Managing Synchronizer MTBF

ALJ001 (V1.0) February 25, 2009

ABSTRACT • The typical article on metastability in synchronizers delivers volumes about the MTBF equation but falls short when explaining practical ways to manage the failure rate. This article reviews synchronizer MTBF and encourages the logic designer implementing synchronizers to employ physical constraints to ensure the design minimizes failure rate. It finishes with an example synchronizer implemented in VHDL w/ Xilinx constraints.

KEYWORDS • MTBF, metastability, FPGA, Xilinx, logic design, synchronizer, VHDL, timing constraint, asynchronous, clock domain crossing

ALJ001

Managing Synchronizer MTBF

I sat on the fence for sometime trying to decide whether to write this article. Many authors have tackled metastability - in the popular press, via the web and of course in research quality journals. Yet, it seemed like something was missing based on my experience reviewing code. An explicit guide to managing metastability and the mean time between failure ("MTBF") of the synchronizer seemed appropriate.

I begin with a look at the standard expression of synchronizer MTBF and what assumptions are made in its construction that apply to FPGA logic.

From there I move to how the synchronizer MTBF might be left to chance in the capable but potentially misguided hands of your place and route tool. Finally I offer two approaches to explicitly controlling the MTBF in your synchronizer code when targeting FPGA devices from Xilinx.

Synchronizer MTBF

The meaning of synchronizer failure

A synchronizer is a pair of edge triggered flip-flops or level sensitive latches which are intended to capture an asynchronous event edge transition into a receiving clock domain thus *synchronizing* it.

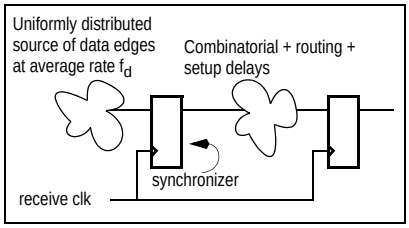

Figure 1 **Two stage flip-flop synchronizer**

Due to the inescapable reality of device physics it is possible that the arriving event edge may be so close to the receiving domain's clock edge that the feedback process driving the cross-coupled gates in the master latch does not have the necessary energy to complete the transition. The latch then comes to rest at a precariously stable (or *metastable)* state halfway between the '0' and '1' voltage levels. The time to exit this state is deterministic (as modeled by second order differential equations) and is a function of how much energy is provided (which relates directly to the amount of overlap between clock and data edges.) The lack of determinism comes from the input process that supplies the data edges and how frequently they will put the device into a metastable position.[1] If the latch output remains at that metastable level longer then some criterion permits then a *failure* has occurred.

Metastability in level sensitive latch circuits

I will not repeat the excellent derivations [Mead], [Kinniment] of the differential equations at the heart of the circuit model (and hence for the MTBF equation derived from that model) for the cross-coupled inverters at the heart of the latch model. But ultimately, even though both offer different formulations they both result in mean time between failure which use an exponential term that relates how much time it takes to exit from the metastable level. Many other sources reference similar equations that differ only in the meaning of the terms. From [Kinniment] the equation is expressed as follows:

$$\mathrm{MTBF} = \frac{e^{\frac{t}{\tau}}}{f_d \cdot f_c \cdot T_W} \qquad (1)$$

Where,

f_d = data edge rate
f_c = synchronizing domain clock rate
T_w = metastability window
τ = feedback loop time constant
t = time at which the synchronizing latch value is sampled

The designer needs to collect these values from some source (manufacturers perhaps or from experiments they design.)

The edge rate of the data being synchronized refers to the number of rising and falling edges present in a given unit of time. Typically the logic designer would simply plug in the source domain clock rate here. If the data toggles at the source clock rate there is one edge per clock.

The synchronizer is connected to a clock with a rate given by f_c.

[Kinniment] explains T_w and τ as follows:

> "The value of T_w is determined by the input time constant θ and the point at which the flip-flop exits from metastability V_e."

> "The value of τ is mainly determined by the feedback loop time constants, and since both T_w and τ are determined by channel conductances and gate capacitances, they are likely to be similar."

The designer needs to obtain estimates T_w and τ either by contacting the device manufacturer or obtaining the values experimentally. [AN219], [Kinniment] and [Alfke] present experimental approaches to determining these values.

