Synchronization and Metastability

Steve Golson

Trilobyte Systems
Carlisle, Massachusetts
USA

web: www.trilobyte.com

email: sgolson@trilobyte.com

ABSTRACT

The phenomenon of metastability is inherent in clocked digital logic. Many techniques have been presented for minimizing metastability, both for crossing clock domains, and for handling asynchronous inputs. Some of these "best practices" have unexpected weaknesses and must be used carefully, particularly at smaller process nodes. This paper will explore these shortcomings and suggest alternative schemes that are more robust. A PrimeTime methodology for verifying multi-clock designs will be presented.

Table of Contents

1. Preface

What do the following products have in common?

- LINC
- DEC PDP-11/45
- DEC PDP-15
- Space Shuttle PASS
- AMD Am29000
- AMD 9513
- AMD 9519
- Zilog Z-80 SIO
- Intel 8048
- Intel 8202
- Intel 8085
- Honeywell 516 / ARPANET IMP
- Synopsys DW04_sync

They all suffered from *synchronizer failures* due to *metastability* [9][10][28][43][49][51][54][57][59][61].

2. Introduction and review

The phenomenon of metastability is inherent in clocked digital logic. Every bistable device will have two *stable states* (hence the name *bistable*) and also a third *metastable state*. If the device enters its metastable state, it will stay there for an indeterminate and unbounded length of time before eventually transitioning into one of the stable states.

Digital latches and flip-flops are bistable devices that can store a one or zero (the two stable states) but may also enter their metastable state under marginal triggering conditions if the specifications of the cell are violated (e.g., input setup time, input hold time, clock slew rate, power supply voltage, clock pulse width)[1]. Since the latch (or flip-flop) will stay metastable for an indeterminate time, your system has by definition failed its timing requirements.

Physics dictates that you cannot eliminate metastability. However you can minimize its effects and reduce the probability of failure by isolating the asynchronous inputs using a *synchronizer circuit*. The synchronizer conditions the input into a known relationship with the system clock.

Synchronizer circuits were in use by 1946 [26] and the first discussion of metastability in digital logic was by Lubkin in 1952 [46]. Synchronization and metastability has remained a fertile and sometimes controversial topic for research ever since [11][14–18][25].

[1] When this happens we say that the latch has “gone metastable.”

2.1. The basics

In universities today, electrical engineering and computer science undergrads are usually taught about metastability and synchronizers in one lecture of one class. The lecture typically goes something like this [30][44][59–62].

Consider the simple D latch in Figure 1.

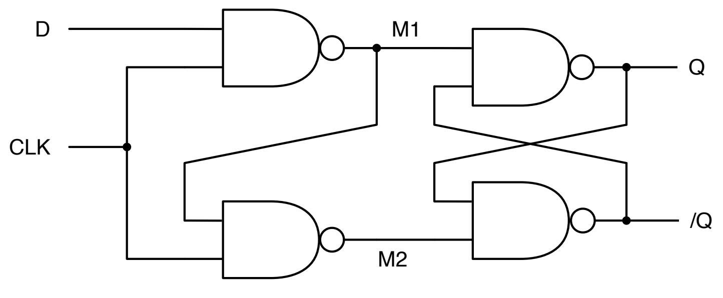

Figure 1: D latch

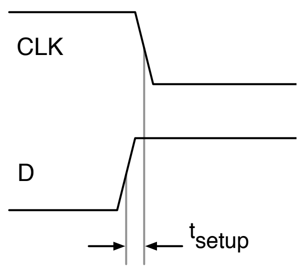

Figure 2: D latch waveform

Initially the data D is low and clock CLK is high. The latch is transparent and thus Q is low and /Q is high. Now assume D goes high just before CLK goes low. What happens? At first M1 will transition low causing output Q to head towards high, but then M1 will transition back to high because CLK has just gone low. If the overlap between D and CLK is sufficiently small, M1 and M2 can both be high and output Q may not yet have reached a high state.

At this point because M1 and M2 are high they cannot have any further effect on the outputs. The final outcome for Q and /Q is determined solely by the cross-coupled gates.

This situation is now similar to the cross-coupled inverters in Figure 3. The steady-state DC transfer curves for each inverter are shown in Figure 4.

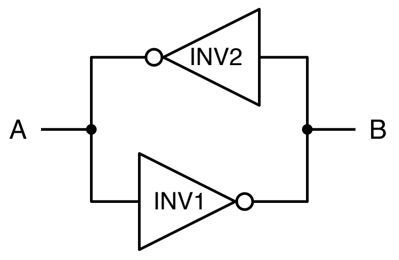

Figure 3: Cross-coupled inverters

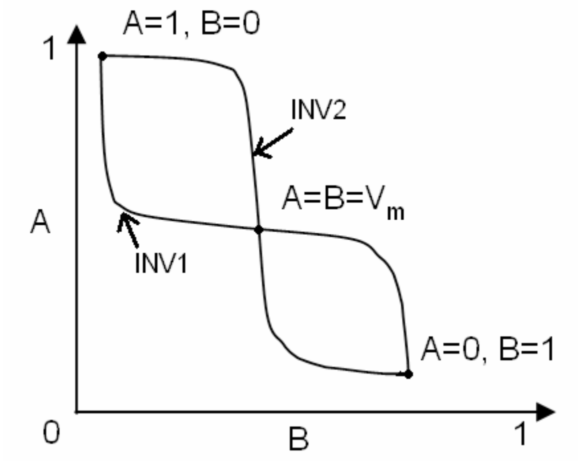

Figure 4: Transfer curves for cross-coupled inverters (from Zhou [62] figure 2.6)

What are the possible steady-state behaviors for these cross-coupled inverters? We can determine this graphically from Figure 4; it is simply the points at which the two transfer curves intersect. Surprisingly there are not two but *three* such points. There are two *stable states* at A=0, B=1 and A=1, B=0. The third point is A=B=V_m where V_m is not a legal logic level. This third intersection represents a *metastable state.* This is a valid solution to the transfer equations; the voltages are self-consistent and the latch can theoretically remain in this state indefinitely. However any noise or other disruption will tend to drive the latch toward one of the stable states.

The behavior of the cross-coupled inverters can be likened to a perfectly smooth ball and hill, as shown in Figure 5 [34][59].

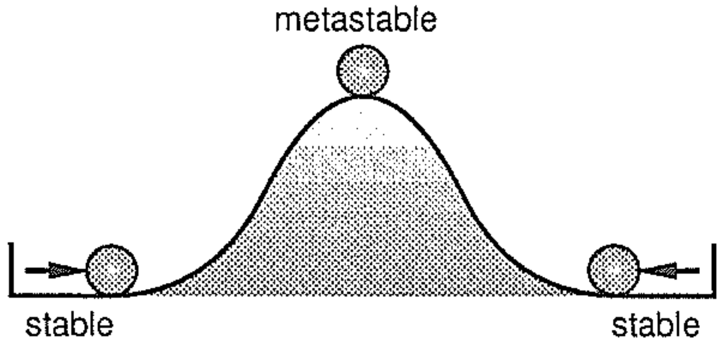

Figure 5: Mechanical analogy of bistable circuit
(from Wakerly [59] figure 5)

If we drop a ball from overhead, it will *probably* immediately roll down to one side of the hill or the other. However what if it lands *exactly* on the top? The ball may precariously balance there for a while before some random force perturbs it and starts it rolling down one side or the other. If you forcefully kick the ball toward the right, it will go over the hill and down to the opposite state on the right side. If instead you kick it weakly it will roll up the hill a bit, and then back down to the left where it started. If you kick the ball *just right* then it may reach the top and balance there for a while, before falling back to one side or the other.

Like the ball at the top of the hill, the cross-coupled inverters may stay in the metastable state for an unpredictable length of time before nondeterministically transitioning into one or the other of the two stable states.

At this point the lecturer may show some pictures of flops caught in the act of metastability.

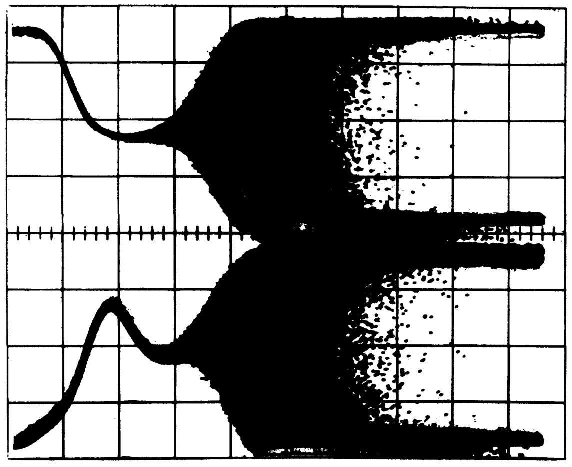

Figure 6: Sampling oscilloscope trace showing Q and /Q of flip-flop with clock and data inputs changing simultanteously (from Chaney and Molnar [15] figure 1)

2.2. The theory

Now the lecture continues with a mathematical analysis of metastable behavior. How often does metastability occur? And once it occurs, how long does it last? We will skip over the differential equations and get right to the results [19][30][38][42][53][58].

Consider our D latch some time t_R after CLK has gone low. What is the probability that the latch is still in the metastable state? It is the product of two probabilities: first, the probability of entering the metastable state in the first place; and second, the probability that the metastable state will persist for at least time t_R [30].

$$\begin{aligned} P(\text{metastable}) &= P(\text{enter metastability}) \times P(\text{still in state after } t_R) \\ &= P_E \times P_S \\ P_F &= P_E \times P_S \end{aligned} \tag{1}$$

P_F is the *probability of failure*, in other words, the latch went metastable and is still metastable even though we have waited for time t_R.

To calculate the probability of entering metastability, we define a short window T_0 around the clock's sampling edge, such that if data changes during that window, then the latch will become metastable. Think of it as "setup-and-hold time" for metastability [30].

Assuming the data can arrive at any time during the clock period t_c, what then is the probability that the data happens to arrive during T_0? It's simply the fraction of the clock period that is occupied by T_0, which is

$$P_E = \frac{T_0}{t_c} \tag{2}$$

or if f_c is the clock frequency

$$P_E = f_c T_0 \tag{3}$$

To calculate the probability that the latch will remain metastable for a certain period of time, first we notice in Figure 4 that at metastability the gates are operating in their linear region, thus any initial small voltage difference in the cross-coupled gates will be exponentially amplified by the gain of the gates. If the latch is metastable at time 0, then the probability that the latch is still metastable after some time t_R is

$$P_S = e^{-t_R/\tau} \tag{4}$$

where time constant τ is roughly the inverse of the gain of the gates and t_R is the *resolution time*.

Then we have

$$P_F = P_E P_S = f_c T_0 e^{-t_R/\tau} \tag{5}$$

Assume our input D is changing at an event rate of f_d then the *failure rate* λ is this event rate multiplied by the failure probability P_F for an individual event

$$\lambda = f_d P_F = f_d f_c T_0 e^{-t_R/\tau} \tag{6}$$

and the *mean time between failures* is defined as the inverse of the failure rate

$$MTBF = \frac{1}{\lambda} = \frac{e^{t_R/\tau}}{f_d f_c T_0} \tag{7}$$

2.3. The synchronizer

The lecture continues with some basic recommendations.

We can minimize the impact of metastability by using a *synchronizer* to isolate the offending signal. The simplest synchronizer is a single flip-flop:

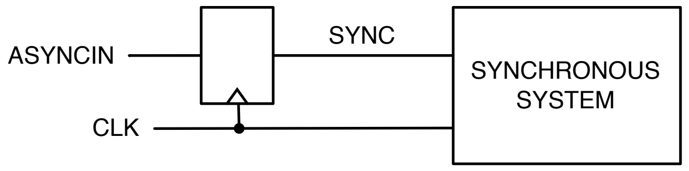

Figure 7: Simple synchronizer

This can work, however it's difficult to guarantee that the flop will have sufficient time to resolve the metastability. The recommended solution is to give the flop an entire clock period to settle:

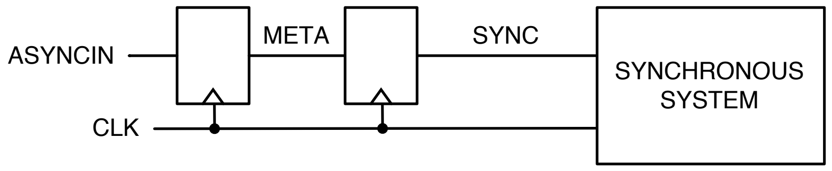

Figure 8: Basic two-flop synchronizer

Net META drives nothing but the second flop. Be careful with loading on these signals. If net META is heavily loaded, the resolution time will be reduced, and MTBF will get worse.

Do not synchronize the same signal in more than one place. In Figure 9 it is possible for one flip-flop to resolve the metastability to 0 while the other resolves to 1.

