Okay, now let's apply all this analysis to improving the performance of our circuits.

The latency of a combinational logic circuit is simply its propagation delay t_PD.

And the throughput is just 1/t_PD since we start processing the next input only after finishing the computation on the current input.

Consider a combinational system with three components: F, G, and H, where F and G work in parallel to produce the inputs to H. Using this timing diagram we can follow the processing of a particular input value X. Sometime after X is valid and stable, the F and G modules produce their outputs F(X) and G(X).

Now that the inputs to H are valid and stable, the H module will produce the system output P(X) after a delay set by the propagation delay of H.

The total elapsed time from valid input to valid output is determined by the propagation delays of the component modules.

Assuming we use those modules as-is, we can't make any improvements on this latency.

But what about the system's throughput?

Observe that after producing their outputs, the F and G modules are sitting sitting idle, just holding their outputs stable while H performs its computation.

Can we figure out a way for F and G to get started processing the next input while still letting H do its job on the first input?

In other words, can we divide the processing of the combinational circuit into two stages where the first stage computes F(X) and G(X), and the second stage computes H(X)?

If we can, then we can increase the throughput of the system.

Mr. Blue's inspiration is to use registers to hold the values F(X) and G(X) for use by H, while the F and G modules start working on the next input value.

To make our timing analysis a little easier, we'll assume that our pipelining registers have a zero propagation delay and setup time.

The appropriate clock period for this sequential circuit is determined by the propagation delay of the slowest processing stage.

In this example, the stage with F and G needs a clock period of at least 20 ns to work correctly.

And the stage with H needs a clock period of 25 ns to work correctly.

So the second stage is the slowest and sets the system clock period at 25 ns.

This will be our general plan for increasing the throughput of combinational logic: we'll use registers to divide the processing into a sequence of stages, where the registers capture the outputs from one processing stage and hold them as inputs for the next processing stage.

A particular input will progress through the system at the rate of one stage per clock cycle.

In this example, there are two stages in the processing pipeline and the clock period is 25 ns, so the latency of the pipelined system is 50 ns, i.e., the number of stages times the system's clock period.

The latency of the pipeline system is a little longer than the latency of the unpipelined system.

However, the pipeline system produces 1 output every clock period, or 25 ns.

The pipeline system has considerably better throughput at the cost of a small increase in latency.

Pipeline diagrams help us visualize the operation of a pipelined system.

The rows of the pipeline diagram represent the pipeline stages and the columns are successive clock cycles.

At the beginning of clock cycle i the input X_i becomes stable and valid.

Then during clock cycle i the F and G modules process that input and at the end of the cycle the results F(X_i) and G(X_i) are captured by the pipeline registers between the first and second stages.

Then in cycle i+1, H uses the captured values do its share of the processing of X_i.

And, meanwhile, the F and G modules are working on X_i+1.

You can see that the processing for a particular input value moves diagonally through the diagram, one pipeline stage per clock cycle.

At the end of cycle i+1, the output of H is captured by the final pipeline register and is available for use during cycle i+2.

The total time elapsed between the arrival of an input and the availability of the output is two cycles.

The processing continues cycle after cycle, producing a new output every clock cycle.

Using the pipeline diagram we can track how a particular input progresses through the system or see what all the stages are doing in any particular cycle.

We'll define a K-stage pipeline (or K-pipeline for short) as an acyclic circuit having exactly K registers on every path from input to output.

An unpipelined combinational circuit is thus a 0-stage pipeline.

To make it easy to build larger pipelined systems out of pipelined components, we'll adopt the convention that every pipeline stage, and hence every K-stage pipeline, has a register on its output.

We'll use the techniques we learned for analyzing the timing of sequential circuits to ensure the clock signal common to all the pipeline registers has a period sufficient to ensure correct operation of each stage.

So for every register-to-register and input-to-register path, we need to compute the sum of the propagation delay of the input register, plus the propagation delay of the combinational logic, plus the setup time of the output register.

Then we'll choose the system's clock period to be greater than or equal to the largest such sum.

With the correct clock period and exactly K-registers along each path from system input to system output, we are guaranteed that the K-pipeline will compute the same outputs as the original unpipelined combinational circuit.

The latency of a K-pipeline is K times the period of the system's clock.

And the throughput of a K-pipeline is the frequency of the system's clock, i.e., 1 over the clock period.