Resolution Time

The value of t represents the resolution time *available* for the synchronizer flip-flop output to exit the metastable level. It is at least as great as the clock to out propagation time of the latch. In [AN219] it is presented simply as the clock period of the synchronizer. That may have been an adequate estimate for 1989 MSI/LSI technologies where routing delays (between components) were negligible versus the clock period. Routing dominates the propagation delay in FPGA technologies employed in today's designs. Consequently the logic designer must consider the *worst case slack* in the timing path to the second flip-flop when establishing the time available for resolution[2]. Worse yet, each place-and-route ("PAR") run could implement a different route between the flip-flops of your synchronizer causing t to vary from run to run.

Typically the logic designer relies on static timing analysis with the design constrained by a clock period constraint to guarantee that the slack is positive for all the synchronous elements connected to the receiving clock domain. Such an approach could leave the resolution time at a value close to the nominal (non-metastable) clock to out time of the flip-flop do to excessive routing delay since the PAR engine need only meet the cycle constraint, not exceed it! Remember, once the metastable result resolve to a '1' or '0' it must propagate across routing (delay), meet the setup time of the flip-flop (delay) then get clocked into the second flip-flop. Since the MTBF is an exponential function the MTBF can go from eons to hours or seconds with a seemingly tiny change in routing delay.

Figure 2 plots MTBF as a function of slack to graphically show how increasing the available slack dramatically affects the MTBF of a synchronizer operating at 300 MHz in a Virtex-II Pro™ device characterized in [Alfke].

The plot assumes T_{period}=3333 ps, T_{su}=270 ps, T_{cko}=420 ps and a minimum route of 310 ps. Zero slack (maximum routing) correlates to a

1. This subtle distinction is lost in all of the papers save one that I have reviewed. The assumption of a uniform distribution of input edges within the receive clock period is a fundamental assumption underpinning the construction of the "standard" MTBF presented for synchronizers.

2. Some papers on metastability, notably [GROSSE] include an explicit mention of the routing delay yet others do not.

route of 3333 ps - 1000 ps = 2333 ps of additional routing above and beyond the initial 310 ps.

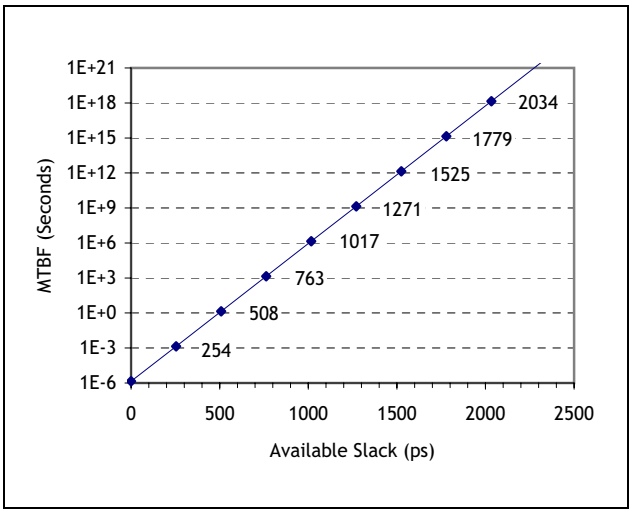

Figure 2 **MTBF versus Slack**

The the value of τ, which equals 36.8 ps, determines the slope of the MTBF curve. That means an MTBF decrease of 1000 times for every τ * ln(1000) = 254 ps decrease in slack!

I think you will agree that, if left unconstrained, the PAR tool could accidently using that additional slack with dramatic consequences for the expected MTBF.

To avoid this lack of rigor in controlling the MTBF, I recommend that the logic designer minimize the routing delay in the synchronizer to drive the slack to the greatest positive value possible in the selected device. That means minimizing the routing delay between the two flip-flops (assuming that neither the setup time nor synchronizer clock period can be changed.)

Maximizing MTBF

Using Tool Constraints

The problem is the uncontrolled routing delay to the second synchronizer flip flop. So what is the logic designer to do?