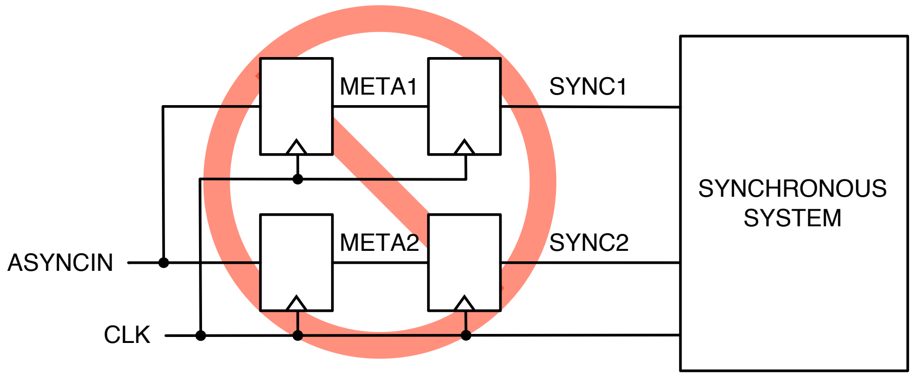

Figure 9: Broken synchronizer—DO NOT BUILD THIS

META1-SYNC1 and META2-SYNC2 can have different ideas about the value of ASYNCIN. This is very bad! Instead, use a synchronizer like Figure 8 and run signal SYNC to multiple destinations.

Do not attempt to synchronize multi-bit buses. Instead synchronize a single-bit control signal that indicates when the bus is stable and data is available:

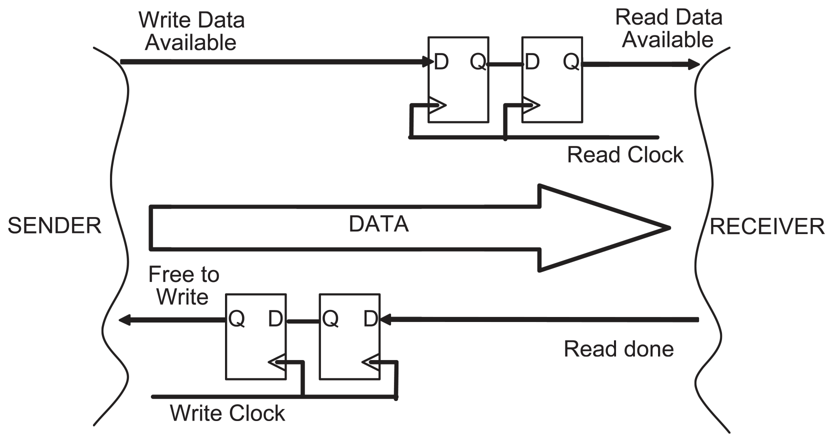

Figure 10: Synchronizing a multi-bit bus
(from Kinniment [42] figure 7.1)

Be careful with pulses crossing clock domains. A single-clock pulse in a fast clock domain may not be sampled at all when synchronized by a slower clock domain. Rather, convert the pulse to a REQ transition in the transmit domain, and when the transition is received, convert it back to a pulse in the receiving domain [29].

2.4. The ubiquity of metastability

Finally the lecture ends with a reminder that metastability is unavoidable. It is an inherent part of clocked digital logic. As Wakerly [59] puts it, referring to the cross-coupled inverters:

> If even the *simplest* sequential circuit is susceptible to metastable behavior, you can be sure that *all* sequential circuits are susceptible.

Thank you class!

3. The rest of the story

Gosh, that covers it all, right? I've got my MTBF equation and my two-flop synchronizer. Nothing else to learn, is there? Where's my SystemVerilog manual? Let's code!

Not so fast. Let's consider what our example lecture left out, shall we?

3.1. Flop parameters

Where do you get τ and T_0 for your flip-flop? It almost certainly will not be in your ASIC cell library databook.[2] What's a designer to do?

[2] If you are using discrete flip-flops on a PCB then you might get this information from your vendor [12][35].

Do you own your own fab? If so then run some test chips and probe your flops. You can build up rather simple test fixtures that will allow you to determine τ and T_0. Make sure you characterize the cell over your full operating range [27][39].

Do you have netlists for your flops? Do you have transistor models for your technology? Do you have SPICE? There are techniques for running SPICE analysis that will allow you to extract the parameters you need [36][42][53][55].

Consider a simple D flip-flop:

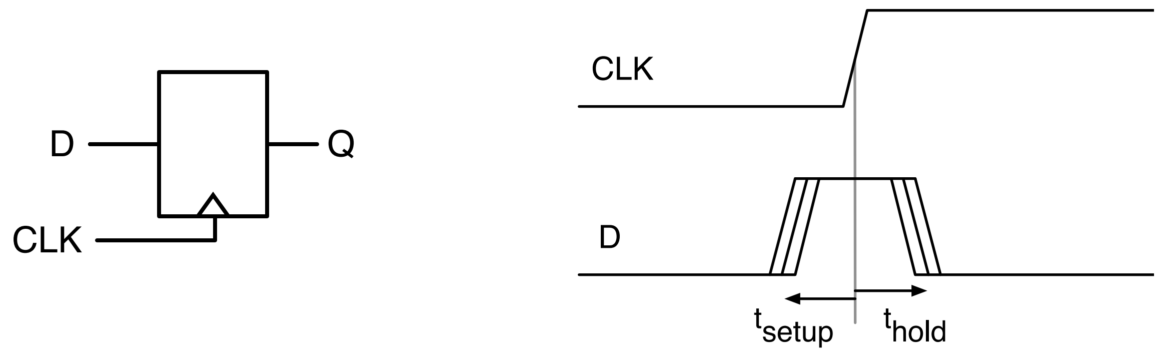

Figure 11: D flip-flop and test waveform

By forcing the setup and hold times of the data input pulse to be arbitrarily small we should be able to force the flop into metastability.

Figure 12 shows SPICE results for a simple D flip-flop in 65nm at typical conditions. The internal nets of the master latch are labeled A and B like the cross-coupled inverters in Figure 3.

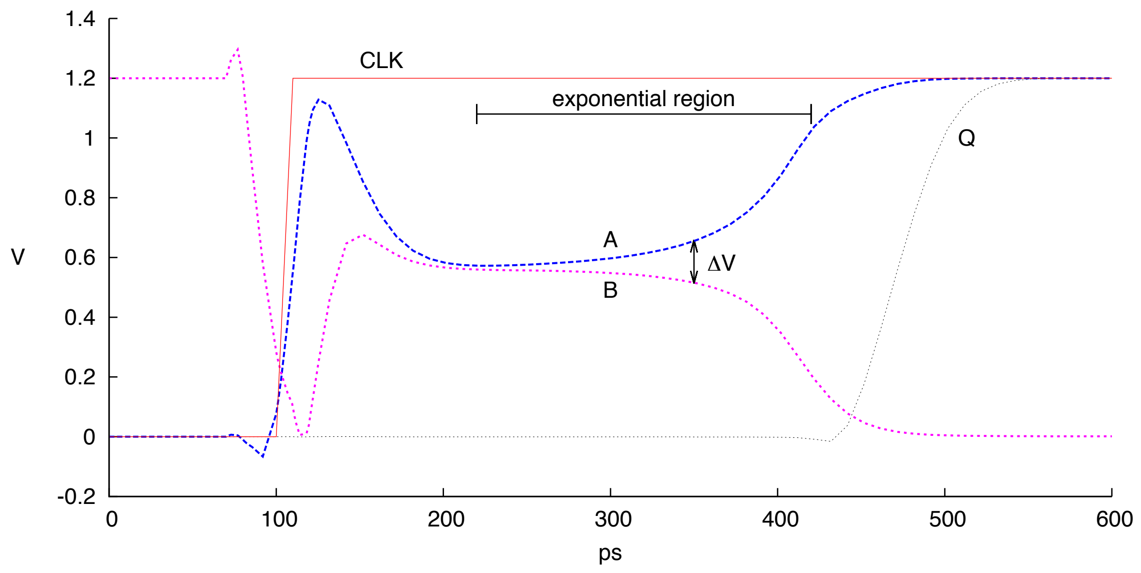

Figure 12: SPICE results from D flip-flop master latch

A very short hold time on the data input causes the master latch to enter its metastable state. Immediately after CLK rises the internal nodes A and B are fluctuating and have not yet reached a stable level. Eventually they have almost the same voltage as the latch enters the metastable state. The small ΔV between A and B is then exponentially amplified by the gain of the cross-coupled gates, and finally the latch enters a stable state with Q driven high.

Time constant τ is defined as the inverse of the gain in the exponential region. By making a log-normal plot of ΔV in this region we can directly measure τ as the slope of the line.

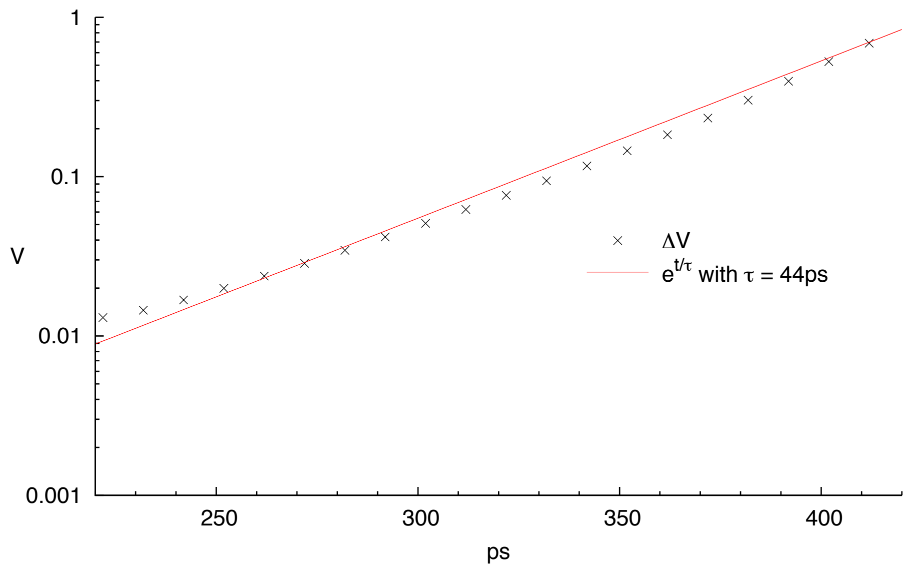

Figure 13: Log-normal plot of ΔV in exponential region

The voltage grows at an exponential rate given by

$$\Delta V_2 = \Delta V_1 \cdot e^{\frac{(t_2 - t_1)}{\tau}} \tag{8}$$

so then

$$\tau = \frac{t_2 - t_1}{\ln\left(\frac{\Delta V_2}{\Delta V_1}\right)} \tag{9}$$

and in this example

$$\tau = \frac{420\,\text{ps} - 220\,\text{ps}}{\ln\left(\frac{0.8\text{V}}{0.009\text{V}}\right)} = 44\,\text{ps} \tag{10}$$

Here's another method to derive τ. With the same D flop as before, use a large setup time, then vary the data hold time relative to CLK, and extract the resolution time from each SPICE run.[3] Here, resolution time is defined as the time from CLK rising to the master latch internal feedback node reaching a stable value (high or low). Figure 14 shows the expected results [53].

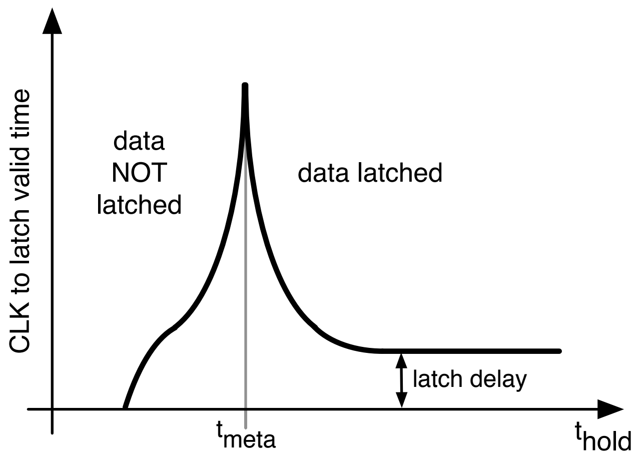

Figure 14: Time to latch valid versus data hold time

At the left side, the input data pulse has gone away well before the clock edge, and thus the latch holds its value. At the far right side, there is adequate hold time and the latch loads its new value quickly (i.e., with very short resolution time). Define t_{meta} as the hold time that causes the latch to go metastable. Around t_{meta} we see exponential behavior with greatly increasing resolution time. Figure 15 shows the results from SPICE.

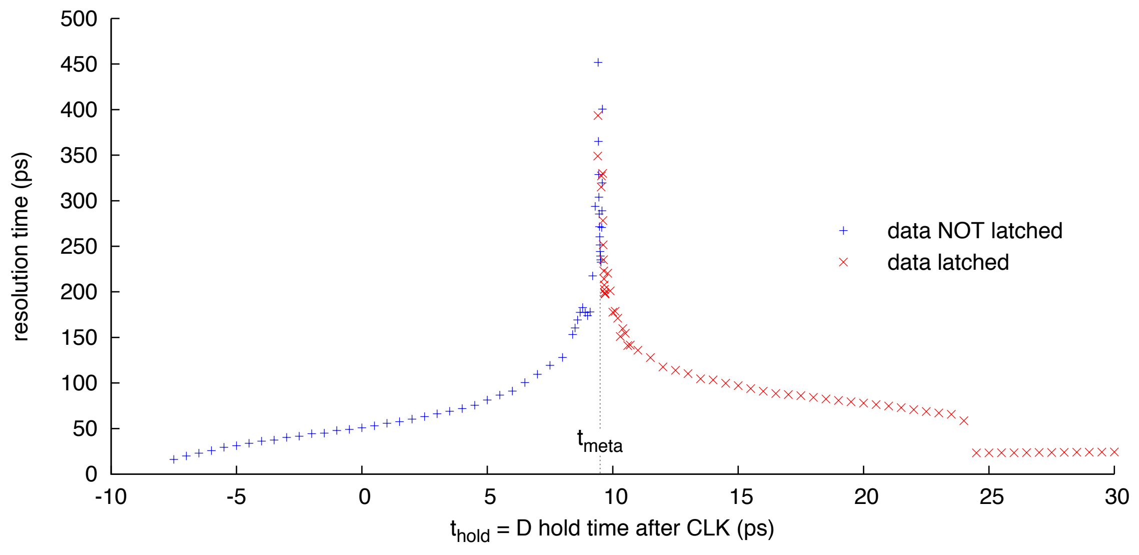

Figure 15: SPICE results with varying hold time

[3] Similar analysis could be done by keeping hold time constant and varying the data setup time.

