Okay, it's finally time to investigate issues caused by asynchronous inputs to a sequential logic circuit.

By "asynchronous" we mean that the timing of transitions on the input is completely independent of the timing of the sequential logic clock.

This situation arises when the inputs arrive from the outside world where the timing of events is not under our control.

As we saw at the end of Chapter 5, to ensure reliable operation of the state registers inputs to a sequential logic system have to obey setup and hold time constraints relative to the rising edge of the system clock.

Clearly if the input can change at anytime, it can change at time that would violate the setup and hold times.

Maybe we can come up with a synchronizer circuit that takes an unsynchronized input signal and produces a synchronized signal that only changes shortly after the rising edge of the clock.

We'd use a synchronizer on each asynchronous input and solve our timing problems that way.

Here's a detailed specification for our synchronizer.

The synchronizer has two inputs, IN and CLK, which have transitions at time t_IN and t_C respectively.

If IN's transition happens sufficiently before C's transition, we want the synchronizer to output a 1 within some bounded time t_D after CLK's transition.

And if CLK's transition happens sufficient before IN's transition, we want the synchronizer to output a 0 within time t_D after CLK's transition.

Finally, if the two transitions are closer together than some specified interval t_E, the synchronizer can output either a 0 or a 1 within time t_D of CLK's transition.

Either answer is fine so long as it's a stable digital 0 or digital 1 by the specified deadline.

This turns out to be an unsolvable problem!

For no finite values of t_E and t_D can we build a synchronizer that's guaranteed to meet this specification even when using components that are 100% reliable.

But can't we just use a D register to solve the problem?

We'll connect IN to the register's data input and connect CLK to the register's clock input.

We'll set the decision time t_D to the propagation delay of the register and the allowable error interval to the larger of the register's setup and hold times.

Our theory is that if the rising edge of IN occurs at least t_SETUP before the rising edge of CLK, the register is guaranteed to output a 1.

And if IN transitions more than t_HOLD after the rising edge of CLK, the register is guaranteed to output a 0.

So far, so good.

If IN transitions during the setup and hold times with respect to the rising edge on CLK, we know we've violated the dynamic discipline and we can't tell whether the register will store a 0 or a 1.

But in this case, our specification lets us produce either answer, so we're good to go, right?

Sadly, we're not good to go.

We're lured by the digital abstraction into assuming that even if we violate the dynamic discipline that Q must be either 1 or 0 after the propagation delay.

But that isn't a valid assumption as we'll see when we look more carefully at the operation of the register's master latch when B and C change at about the same time.

Recall that the master latch is really just a lenient MUX that can be configured as a bi-stable storage element using a positive feedback loop.

When the latch is in memory mode, it's essentially a two-gate cyclic circuit whose behavior has two constraints: the voltage transfer characteristic of the two-gate circuit, shown in green on the graph, and that V_IN = V_OUT, a constraint that's shown in red on the graph.

These two curves intersect at three points.

Our concern is the middle point of intersection.

If IN and CLK change at the same time, the voltage on Q may be in transition at the time the MUX closes and enables the positive feedback loop.

So the initial voltage in the feedback loop may happen to be at or very near the voltage of the middle intersection point.

When Q is at the metastable voltage, the storage loop is in an unstable equilibrium called the metastable state.

In theory the system could balance at this point forever, but a small change in the voltages in the loop will move the system away from the metastable equilibrium point and set it irrevocably in motion towards the stable equilibrium points.

Here's the issue we face: we can't bound the amount of time the system will spend in the metastable state.

Here's what we know about the metastable state.

It's in the forbidden zone of the digital signaling specifications and so corresponds to an invalid logic level.

Violating the dynamic discipline means that our register is no longer guaranteed to produce a digital output in bounded time.

A persistent invalid logic level can wreak both logical and electrical havoc in our sequential logic circuit.

Since combinational logic gates with invalid inputs have unpredictable outputs, an invalid signal may corrupt the state and output values in our sequential system.

At the electrical level, if an input to a CMOS gate is at the metastable voltage, both PFET and NFET switches controlled by that input would be conducting, so we'd have a path between V_DD and GROUND, causing a spike in the system's power dissipation.

It's an unstable equilibrium and will eventually be resolved by a transition to one of the two stable equilibrium points.

You can see from the graph that the metastable voltage is in the high-gain region of the VTC, so a small change in V_IN results in a large change in V_OUT, and once away from the metastable point the loop voltage will move towards 0 or V_DD.

The time it takes for the system to evolve to a stable equilibrium is related to how close Q's voltage was to the metastable point when the positive feedback loop was enabled.

The closer Q's initial voltage is to the metastable voltage, the longer it will take for the system to resolve the metastability.

But since there's no lower bound on how close Q is to the metastable voltage, there's no upper bound on the time it will take for resolution.

In other words, if you specify a bound, e.g., t_D, on the time available for resolution, there's a range of initial Q voltages that won't be resolved within that time.

If the system goes metastable at some point in time, then there's a non-zero probability that the system will still be metastable after some interval T, for any finite choice of T. The good news is that the probability of being metastable at the end of the interval decreases exponentially with increasing T.

Note that every bistable system has at least one metastable state.

So metastability is the price we pay for building storage elements from positive feedback loops.

If you'd like to read a more thorough discussion of synchronizers and related problems and learn about the mathematics behind the exponential probabilities, please see Chapter 10 of the Course Notes.

Our approach to dealing with asynchronous inputs is to put the potentially metastable value coming out of our D-register synchronizer into "quarantine" by adding a second register hooked to the output of the first register.

If a transition on the input violates the dynamic discipline and causes the first register to go metastable, it's not immediately an issue since the metastable value is stopped from entering the system by the second register.

In fact, during the first half of the clock cycle, the master latch in the second register is closed, so the metastable value is being completely ignored.

It's only at the next clock edge, an entire clock period later, that the second D register will need a valid and stable input.

There's still some probability that the first register will be metastable after an entire clock period, but we can make that probability as low as we wish by choosing a sufficiently long clock period.

In other words, the output of the second register, which provides the signal used by the internal combinational logic, will be stable and valid with a probability of our choosing.

Validity is not 100% guaranteed, but the failure times are measured in years or decades, so it's not an issue in practice.

Without the second register, the system might see a metastability failure every handful of hours - the exact failure rate depends on the transition frequencies and gains in the circuit.

What happens if our clock period is short but we want a long quarantine time?

We can use multiple quarantine registers in series.

It's the total delay between when the first register goes metastable and when the synchronized input is used by the internal logic that determines the failure probability.

The bottom line?

We can use synchronizing registers to quarantine potentially metastable signals for some period of time.

Since the probability of still being metastable decreases exponentially with the quarantine time, we can reduce the failure probability to any desired level.

Not a 100% guaranteed, but close enough that metastability is not a practical issue if we use our quarantine strategy.