem.

The technique illustrated here will be useful in our next task.

In this schematic we've redrawn the carry chains.

Carry-outs are still connected to a module one column to the left, but, in this case, a module that's down a row.

So all the additions that need to happen in a specific column still happen in that column, we've just reorganized which row does the adding.

Let's pipeline this revised diagram, creating stages with approximately two module's worth of propagation delay.

The horizontal contours now break the long carry chains and the latency of each stage is now constant, independent of N. Note that we had to add order N extra rows to take of the propagating the carries all the way to the end - the extra circuitry is shown in the grey box.

To achieve a latency that's independent of N in each stage, we'll need order N contours.

This means the latency is constant, which in order-of notation we write as "order 1".

But this means the clock period is now independent of N, as is the throughput - they are both order 1.

With order N contours, there are order N pipeline stages, so the system latency is order N.

The hardware cost is still order N^2.

So the pipelined carry-save multiplier has dramatically better throughput than the original circuit, another design tradeoff we can remember for future use.

We'll use the carry-save technique in our next optimization, which is to implement the multiplier using only order N hardware.

This sequential multiplier design computes a single partial product in each step and adds it to the accumulating sum.

It will take order N steps to perform the complete multiplication.

In each step, the next bit of the multiplier, found in the low-order bit of the B register, is ANDed with the multiplicand to form the next partial product.

This is sent to the N-bit carry-save adder to be added to the accumulating sum in the P register.

The value in the P register and the output of the adder are in "carry-save format".

This means there are 32 data bits, but, in addition, 31 saved carries, to be added to the appropriate column in the next cycle.

The output of the carry-save adder is saved in the P register, then in preparation for the next step both P and B are shifted right by 1 bit.

So each cycle one bit of the accumulated sum is retired to the B register since it can no longer be affected by the remaining partial products.

Think of it this way: instead of shifting the partial products left to account for the weight of the current multiplier bit, we're shifting the accumulated sum right!

The clock period needed for the sequential logic is quite small, and, more importantly is independent of N. Since

there's no carry propagation, the latency of the carry-save adder is very small, i.e., only enough time for the operation of a single full adder module.

After order N steps, we've generated the necessary partial products, but will need to continue for another order N steps to finish propagating the carries through the carry-save adder.

But even at 2N steps, the overall latency of the multiplier is still order N.

And at the end of the 2N steps, we produce the answer in the P and B registers combined, so the throughput is order 1/N. The big change is in the hardware cost at order N, a dramatic improvement over the order N^2 hardware cost of the original combinational multiplier.

This completes our little foray into multiplier designs.

We've seen that with a little cleverness we can create designs with order 1 throughput, or designs with only order N hardware.

The technique of carry-save addition is useful in many situations and its use can improve throughput at constant hardware cost, or save hardware at a constant throughput.