Stucki and Cox [56] and Stoll [55] were among the first to describe this type of plot and Portmann [53] gives a very elegant graphical explanation and equations to define the behavior. See Figure 16.

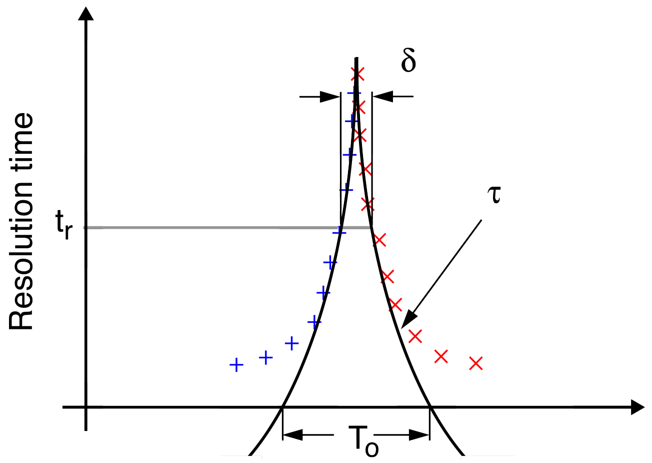

Figure 16: Fitted exponential showing metastability parameters (from Portmann [53] figure 2.9b)

Define δ as the *metastability window* such that if data transitions within δ then it will not be resolved within a given resolution time t_r. Then the width of this window can be calculated for a given resolution time:

$$\delta = T_0 e^{-t_R/\tau} \tag{11}$$

where τ is the time constant and T_0 is the asymptotic width of the window with zero resolution time.[4]

[4] That's why the parameter is named T_0, because it's the width (T) measured when there is 0 (zero) resolution time.

From our simulations we can calculate δ as the difference between the hold time and t_{meta}. Taking the absolute value of this difference allows both left and right sides of the data to be used. Now show our measured resolution times on a log-normal plot:

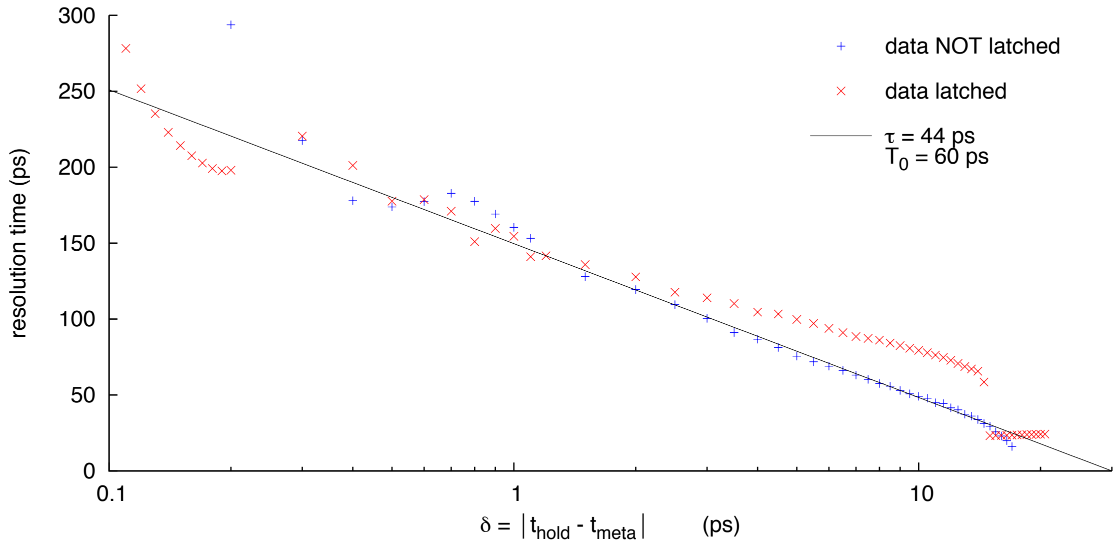

Figure 17: Log-normal plot of the same data

Again we can derive τ from the slope

$$\tau = -\frac{t_2 - t_1}{\ln\left(\frac{\delta_2}{\delta_1}\right)} = -\frac{250\,\text{ps} - 0\,\text{ps}}{\ln\left(\frac{0.1\,\text{ps}}{30\,\text{ps}}\right)} = 44\,\text{ps} \tag{12}$$

and T_0 is twice the x intercept

$$T_0 = 30\,\text{ps} \times 2 = 60\,\text{ps} \tag{13}$$

There is very good agreement with the previous derivation of τ. Figure 18 shows the same data again on a normal plot but now with the exponential curves added.

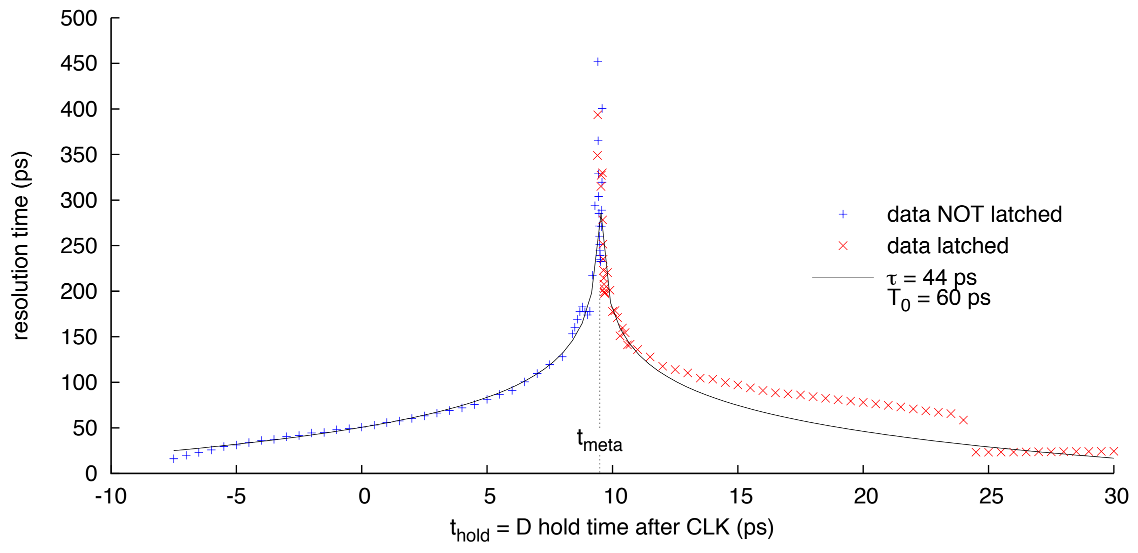

Figure 18: SPICE results with varying hold time, and exponential model for t_R

The preceding required access to the internal nets of the latch. However the same analysis can be done by measuring just the Q output pin of the flop. In this case we only get the right side data; the left side is hidden as there is no visible change to Q. The curve has a similar shape:

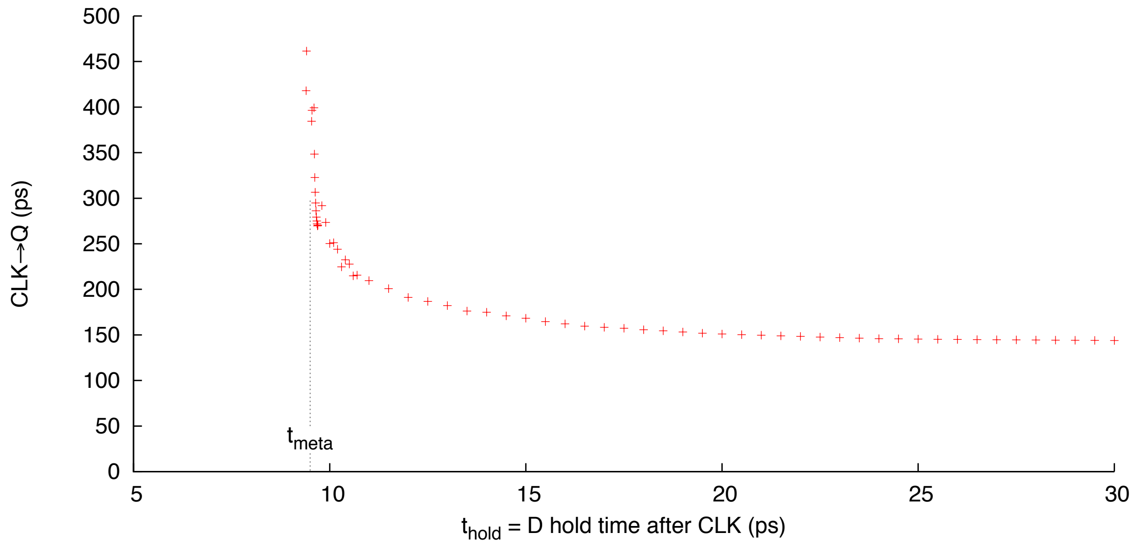

Figure 19: SPICE results with varying hold time

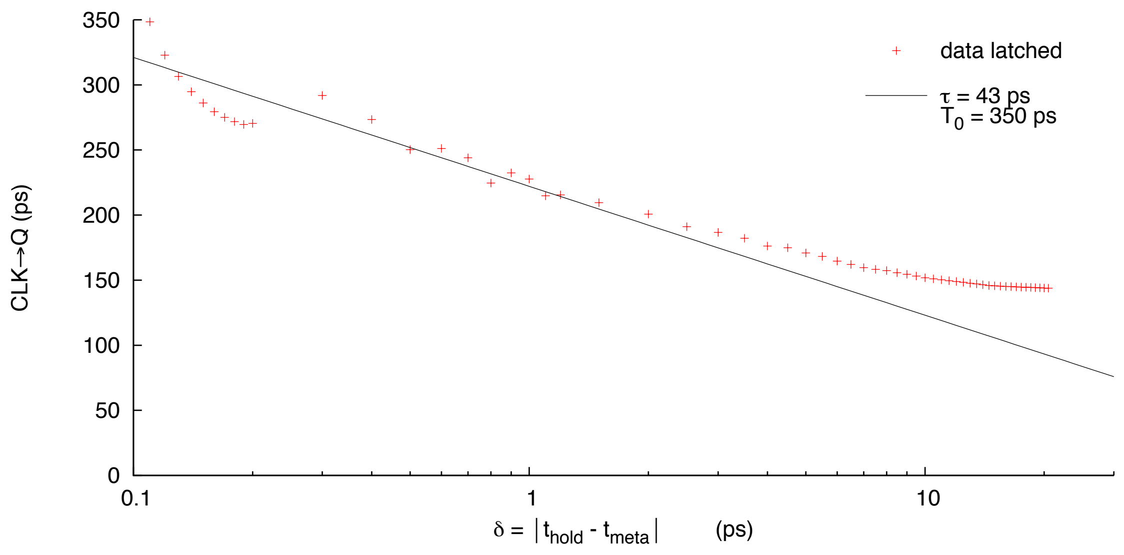

Figure 20: Log-normal plot of the same data

The log-normal plot allows us to extracting τ and *T₀* as before

$$\tau = -\frac{t_2 - t_1}{\ln\left(\frac{\delta_2}{\delta_1}\right)} = -\frac{300\,\text{ps} - 200\,\text{ps}}{\ln\left(\frac{0.2\,\text{ps}}{2\,\text{ps}}\right)} = 43\,\text{ps} \tag{14}$$

$$T_0 = 175\,\text{ps} \times 2 = 350\,\text{ps} \tag{15}$$

Very close agreement with τ from the previous calculations. *T₀* is different because of the extra delay through the slave latch and output buffers.

This measured τ is from the master latch only. CLK was held high so the slave latch was transparent. By taking CLK low at just the right time we should be able to induce metastability in the slave and measure its τ also. Also note that τ for the overall flop is sensitive to the duty cycle of the clock [20].

This analysis only measured the CLK and Q pins of the flop; it did not require access to the flop internals. It may be possible to use this technique to measure physical devices in your lab.

Note that simulation techniques like those outlined here may not be sufficiently accurate to model long duration metastability, due to the limited resolution of the simulator and potential variations in τ [62]. It is a complex and tricky procedure to generate an accurate characterization [6][21][22]. Process variation has a significant effect on τ [63].

However you determine the metastability parameters *τ* and *T₀*, make sure you analyze all your flip-flop types. Make sure you run the analysis for all possible operating conditions. Read carefully the available literature and research. Implore your library vendor to provide these parameters to you.

3.2. Multiple synchronizers

How many bistable devices are there in Figure 21?

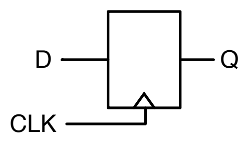

Figure 21: D flip-flop

Figure 21 has *two* bistable devices.

Recall that an edge-triggered flip-flop is implemented internally with *two* latches:

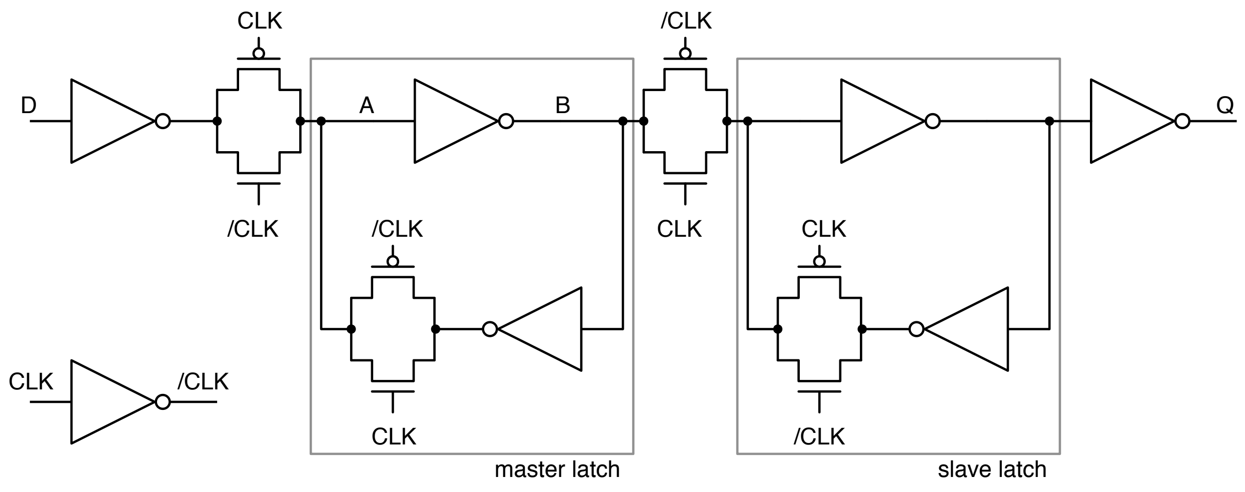

Figure 22: D flip-flop internal structure

The master latch samples D on rising CLK. The slave latch samples the output of the master latch (net B) on falling CLK. Each of these bistable latches can go metastable. The resolution time of each latch is *half* the clock period. There are *two* time constants τ_{master} and τ_{slave}. There are *two* metastability windows $T0_{master}$ and $T0_{slave}$. There is propagation delay between the latches.

How do we analyze this structure? What does our MTBF equation look like now?

What matters to the system is not whether the *master* latch is metastable, but whether the *slave* latch is metastable, because that is what drives the output Q. Metastability in the master can propagate to the slave during the first half of the clock period, and additionally if the master resolves its value close to the clock falling edge then the transition of B can induce metastability in the slave.

The overall MTBF for an edge-triggered flip-flop typically will have the same form as Equation (7) however the parameters τ and T_0 will be slightly different from the values for its component latches. Kinniment [42] gives a detailed analysis. Clock duty cycle also has an effect [20].

Kinniment [42] discusses the *clock back edge effect* which can cause a significant increase in failure rates during the second half of the clock period. This is due in part to the propagation delay between the two latches.

3.3. Multiple synchronizers

How many synchronizers are there in Figure 23?

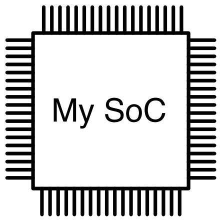

Figure 23: Your state-of-the-art bleeding-edge chip

There might be thousands of synchronizers in a modern SoC. What does our MTBF equation look like now?

For an ensemble of *N* synchronizers we can sum the failure rates of the individual synchronizers to get

$$MTBF(N) = \frac{1}{\sum_{n=1}^{N} \frac{1}{MTBF_n}} \tag{16}$$

Note that the worst-behaved synchronizer will dominate the calculation. Another way of putting this: adding many really good synchronizers to your design will not affect the overall MTBF much. However adding *one really bad* synchronizer can dramatically worsen your system MTBF.

If all the synchronizers are identical[5] then Equation (16) simplifies [31] to

$$MTBF(N) = \frac{e^{t_R/\tau}}{N\, f_d\, f_c\, T_0} \tag{17}$$

This analysis assumes all the synchronizers have independent uncorrelated failure rates. Is this correct? If not, how should the equations be modified?

This analysis further assumes that a failure (i.e., unresolved metastability) in a given synchronizer is just as bad as a failure in any other synchronizer. Is this correct? Are some synchronizers in your system more important than others? If so, how should the equations be modified?

[5] Are they really? Are you sure? They all have the same parasitics? I didn't think so.

3.4. Multiple synchronizers

How many synchronizers are there in Figure 24?

Figure 24: LAN party at DreamHack Winter 2011
(from ExtremeTech.com [2])

How many of your product will be sold? You don't want *any* of them to fail, do you?

What does our MTBF equation look like *now*?

3.5. Reliability and your definition of failure

Let's leave MTBF aside for the moment, and consider *probability* and *reliability* and *failure*.

There are three problems you must solve [41]:

1. Specify what your reliability goals are for your product. Rather than MTBF of a single synchronizer, this will involve specifying the aggregate probability of failure of a large number of flip-flops. Consider these assumptions:
   - We plan to sell K=100,000 of this chip
   - Each chip contains N=1,000 synchronizers
   - Each chip is expected to operate for five years

   Now we can specify our reliability goals, for example:
   - With probability 85%, not a single synchronizer out of the entire population of 10^8 will have a metastability failure during its operating lifetime

   or perhaps
   - Only 3% of our entire population of synchronizers will fail over the lifetime of the product

   or perhaps
   - During its operating lifetime, at most 25 of the synchronizers on any given chip will fail
2. Find the metastability parameters τ and T_0 for each of your synchronizers, including layout parasitics
3. Write an equation to calculate the aggregate probability of failure in terms of τ, T_0, clock frequency, data rate, lifetime of the system, etc.

Item 2. looks easiest but has some surprising subtleties (refer to previous Section 3.1).

Item 1. is the hardest because it requires your management to make a decision. Requiring 100% perfection is not an acceptable answer. Mother Nature won't allow it. Failure is inevitable – we can only try to minimize its occurrence [37].

Item 3. goes something like this [40]:

Assuming our chip has a linear failure rate λ given by

$$\lambda = \frac{1}{MTBF(N)} \tag{18}$$

then the *reliability function* R is defined as

$$R(T) = e^{-\lambda T} \tag{19}$$

This is the probability of no failure between time 0 and T. *R(T)* is also called the *survival function*.

Thus the probability of failure within time 0→T is

$$P(fail) = 1 - R(T) = 1 - e^{-\lambda T} \tag{20}$$

To calculate the probability that all K chips we sold will survive (not fail) for time 0→*T*, just multiply the individual probabilities that each one will survive:

$$\begin{aligned} P(\text{no fail in time } 0 \rightarrow T) &= \prod_{i=1}^{K} R_i(T) \\ &= \prod_{i=1}^{K} e^{-\lambda T} \\ &= \left(e^{-\lambda T}\right)^{K} \\ &= e^{-\lambda K T} \end{aligned} \tag{21}$$

To calculate the probability of no more than M failures from our population of K chips:

$$P(\leq M \text{ failures in time } 0 \rightarrow T) = \sum_{i=0}^{M-1} \left\{ {}_{K}C_{i} \cdot \left(1 - e^{-\lambda T}\right)^{i} \cdot \left(e^{-\lambda T}\right)^{K-i} \right\} \tag{22}$$

where the binomial coefficients are given by

$${}_{K}C_{i} = \frac{K!}{i!\,(K-i)!} \tag{23}$$

But don’t take my word for it. Do some research and figure out the equations for yourself.

4. Recommendations

Recall that our equation for MTBF of one synchronizer is

$$MTBF = \frac{e^{t_R/\tau}}{f_d f_c T_0} \tag{24}$$

and for our entire chip is

$$MTBF(N) = \frac{1}{\sum_{n=1}^{N} \frac{1}{MTBF_n}} \tag{25}$$

where

$MTBF$ =	mean time between failures (sec), this is the inverse of the failure rate λ
t_R =	resolution time, the time after the clock edge that output data is needed (sec)
τ =	time constant of the latch (sec)
f_d =	data arrival rate (Hz)
f_c =	clock sampling rate (Hz)
T_0 =	metastability window of the latch (sec)
N =	number of synchronizers in the chip

Let's consider each parameter and how it can be used to improve MTBF.

4.1. t_R

Bigger is better.

This is the synchronizer resolution time. This is how long you are willing to wait for any metastability to resolve itself.

Try and wait longer. Use a slower clock. Use a divided-down clock. Minimize the required setup of the following flop. Minimize the load on the output of the synchronizer flop, including the parasitic capacitance.

Improving this parameter is very beneficial because it affects the exponent.

4.2. τ

Smaller is better.

This is the synchronizer time constant, which is approximately the inverse of the small-signal amplifier gain around the cross-coupled gates of the latch.

Compare τ for all your flops and pick a better flop. In general this will be flops with higher power, faster CLK→Q delay, and the simplest logic (e.g., no reset, no scan, no input mux).

Note these characteristics (high power, faster, simple logic) are not requirements; rather they are just indications that the gain will most likely be higher. For example if the flop has all sorts of extra transistors implementing reset and scan, then it will probably not have as much gain as the equivalent simpler flop. In the absence of measured τ you can use these heuristics to hopefully pick a better flop.

Another way to improve τ is to perform a more accurate characterization of the flop you are using, including all parasitics. This hopefully will improve τ, or might not…

Improving this parameter is very beneficial because it affects the exponent.

Be aware that many factors can affect τ. Ginosar [30] reports that process variations, extreme temperatures, low supply voltages can increase τ by several orders of magnitude. This makes measurement and simulation of τ a challenging task, particularly at smaller geometries [6][21][22][62][63].

4.3. f_d

Smaller is better.

This is the input data rate. It can be difficult to estimate the actual data rate for a particular synchronizer. Slowing the transmit clock will always help.

Sometimes you will see MTBF equations with a factor of 2 in the denominator:

$$MTBF = \frac{e^{t_R/\tau}}{2 f_d f_c T_0} \tag{26}$$

This is because they are interpreting the data input waveform as a “clock”. A clock with frequency *f* has a signal transition rate of 2*f*. If you put a frequency counter on your data signal then you should multiply the measured “clock frequency” by 2 to get the data rate. Always think of this as *data rate* and you won’t be confused, and you won’t need the 2.

This has a linear effect on MTBF.

4.4. f_c

Smaller is better.

This is the clock sampling rate. Slower is better. In other words, a longer clock period is better. So use a slower clock, or use a divided-down clock.

Generally any change to f_c will affect t_R also.

This has a linear effect on MTBF.

4.5. T_0

Smaller is better.

This is the synchronizer's metastability window. This is *much* smaller than the minimum $t_{setup} + t_{hold}$ for the flop [55]. T_0 is related to the setup time and minimum voltage required for a valid output [53].

Compare T_0 for all your flops and pick a better flop. In general, but not necessarily, this will be flops with smallest minimum specification for $t_{setup} + t_{hold}$. In the absence of measured T_0 you can use this heuristic to hopefully pick a better flop. T_0 is affected by the latch time constant, so a flop with better τ will most likely have better T_0 also.

Another way to improve T_0 is to perform a more accurate characterization of the flop you are using, including all parasitics. This hopefully will improve T_0, or might not… And be aware of subtle interactions when trying to simultaneously derive T_0 and τ [21].

This has a linear effect on MTBF.

4.6. N

Smaller is better.

This is the number of synchronizers on your chip.

Change your design to avoid synchronization. Check your system specification—perhaps some of your "asynchronous" clocks are actually coherent. Can you run some of your subsystems on the same clock?

Recall that synchronizers with very high MTBF will have little effect on the system MTBF. So improving (or removing) the worst ones will have the most benefit on your system.

This generally has a linear effect on overall MTBF.

5. Common Fallacies

Let's discuss some common fallacies and misconceptions about metastability and synchronizers.

5.1. MTBF

Our required MTBF is 10 years, because nobody owns a computer for longer than that.

MTBF is *not* the lifetime of your part!

MTBF is *not* service time!

MTBF is *not* operating time!

MTBF is *not* time to first failure!

MTBF is simply the inverse of failure rate λ. An MTBF of 10 years is the same as saying

In the first year we expect 10% of our chips to fail.

Using Equation (20) we calculate that an MTBF of 10 years is the same as saying

Within ten years, 62% of our chips will fail.

MTBF is *not* operating time!

Let's calculate the MTBF of all SNUG attendees. We can model our population as 2,000 males[6] of age 35 years. The latest CDC life tables [3] give the probability of failure (death) over the next year as 0.001612. So we can expect

$$2{,}000 \times 0.001612 = 3.22 \text{ failures in one year} \tag{27}$$

Then

$$MTBF(\text{SNUG}) = \frac{2{,}000 \text{ units} \times 1 \text{ year operation}}{3.22 \text{ failures}} = 621 \text{ years} \tag{28}$$

The MTBF of SNUG attendees is over 600 years! And yet our lifetime will not be nearly that long.

MTBF is *not* the lifetime of your part!

MTBF is simply the inverse[7] of failure rate λ [13].

For more about the misuse of MTBF see [50].

[6] I do wish we had more women in engineering. That is a problem which is outside the scope of this paper.

[7] The simple way to arrive at our answer: take the inverse of failure rate 0.001612 and get 620 years directly.

5.2. Gray code

We use gray code for our buses. No problem.

Actually this could be a big problem.

The binary reflected code or gray code was invented by Frank Gray of Bell Labs [33]. The advantage of this coding is that only one bit of your multi-bit bus will change state for each change in value. So if you simply run each bit of the bus through its own synchronizer, we will always have a valid value, because on any given clock edge only one synchronizer can possibly go metastable [23], and whichever way that synchronizer resolves you'll have a good value.

This seems very attractive for FIFO pointers, however be aware of these restrictions:

1. Gray code only works for value changes of ±1. If you make larger changes then this scheme will not work. Consider if you write a multi-word packet of data into a FIFO, and don't want to signal the read side until the packet is complete. Now increment the pointer by the number of words in the packet. More than one bit will change in the pointer. Your gray code assumption is violated.
2. Gray code only works for 2^n values. Consider a FIFO with 13 entries. What will happen when you wrap around? More than one bit will change.
3. Be careful with reset. You will have more than one bit change. Can your system handle this?

Figure 25 shows the logic required to implement a gray code clock crossing for a FIFO pointer:

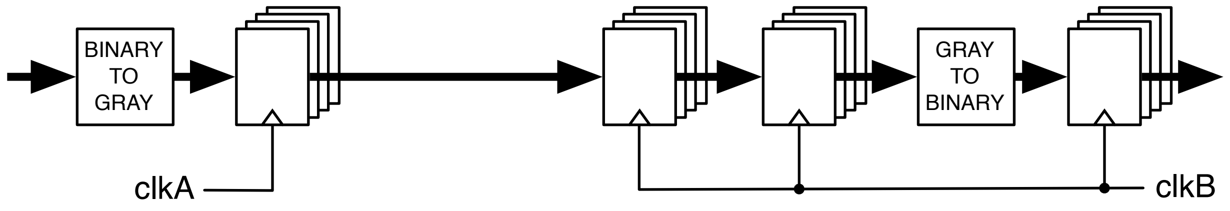

Figure 25: *M*-bit bus clock crossing using gray code

Figure 26 shows the logic required to send an arbitrary binary value across:

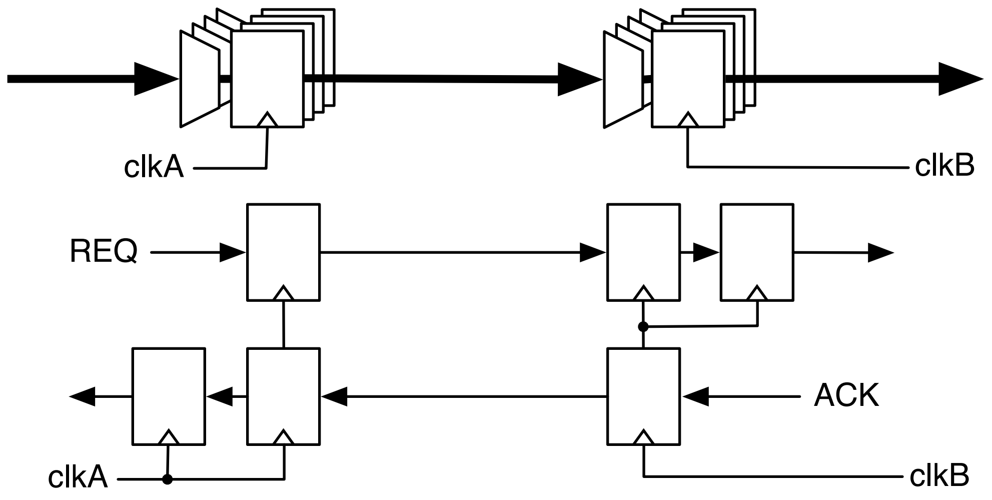

Figure 26: *M*-bit bus clock crossing using binary code

For an *M*-bit bus, the gray code version required 2*M* additional flops, plus the binary-to-gray and gray-to-binary logic. On the other hand, the binary version requires 2*M* muxes[8], about six flops to implement the two handshake synchronizers, and a small handful of gates to generate the REQ, ACK, and load enable signals. In almost all cases the binary version will have less logic.

In the binary version, the *M*-bit data value is stable when it is loaded into the destination clock domain, so there is no chance of metastability in those flops. The gray code version requires *M* synchronizers whereas the binary version has only two. Fewer is better.

It is possible to simply count in gray code, however generally you want binary values so you can do arithmetic on the pointers to calculate an ALMOST_FULL signal for the FIFO. Thus you must use binary counters, and then convert binary-to-gray and gray-to-binary. This extra logic is not needed by the binary version.

Since you need a standard methodology for sending arbitrary data buses across clock domains, why not just use it here [29]? Create a parameterized module to implement the logic of Figure 26

[8] Or, use flops with load enable.

and use it throughout your design, including for all your FIFO pointers. There is really no need to have a special-case gray code synchronizer.

5.3. Related clocks

Of course these two clocks are asynchronous.

One of the fundamental assumptions of our MTBF calculation is that the clock and data are asynchronous. Recall that Equation (2) shows that the probability of entering metastability is the metastability window T_0 expressed as a fraction of the clock period $t_c = 1/f_c$. However this only holds if the data event can appear anywhere in the sampling clock period with *equal probability*. In other words, the probability density function is constant across the clock period:

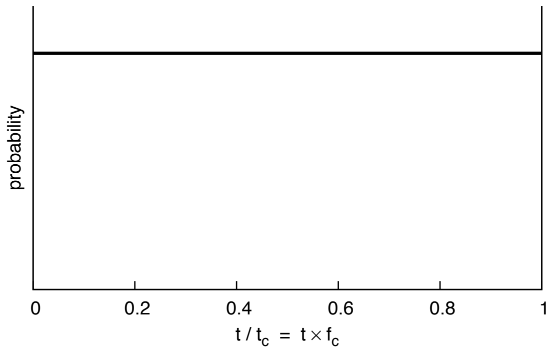

Figure 27: Probability density function showing possible data arrival times during sampling clock period, asynchronous clocks

But what if the clocks are related? If the clocks are derived from the same reference source (oscillator) then they are *coherent* clocks and the probability density function is *not* uniform [7].

Consider a data rate of 125MHz and clock sampling rate of 150MHz, where both clocks are derived from the same source. The frequencies of these two clocks are a fixed ratio; they are *rational* clocks. The data can only arrive at fixed locations in the clock period. The probability density function is a series of spikes [7].

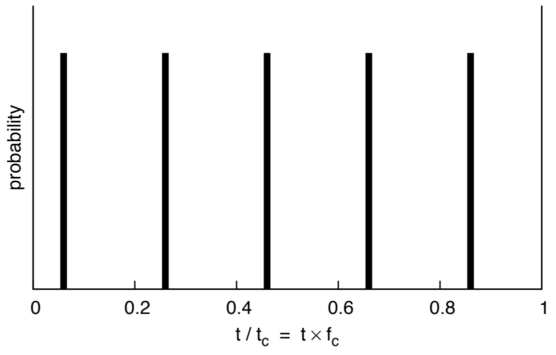

Figure 28: Probability density function for data rate $f_d = 125$MHz and clock rate $f_c = 150$MHz

What does this do to our MTBF calculation? Depending on the phase difference, it's possible for our data to *always* arrive during the metastability window, and we may see MTBF worsen by *several orders of magnitude* [7].

Understanding the clock relationship is critical to an accurate calculation of MTBF. Dally and Poulton [24] give an excellent discussion of various signal-clock relationships: synchronous, mesochronous, plesiochronous, periodic, asynchronous. They provide synchronizer designs appropriate for each condition. Also refer to the discussion by Ginosar [30].

Be very careful with your clock specifications. You may design a subsystem assuming two clock domains will be asynchronous, and then at the system level they end up being driven by coherent clocks. This can happen when a future chip reuses your subsystem and those designers are unaware of your clocking assumptions…

5.4. Flop selection

I'm using the flop that Design Compiler selected.

Design Compiler can do amazing and wonderful things, but picking the proper synchronizer flop is not one of them. Your synthesis tool is not thinking about metastability when it evaluates its cost function.

In particular, the absolute worst type of flop to use is a dynamic pass-gate flop:

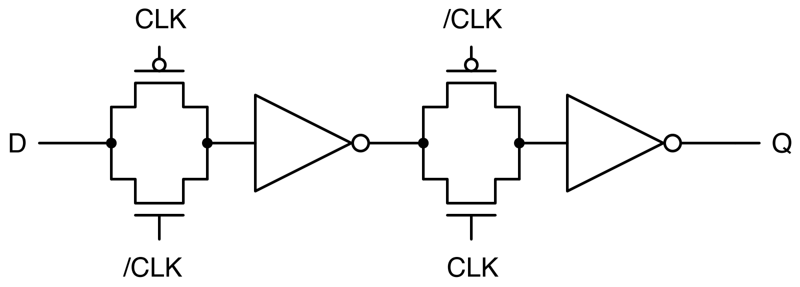

Figure 29: Flip-flop implemented with pass gates

This is a nonrestoring flop. There are no cross-coupled gates! There is no amplification, τ is effectively infinite and there is no metastability resistance at all. Very bad choice for a synchronizer.

In general, the best choices will be flops with higher power, faster CLK→Q delay, and the simplest logic (e.g., no reset, no scan, no input mux). These tend to have the highest gain and thus smallest τ. But rely on these heuristics only if you must; measurement is always best.

5.5. Basic two-flop synchronizer

One clock period is long enough to wait.

Not necessarily.

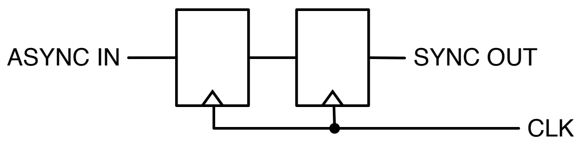

Figure 30: Basic two-flop synchronizer

There is nothing magical about the classic two-flop synchronizer. One clock period might not be enough resolution time t_R to give you adequate MTBF. Your second flop might have large setup time, which further reduces t_R. Your first flop may have bad τ and T_0 parameters. Your layout may put excessive parasitic capacitances on the intermediate net. Your system may require very high reliability. And so on…

See Dally and Poulton [24], Kinniment [42] and Ginosar [29] for many interesting variations of this classic synchronizer. Careful analysis is the only way to be sure that your synchronizer will give sufficient protection against metastability.

5.6. Multistage synchronizer

A two-flop synchronizer isn't enough, so we use three. Maybe four. Five for sure.

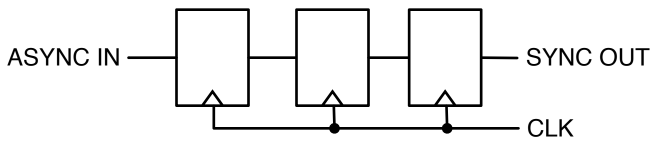

Figure 31: Multistage synchronizer

It's tempting to think that if two flops are good, then three (or more) must be better.

It's tempting when our MTBF is too small, to simply add more flops to the synchronizer.

What is the probability that the second flop will become metastable, and still be metastable after time t_R?

$$P_2(\text{metastable}) = P_2(\text{enter metastability}) \times P_2(\text{still in state after } t_R) \tag{29}$$

But the second flop will only enter metastability if the first flop is still metastable at the end of its resolution time:

$$\begin{aligned} P_2(\text{metastable}) = {} & P_1(\text{enter metastability}) \\ & \times P_1(\text{still in state after } t_{R_1}) \\ & \times P_2(\text{still in state after } t_{R_2}) \end{aligned} \tag{30}$$

Then we have

$$MTBF = \frac{e^{t_{R_1}/\tau_1} \cdot e^{t_{R_2}/\tau_2}}{f_d f_c T_{0_1}} \tag{31}$$

Assuming all flops have identical characteristics[9] we can simplify this to

$$MTBF = \frac{e^{2t_R/\tau}}{f_d f_c T_0} \tag{32}$$

effectively doubling the resolution time.

The correct analysis of MTBF in a multistage synchronizer can be rather complex [5]. Each stage adds additional propagation delay which can decrease the overall resolution time. The reliability may be marginally reduced by back edge effects [42].

[9] A very big assumption!

A preferable alternative may be to use a normal two-flop synchronizer with a slower clock:

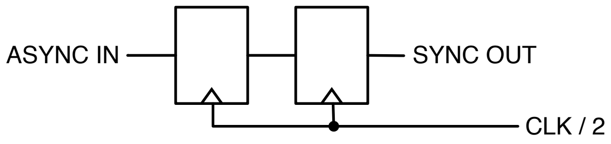

Figure 32: Synchronizer with slow clock

This has the same latency of two clock periods, however each synchronizing latch stage now has *more than twice* the resolution time, because there is no additional propagation delay:

$$MTBF = \frac{e^{(t_c+t_R)/\tau}}{f_d f_c T_0} \tag{33}$$

Rather than using a clock divider, you could gate the clock. Carefully consider the clock duty cycle. It may be better to have a high/low ratio of 3/1, giving the initial master latch at least 3× the resolution time. This deserves further study.

If you must maintain the same throughput[10] you can implement two of these synchronizers running on opposite phases of the slow clock, and combine the outputs at the end.

Do not use flops with load enables or input muxes. You must implement this by gating or dividing the clock, otherwise the metastable latch will try to reload itself through the input mux.

Slower clocks (divide-by-3, -4, etc.) can be used if even longer resolution time is needed.

5.7. Custom synchronizer cells

Our library group designed a special SYNC cell. I'm sure they know what they are doing.

This can be very beneficial if done correctly.

Some things to consider:

How many stages of latches did they implement? Why? Does the feedback loop have high enough gain? Is this a simple latch cell, or does it include complexity like reset and scan? Is there minimal delay between latch stages? Did they isolate the feedback loop from the output load? Did they minimize the propagation delay between stages?

Do you have a Liberty functional model? Do you have a corresponding test functional model? Do you have a scan model? Do you have a timing model? Do they all agree?

Does this library cell require *any* special treatment by *any* of your tools at *any* point during your flow? If so, are you willing to change and maintain your flow just to support this cell?

[10] Think about this carefully. Throughput may be adequate even with a slower clock.

Is this hand-instantiated in the RTL or inserted later? How do you prevent Design Compiler from replacing it with a functional equivalent during synthesis? During place and route? Are you *sure* it hasn't been replaced?

Have you been provided with the metastability characteristics for this cell? Over the entire range of operating conditions? Is this based on simulation or actual silicon? Actual silicon from the current process node, or a previous one?

5.8. Design flow

Sure, we know where all the synchronizer flops are.
Fred maintains a list.
I think he's got that file checked in.

This can be tricky.

Some designers use special dummy cells in their RTL to indicate signals that are crossing clock domains. Or, mark the synchronizer flops with special names, e.g., `_sync` suffix. Then various tools can subject these paths to special treatment—constraining delay, limit transition time, etc.

Alternatively you can use various formal checking and CDC tools to find and verify your clock crossings.

Be prepared to use a combination of approaches. If you are using the "dummy cell" technique, sure enough you will incorporate some IP that doesn't use that scheme, or you use some RTL from that start-up you just acquired…

5.9. Design reuse

This module has been used before. It's proven in silicon.

Many bad things can happen when you reuse a design. Make sure you haven't violated any of the design assumptions regarding metastability.

What clock rates was this designed for? What process and library? What operating conditions?

What assumptions were made regarding clocks? Were they coherent or asynchronous?

What were their reliability goals?

Consider what will happen when someone reuses what *you* are designing *right now*. Are there any assumptions you are making regarding metastability and synchronization? Will future users know what they are? It's all in the spec, right?

5.10. Cell characterization

I trust the characterization of our library. Right down to the picosecond.

What is the precision of your library characterization? What is the accuracy of your library characterization? Do you know the difference between precision and accuracy? What is the accuracy of your timing analysis tool?

How much margin is included in your library characterization? How much margin is included in your fab process models? How much margin is included in your system specification?

We find ourselves in some weird corners of the design space when we investigate metastability. Funny things can happen.

Consider the setup t_{su} and hold t_{hold} parameters of a simple D flip-flop. For a given process corner and operating conditions these are specified as hard numbers. If you violate one of these parameters then you *fail* your timing analysis, right?

Let's plot what happens when we input a pulse on the D pin of the flop.[11]

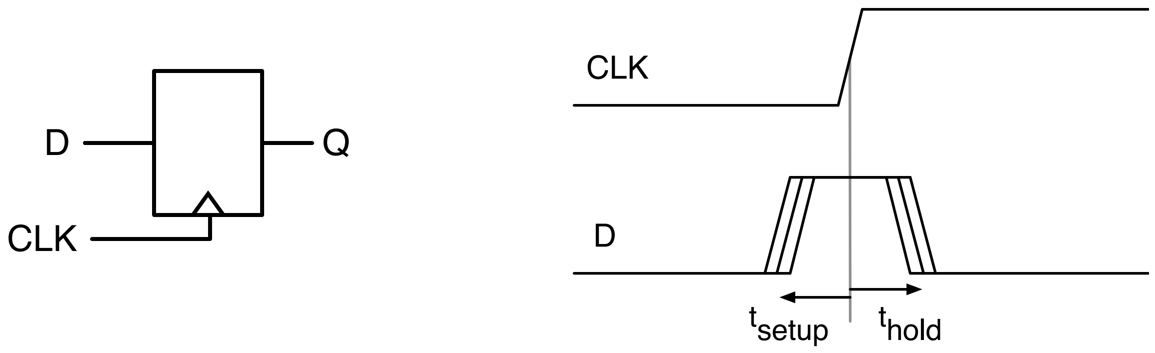

Figure 33: D flip-flop and test waveform

The flop will successfully capture the data as long as the pulse passes the minimum setup and hold parameters, and otherwise we predict it will fail. That's what the cell characterization tells us. We expect a schmoo plot that looks like Figure 34.

[11] This is the same test setup used in section 3.1 to extract τ and T_0.

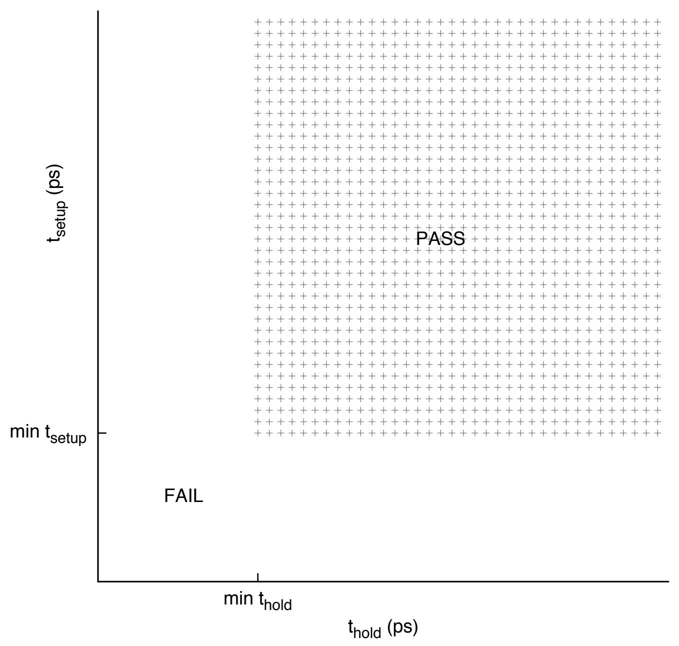

Figure 34: Expected schmoo plot for simple D flip-flop

Now, if you actually go and simulate this using SPICE, you get a slightly different result:

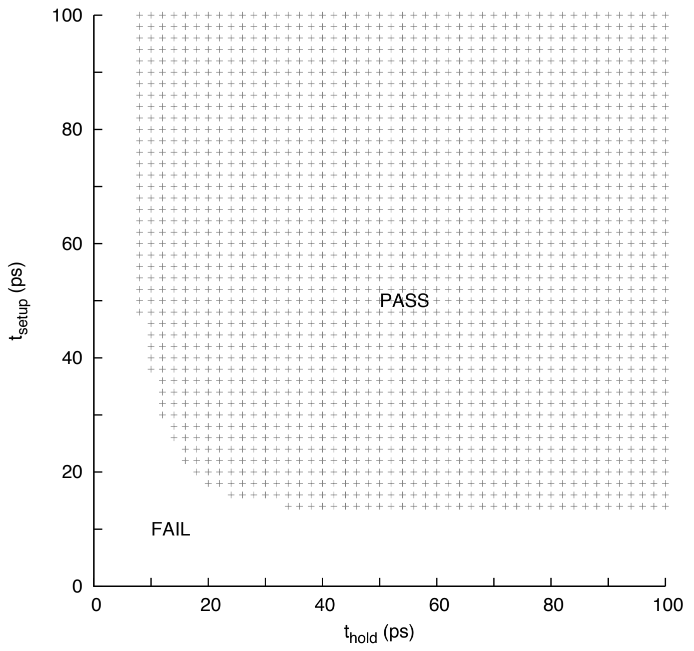

Figure 35: Schmoo plot from SPICE modelling of simple D flip-flop, 65nm

The PASS region is not a simple rectangle at all. If you have a really large setup time then the hold time can be small. Alternatively if you have a large hold time then the setup can be small. It appears the flop data input requires a minimum pulse width, which makes some intuitive sense.

Let's investigate a bit further, and plot how the CLK→Q delay varies.

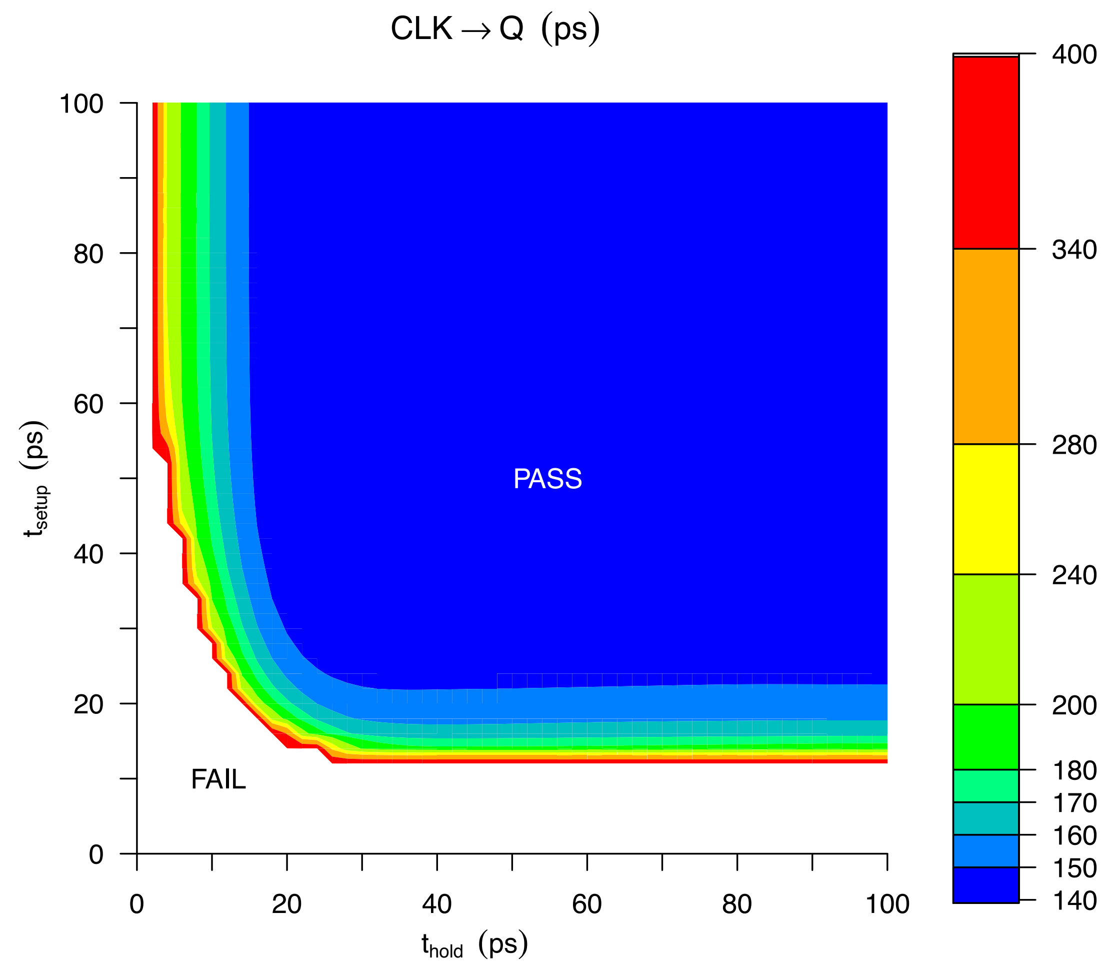

Figure 36: SPICE model contoured schmoo plot for simple D flip-flop, 65nm

As you approach the failure region the propagation delay begins to increase. The flop is nearing metastability. The flop is taking longer and longer to resolve its final value.

Given these results, I wonder what value your library has for minimum t_{setup} and t_{hold}? And what's the maximum CLK→Q delay?

How much margin is included in your library characterization?

By the way, taking a horizontal slice through this dataset at a fixed t_{setup} gives us the flop analysis discussed in Section 3.1.

5.11. Moore's Law

I'm sure at 22nm all these problems will go away.

Actually, no. In fact it started to get worse at 65nm.

The synchronizer time constant τ is usually the same order of magnitude as FO4 (i.e., the typical gate delay with fanout of 4). As process technology scales, τ decreases along with the gate delays getting faster [62]. This keeps our synchronizer reliability manageable as we scale, even as our clock rates increase.

However this τ scaling appears to have stopped at about the 65nm node. Beer et al. [8] reports several measured and simulated values of τ for a variety of process nodes, indicating that τ is no longer scaling with FO4 and actually is getting worse at smaller nodes. Figure 37 plots the ratio of τ to FO4 and shows the degradation of τ with technology scaling.

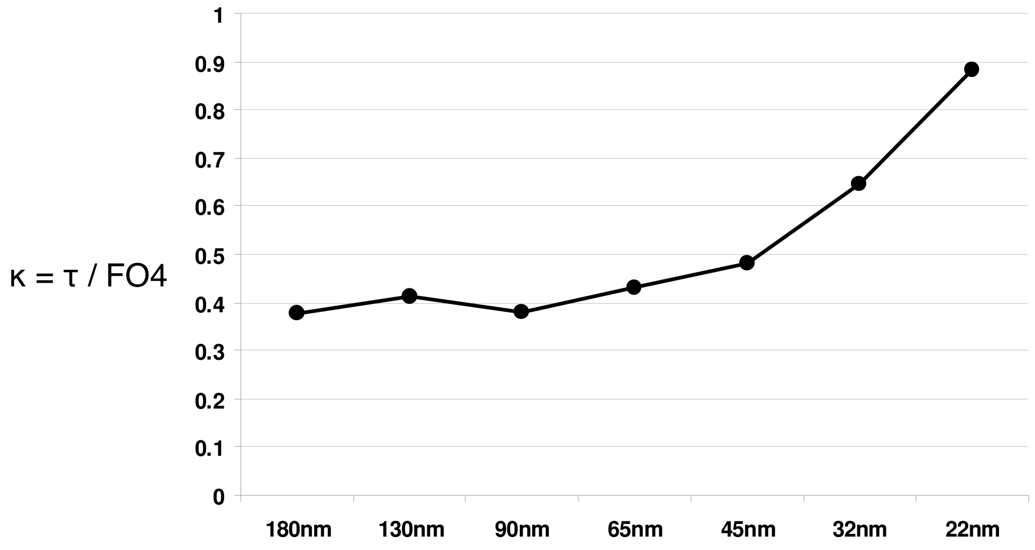

Figure 37: Simulation of κ = τ / FO4 at various nodes
(from Beer et al. [8] figure 19)

Characterize *your* process and *your* cells! Don't rely on rules of thumb.

I wonder what sort of challenges FinFETs may cause?

5.12. I just don’t believe it

I’ve never seen a synchronization failure. I don’t believe such a thing exists.

It’s not about *belief*. It’s about *understanding*.

These failures exist whether you believe it or not [25][45][47].

Synchronization failures are notoriously hard to quantify and capture, because they can leave no discernible evidence.

Imagine what a synchronizer failure might look like in your design. The best mental model is a stuck-at-0 or stuck-at-1 fault on your synchronizer flop—but the fault only lasts for one clock period! How can you test for that? How can you simulate what might happen?

If this flop controls the LSB of color value for a video pixel being displayed on an HDTV, you might never notice (or care). But what if this flop controls the main memory request signal from your CPU?

How often does this failure occur? If it’s several times a second then you will certainly notice in the lab during debug. If it’s only once in a great while, then you may not find out until significant sales have been made…

Beer et al. [7] reports of a failure in a commercial 40nm SoC. The relevant part of the design is shown in Figure 38. The clocks were coherent and at critical ratios caused excessive metastable events at S1.

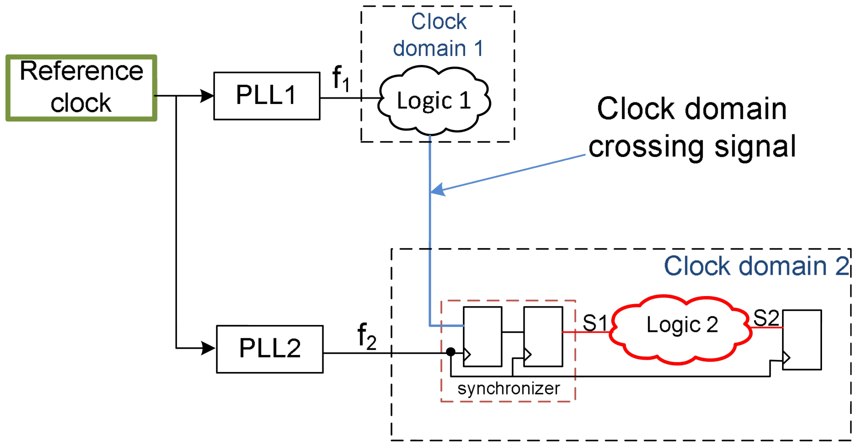

Figure 38: Block diagram showing failing circuitry
(from Beer et al. [7] figure 20)

Using IREM to locate the problem and FIB microprobes to observe, the failure was captured as seen in Figure 39. The synchronizer output S1 shows an unexpectedly short pulse caused by metastability.

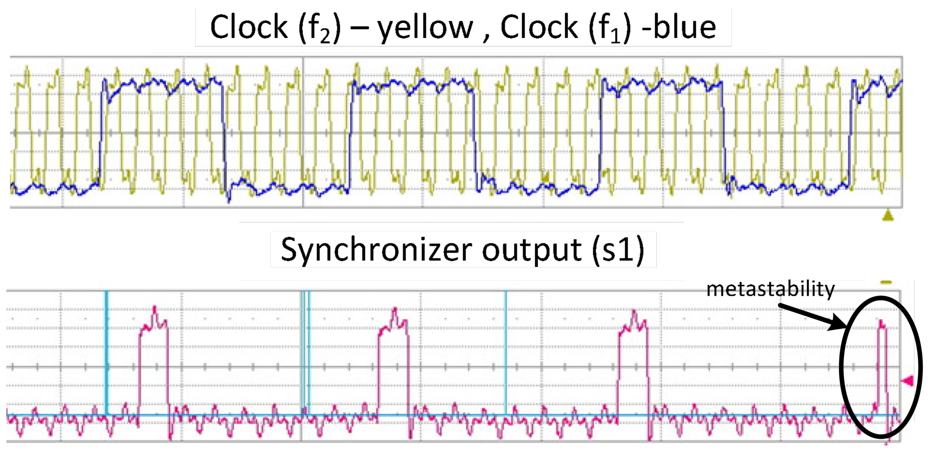

Figure 39: Oscilloscope waveform showing synchronizer failure
(from Beer et al. [7] figure 21)

Blendics [10] also reports on several commercial failures.

Occasionally, brave engineers will write about their experiences with metastability failures [1][48][52].

6. The rest of the story: Part 2

6.1. Example MTBF calculations

Using our example D flip-flop let's calculate the MTBF for a two-flop synchronizer.

Let's assume the following:

τ = 44 ps (from our SPICE simulation)

T_0 = 350 ps (from our SPICE simulation)

f_c = 600 MHz (nice fast design)

f_d = 125 MHz

t_R = $1/f_c$ – setup and propagation delay = 1667 ps – 400 ps = 1267 ps

Assuming 400 ps of setup and propagation delay between the flops, the calculation is then

$$\begin{aligned} MTBF &= \frac{e^{t_R/\tau}}{f_d f_c T_0} \\ &= \frac{e^{\frac{1267\text{ ps}}{44\text{ ps}}}}{(125 \cdot 10^6\text{ Hz}) \times (600 \cdot 10^6\text{ Hz}) \times (350 \cdot 10^{-12}\text{ sec})} \\ &= 122 \cdot 10^3\text{ sec} \\ &\approx 34\text{ hours} \end{aligned} \tag{34}$$

Our MTBF is 34 hours. In less than 2 days operating time, 62% of our flops like this will fail. I imagine we would notice this in the lab.

Now let's modify our synchronizer to use a divided-by-two clock. Our resolution time is now more than doubled to

t_R = 2 × ($1/f_c$) – setup and prop delay = 2 × (1667 ps) – 400 ps = 2934 ps

and the calculation becomes

$$\begin{aligned} MTBF &= \frac{e^{\frac{2934\text{ ps}}{44\text{ ps}}}}{(125 \cdot 10^6\text{ Hz}) \times (300 \cdot 10^6\text{ Hz}) \times (350 \cdot 10^{-12}\text{ sec})} \\ &= 6.9 \cdot 10^{21}\text{ sec} \\ &\approx 220\text{ trillion years} \\ &\approx 16{,}000 \times \text{age of the universe} \end{aligned} \tag{35}$$

Such is the power of the exponential.

6.2. Methodology for constraining synchronizers

Consider our basic synchronizer:

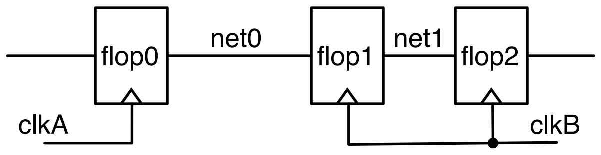

Figure 40: Schematic of synchronizer

We need to create a set of constraints for our flow: Design Compiler (maybe), IC Compiler (definitely), PrimeTime (most definitely).

The physical clocks `clkA` and `clkB` are defined elsewhere, and they will have `set_false_path` or `set_clock_group` applied such that there is no constrained timing path between `flop0` and `flop1`.

The first step is to find all the synchronizers. There are many possible ways:

- In your RTL put a dummy cell on `net0` so you can find it easily
- In your RTL have a special name pattern for synchronizer flops like `flop1`
- In your RTL instantiate a special SYNC cell from your library
- Use a CDC tool to locate all synchronizers
- Use a custom PrimeTime script to locate nets on unconstrained paths
- All of the above

What you want to end up with is an automatically-generated list of synchronizers. For each synchronizer our list should recite the cellnames `flop0`, `flop1`, netnames `net0`, `net1`, and the clock pinnames `flop0/CK`, `flop1/CK`. With those names and a bit of Perl we can create an SDC file to constrain the synchronizer. This SDC file must be automatically generated—you don't want to maintain such a thing by hand. However you only need to regenerate it when the netlist changes; for much of your STA work this will be unchanged [32] and won't add any time to your implementation or STA flow.

Zimmer [64] has a method to dynamically generate the appropriate constraints from inside PrimeTime. But I don't mind generating this SDC file once in a while. And then it can be used for tools other than PrimeTime.

What constraints go in this synchronizer SDC file? Here are some suggestions:[12]

```
create_clock -name sync_launch_clock  [get_pins flop0/CK]
create_clock -name sync_capture_clock [get_pins flop1/CK]
set_false_path -to   [get_clock sync_launch_clock]
set_false_path -from [get_clock sync_capture_clock]
```

With the proper `set_clock_group` we now have a timing constraint from `flop0` to `flop1`. Be sure to select reasonable clock periods.

```
set_max_transition [get_net net0]
set_max_transition [get_net net1]
set_max_capacitance [get_net net1]
```

We want these nets to have reasonably fast edges. We want `net1` to be short and lightly loaded.

There will already be a timing constraint between `flop1` and `flop2` because they have the same physical clock. However `flop1` is a synchronizer flop. We want `net1` to be short and lightly loaded to give more resolution time to `flop1`. How can we fool our implementation tools into making sure this happens? A common solution is to set a custom placement group that keeps the flops together. Another way would be to backannotate a very long CLK→Q delay onto cell `flop1`. This could be done with a special SDF file, also automatically generated.

Note we also need to constrain any data flops that this synchronizer controls (for example, nets `data[n:0]` in Figure 41). Similar techniques can be used to find and constrain these paths.

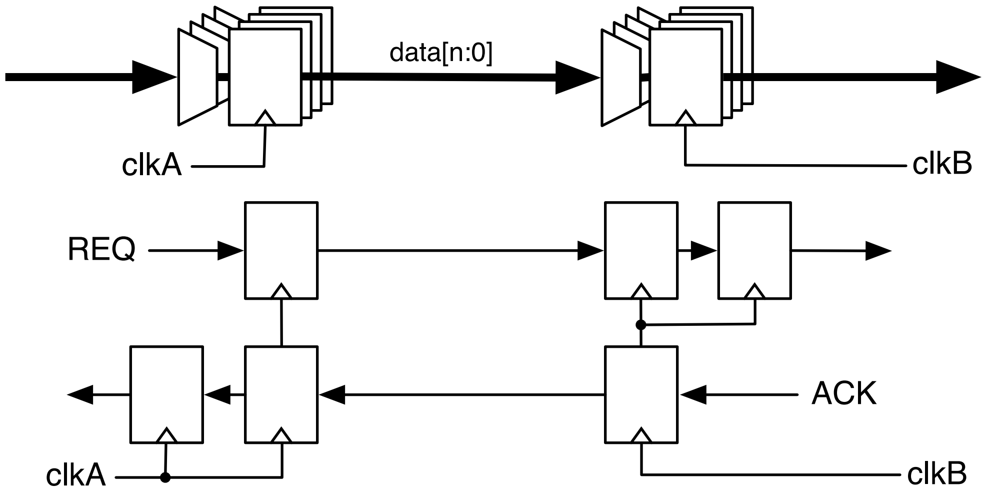

Figure 41: Multibit data crossing clock domains

[12] This is not legal SDC, just a sketch of what you want.

6.3. Methodology for constraining managers

After reading this far, perhaps you suddenly realize that your current methodology for handling metastability and synchronization may be fatally flawed and dangerously inadequate. What shall you do? How can you convince your company's management that your methodology needs to be updated?

This is a big problem, because making any changes could imply that your current methodology must be *wrong*, and all your existing products are just time bombs waiting to *fail* in the field. Ouch. This is not something that you or your managers will want to consider.

Here are some suggested phrases you can use to try and make these changes more palatable:

Our new chips are more complex, with more clocks, and many more clock domain crossings.

Metastability seems to be getting worse below 65nm. τ *may no longer scale with process.*

Running at higher clock rates makes this problem deserving of more careful study.

New process features (e.g., FinFETs) make it prudent to review our methodology.

Focus on your new designs, rather than your old ones!

6.4. Why metastability is a Special Problem

Charles E. Molnar, one of the first to recognize metastability as a serious issue worthy of study, had the following to say [54] about why metastability is a *Special Problem*:

- Metastability breaks most of the conceptual and computational tools that we use from day to day (e.g., binary or two-state circuits)
- Metastability defies careful and accurate measurements
- Metastability can produce failures that leave no discernable evidence
- Metastability can cause failures in systems whose software is "correct" and whose hardware passes all conventional tests
- Metastability involves magnitudes of time and voltage far removed from our daily experience

7. Conclusions

Synchronizer design is a very deep and subtle art, with many counterintuitive features. To be successful you must be willing to put in a *lot* of work. To truly understand metastability and synchronization you may need to be familiar with:

- front-end flow
- digital design
- analog design
- cell library design
- library characterization
- reliability engineering
- high-speed lab techniques
- risk evaluation
- data presentation
- back-end implementation
- statistics
- probability
- combinatorics
- design for test
- fault simulation
- marketing
- sales forecasting
- human psychology

Remember, you cannot achieve 100% synchronizer reliability, so how much is enough? Consider what is the probability of a sea-level cosmic ray causing a bit flip in one of your flip-flops or SRAM cells. Are you willing to accept that much risk?

Nevertheless, do not despair! Help is here. You are now empowered!

- Avoid the Fallacies (see Section 5)
- Follow the Recommendations (see Section 4)

You cannot prevent metastability—but you can *manage it*.

8. Acknowledgements

Grateful thanks to the following for their input and review: Howard A. Landman, Leah Clark, Nick Howorth, Jerry Frenkil, Peter A. Stoll, John F. Wakerly, Robert Grozier, Peter Churchill, Scott Griffith, Jon Stahl, Paul Zimmer. Please note they spoke with me as individual engineers and were not representing their employers. This paper is vastly improved due to their careful comments and suggestions.

Many thanks to the following for helping me chase down references: Cliff Cummings, John F. Wakerly, Peter A. Stoll, Alan Smith, the staff at Library Storage Annex of MIT Libraries, and the reference desk at Concord Free Public Library.

Thank you to the following instructors and professors for allowing me to review their class lecture notes: Gim P. Hom (MIT), Subhasish Mitra (Stanford), Ronald Fearing (UC Berkeley), Shawn Blanton (CMU), Mark A. Horowitz (Stanford). You give your students a great start!

Special thanks to Howard A. Landman for the tutorial on programming and use of R.

As always, these are my thoughts only.

9. References

If you are just starting to learn about metastability, begin with Ginosar [30]. Move on to Kinniment [42]. From a systems perspective, Dally and Poulton [24] cover a variety of clocking schemes and synchronizer strategies.

It's a pleasure learning from the old masters: anything from Thomas Chaney or Charles Molnar. Start with [49] and [17].

Ian G. Clark maintains an excellent online bibliography [18].

I found PhD theses by Portmann [53] and Zhou [62] to be very helpful.

The IEEE ASYNC conference [4] has many great papers about synchronizers and metastability. Let's hope more of this research leaves academia and makes its way to practicing engineers.

[1] Aluru, Shashi. "Synch Probs W/ 2 Clock Domains W/ Close Freqs." *ESNUG*. 281 Item 6, February 19, 1998. http://deepchip.com/items/0281-06.html

[2] Anthony, Sebastian. "Inside DreamHack, the 12,000-computer LAN party." *ExtremeTech*, November 30, 2011. http://www.extremetech.com/extreme/107245-inside-the-worlds-largest-lan-party

[3] Arias, Elizabeth, "United States Life Tables, 2009." *National Vital Statistics Reports* 62 no. 7 (2014): 11.

[4] *ASYNC IEEE International Symposium on Asynchronous Circuits and Systems*, IEEE. http://asyncsymposium.org/

[5] Beer, Salomon, Jerome Cox, Tom Chaney, and David M. Zar. "MTBF Bounds for Multistage Synchronizers." In *Asynchronous Circuits and Systems (ASYNC), 2013 IEEE 19th International Symposium on* (2013): 158-165.

[6] Beer, Salomon, Ran Ginosar, Jerome Cox, Tom Chaney, and David M. Zar. "Metastability challenges for 65nm and beyond; Simulation and measurements." In *Design, Automation & Test in Europe Conference & Exhibition (DATE)* (2013): 1297-1302.

[7] Beer, Salomon, Ran Ginosar, R. Dobkin, and Yoav Weizman. "MTBF Estimation in Coherent Clock Domains." In *Asynchronous Circuits and Systems (ASYNC), 2013 IEEE 19th International Symposium on* (2013): 166-173.

[8] Beer, Salomon, Ran Ginosar, Michael Priel, R. Dobkin, and Avinoam Kolodny. "The Devolution of synchronizers." In *Asynchronous Circuits and Systems (ASYNC), 2010 IEEE Symposium on* (2010): 94-103.

[9] Bell, C. Gordon, J. Craig Mudge, and John E. McNamara. *Computer engineering: A DEC view of hardware systems design*. Digital Press (1978).

[10] Blendics. "Real-World Examples of Metastability." (2013). http://www.blendics.com/images/Resource_Library/Real%20World%20Examples%20of%20Metastability.pdf

[11] Bolton, Martin. "A guided tour of 35 years of metastability research." In *Western Electronic Show and Convention (WESCON), Program Session*, vol. 16 (1987).

[12] Breuninger, Robert K., and Kevin Frank. "Metastable Characteristics of Texas Instruments Advanced Bipolar Logic Families." *Application note SDAA004, Texas Instruments* (1985).

[13] Brown, Kevin. "Failure Rates, MTBFs, and All That." *MathPages*. http://www.mathpages.com/home/kmath498/kmath498.htm

[14] Catt, Ivor. “Time loss through gating of asynchronous logic signal pulses.” *Electronic Computers, IEEE Transactions on* 1 (1966): 108-111.

[15] Chaney, Thomas J., and Charles E. Molnar. “Anomalous behavior of synchronizer and arbiter circuits.” *Computers, IEEE Transactions on* 100, no. 4 (1973): 421-422.

[16] Chaney, Thomas J. *The Synchronizer ‘Glitch’ Problem*. Technical report no. 47. Computer Systems Laboratory, Washington University, St. Louis (1974).

[17] Chaney, Thomas J. “My Work on All Things Metastable OR:(Me and My Glitch).” (2012). http://www.blendics.com/images/Resource_Library/My%20Work%20on%20All%20Things%20Metastable%20OR%20Me%20and%20My%20Glitch.pdf

[18] Clark, Ian G. “Metastability Bibliography.” http://iangclark.net/metastability.html

[19] Couranz, George R., and Donald F. Wann. “Theoretical and experimental behavior of synchronizers operating in the metastable region.” *Computers, IEEE Transactions on* 100, no. 6 (1975): 604-616.

[20] Cox, Jerry. “Is Your Synchronizer Doing its Job (Part 1)?” *SemiWiki*. June 8, 2013. http://www.semiwiki.com/forum/content/2494-your-synchronizer-doing-its-job-part-1.html

[21] Cox, Jerry. “Ten Ways Your Synchronizer MTBF May Be Wrong.” *SemiWiki*. August 25, 2013. http://www.semiwiki.com/forum/content/2703-ten-ways-your-synchronizer-mtbf-may-wrong.html

[22] Cox, J., T. Chaney, and D. Zar. “Simulating the Behavior of Synchronizers.” (2011). http://www.blendics.com/images/Resource_Library/Simulating%20the%20Behavior%20of%20Synchronizers.pdf

[23] Cummings, Clifford E. “Synthesis and Scripting Techniques for Designing Multi-Asynchronous Clock Designs.” *SNUG San Jose* (2001).

[24] Dally, William J., and John W. Poulton. *Digital systems engineering*. Cambridge University Press (1998).

[25] “Decisions, Decisions” in “Science and the Citizen.” *Scientific American* 228, no. 4 (1973): 43-44.

[26] Eckert Jr, John Presper, and John W. Mauchly. “Electronic numerical integrator and computer.” U.S. Patent 3,120,606, issued February 4, 1964, filed June 26, 1947.

[27] Foley, Clark. “Characterizing metastability.” *Advanced Research in Asynchronous Circuits and Systems, 1996. Proceedings., Second International Symposium on* (1996): 175-184.

[28] Garman, John R. “The ‘bug’ heard ’round the world.” *ACM SIGSOFT Software Engineering Notes* 6, no. 5 (1981): 3-10.

[29] Ginosar, Ran. “Fourteen ways to fool your synchronizer.” *Asynchronous Circuits and Systems (ASYNC), 2003 IEEE 9th International Symposium on* (2003): 89-96

[30] Ginosar, Ran. “Metastability and synchronizers: A tutorial.” *IEEE Design and Test of Computers* 28, no. 5 (2011): 23-35. See if you can find the typo in Figure 11.

[31] Ginosar, Ran. “MTBF of a Multi-Synchronizer System on Chip.” (2005). http://webee.technion.ac.il/~ran/papers/MTBFmultiSyncSoc.pdf

[32] Golson, Steve. “Consistent Timing Constraints with PrimeTime.” *SNUG San Jose* (2009).

[33] Gray, Frank. “Pulse Code Communication.” U.S. Patent 2,632,058, issued March 17, 1953, filed November 13, 1947.

[34] Gray, Paul E., and Campbell L. Searle. *Electronic Principles: Physics, Models, and Circuits*. Wiley (1969).

[35] Haseloff, Eilhard. “Metastable Response in 5-V Logic Circuits.” *Application note SDYA006, Texas Instruments* (1997).

[36] Hohl, Jakob H., Wendell R. Larsen, and Larry C. Schooley. “Prediction of error probabilities for integrated digital synchronizers.” *Solid-State Circuits, IEEE Journal of* 19, no. 2 (1984): 236-244.

[37] Johnson, Howard. “Acceptable Failure.” http://www.sigcon.com/Pubs/edn/acceptablefail.htm

[38] Johnson, Howard. “Flip Flops.” http://www.sigcon.com/Pubs/news/4_2.htm

[39] Johnson, Howard. “Inducing Metastability.” http://www.sigcon.com/Pubs/news/4_4.htm

[40] Johnson, Mark G., [mark@mips.COM] “Arbiter / Synchronizer failure; MTBF” In [comp.lsi], September 15, 1989; Internet.

[41] Johnson, Mark, [mjohnson@netcom13.netcom.com] “Re: MTBF Calculation. Looking for articles on the subject” In [comp.lsi.cad,sci.electronics.design,comp.arch.fpga], December 15, 1997; Internet.

[42] Kinniment, D. J. *Synchronization and arbitration in digital systems*. John Wiley & Sons Ltd (2007).

[43] Kleeman, Lindsay, and Antonio Cantoni. “Can redundancy and masking improve the performance of synchronizers?” *IEEE Transactions on Computers* 35, no. 7 (1986): 643-646.

[44] Kleeman, Lindsay, and Antonio Cantoni. “Metastable behavior in digital systems.” *IEEE Design and Test of Computers* 4, no. 6 (1987): 4-19.

[45] Kleeman, Lindsay, and Antonio Cantoni. “On the unavoidability of metastable behavior in digital systems.” *Computers, IEEE Transactions on* 100, no. 1 (1987): 109-112.

[46] Lubkin, S. “Asynchronous Signals in Digital Computers.” Automatic Computing Machinery, Discussions, *Mathematical Tables and other Aids to Computation*, 6, no. 40, (1952): 238-241.

[47] Marino, Leonard R. “General theory of metastable operation.” *Computers, IEEE Transactions on* 100, no. 2 (1981): 107-115.

[48] McLellan, Paul. “Clock Domain Crossing, a potted history.” *SemiWiki*. March 3, 2011. https://www.semiwiki.com/forum/content/396-clock-domain-crossing-potted-history.html

[49] Molnar, Charles E. “Metastability Lecture.” video lecture on February 11, 1992, at HP Corporate Engineering, Palo Alto CA http://www.cse.wustl.edu/history/molnar_c/metastability/metastability-lecture.html

[50] “No MTBF: A site devoted to the eradication of the misuse of MTBF.” http://nomtbf.com

[51] Ornstein, Severo M. *Computing in the Middle Ages: A View from the Trenches 1955-1983*. 1stBooks Library (2002).

[52] Peterson, Sigurd. “Solving a metastable mystery” in “Tales from the Cube.” *EDN.com*, February 13, 2013. http://www.edn.com/electronics-blogs/tales-from-the-cube/4406127/Solving-a-metastable-mystery

[53] Portmann, Clemenz Lenard. “Characterization and reduction of metastability errors in CMOS interface circuits.” PhD diss., Stanford University, 1995.

[54] Rosenberger, F. U. “Metastability.” slides for EE463, Washington University, April 4, 2001. https://web.archive.org/web/20060910062500/http://www.cse.wustl.edu/~fred/CLASSES/463FA03/Metastability.pdf

[55] Stoll, Peter A. “How to avoid synchronization problems.” *VLSI Design* 3, no. 6 (1982): 56-59.

[56] Stucki, M. J., and J. R. Cox Jr. “Synchronization strategies.” *Proc. Caltech Conference on VLSI*, (1979): 375-393.

[57] Synopsys. “DW04_sync Variable Input Synchronizer.” *DesignWare Foundation Library Databook, Volume 2*. Release 2001.08 (2001):31-36.

[58] Veendrick, Harry J. M. “The behavior of flip-flops used as synchronizers and prediction of their failure rate.” *Solid-State Circuits, IEEE Journal of* 15, no. 2 (1980): 169-176.

[59] Wakerly, John. “Designer’s Guide to Synchronizers and Metastability, Part I.” *Microprocessor Report* 1, no. 1 (1987): 4-8.

[60] Wakerly, John. “Designer’s Guide to Synchronizers and Metastability, Part 2.” *Microprocessor Report* 1, no. 2 (1987): 4-8.

[61] Wakerly, John F. *Digital design: principles and practices*. Prentice-Hall, Inc. (1990).

[62] Zhou, Jun. “Design and Measurement of Synchronizers.” PhD diss., Newcastle University, 2008.

[63] Zhou, Jun, Ashouei, Maryam, Kinniment, David, Huisken, Jos and Russell, Gordon. “Extending Synchronization from Super-Threshold to Sub-threshold Region.” In *Asynchronous Circuits and Systems (ASYNC), 2010 IEEE Symposium on* (2010): 85-93.

[64] Zimmer, Paul. “‘No Man’s Land’ Constraining async clock domain crossings.” *SNUG Silicon Valley* (2013).