TYPICAL SYNCHRONIZER CODING. Figure 3 illustrates a typical synchronizer coding in VHDL. Once instantiated your design will get two flip flops connected to some asynchronous source of data. Based on my experience reading the code of others this is all the typical designer ever does. Probably due to suggestions by others that synchronizers just need "two flip-flops in series".

```
LIBRARY std_logic_1164;
USE work.std_logic_1164.all;

ENTITY sync(d, clk: IN std_logic; q: OUT std_logic);
ARCHITECTURE typical OF sync IS
   SIGNAL d1 : std_logic;
BEGIN
   PROCESS (clk)
   BEGIN
      IF rising_edge(clk) THEN
         d1 <= d;
         q <= d1;
      END IF;
   END PROCESS;
END ARCHITECTURE;
```

Figure 3 **Typical Synchronizer Coding**

With the appropriate cycle period constraint on signal "clk" your PAR tool will ensure that the two flops meet the cycle time constraint. As indicated in the last section that is insufficient to "engineer" the MTBF to a desired value.

SOLUTION USING A MAXDELAY CONSTRAINT. Examine Figure 4 illustrates a delay constraint for maximizing the resolution time using a Xilinx "MAXDELAY" constraint, placed directly in the RTL VHDL[1]. It demonstrates a way for the designer to *explicitly over constrain* the path delay so that the slack time and thus MTBF parameter t is maximized.

```
LIBRARY std_logic_1164;
USE work.std_logic_1164.all;

ENTITY sync(d, clk: IN std_logic; q: OUT std_logic);
ARCHITECTURE typical OF sync IS
  ATTRIBUTE maxdelay : string;
  SIGNAL d1 : std_logic;
  -- Maximize the synchronizer's metastability
  -- resolution time by constraining the maximum
  -- routing delay to 400 ps thereby maximizing the
  -- MTBF.
  ATTRIBUTE maxdelay OF d1 : SIGNAL IS "400 ps";
BEGIN
  PROCESS (clk)
  BEGIN
    IF rising_edge(clk) THEN
      d1 <= d;
      q <= d1;
    END IF;
  END PROCESS;
END ARCHITECTURE;
```

Figure 4 **Synchronizer Coding w/ MAXDELAY attribute**

"MAXDELAY" sets the maximum *routing* delay that the circuit can tolerate. Perhaps more importantly, if for some reason the delay cannot be met, the timing analyzer will report a timing path violation.

When the synchronizer is appropriately named and located in its own hierarchical block a timing violation can alert the logic designer or test personnel to the critical nature of that portion of the circuit.

It is important to make the value reasonable and as small as possible. My experiments indicate that 400 ps is a good default. This results in maximum resolution time with a correspondingly high MTBF. Obviously the value you select for each design should be carefully considered.

Alternatively, you could use a VHDL generic in the entity to support a flexible means for establishing the maximum delay and place the synchronizer into a shared component library for all to access.

OTHER CONSTRAINT METHODS. Other physical constraint methods exist which the logic design could exploit to obtain a similar result. For example the Xilinx "HBLKNM" constraint forces flip flops into the same CLB which would typically result in very short routes. However, it can only be applied to flip flop instances and in the end doesn't completely guarantee a fast route. It might give the placer additional, early guidance on what it will have to do to satisfy the MAXDELAY constraint and that could possibly speed up your PAR run. In a similar vain RLOC constraints might also be used to form a relationally placed macro component.

Minimizing Events to Synchronize

[Mead] suggests limiting unnecessary synchronization events by gating the input to the synchronizer with a lower frequency qualifier originating in the receiving clock domain. The cost of the qualifying AND gate is trivial since the CLB/SLICE containing the synchronizer flip-flop will have a LUT available for this.

Ultimately the increase in MTBF bang-for-the-buck is smaller then with controlling t because only the f_d is decreased which is outside the expo-

1. The constraint can also be placed in an external file but this disconnects the necessity of the constraint from the code that requires it. I think it is best to have it in the code.

nential term. I mention this approach only because it may prove useful if f_c is very large (making it harder to get sufficient slack to meet an MTBF goal) and the qualifier occurs infrequently.

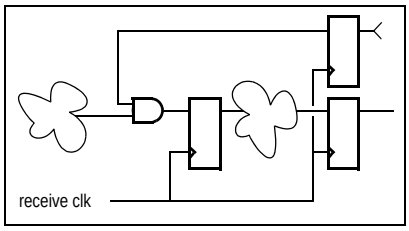

Figure 5 **Synchronizer w/ data filter**

Conclusion

The probability of a synchronizer failing, given enough operating time, is 100%. The logic designer should not fail, however, to explicitly constrain the timing path within the synchronizer to maximize the MTBF at essentially zero cost. That is done by maximizing the synchronizer flip-flop slack time. A MAXDELAY constraint is ideal for that purpose.

In addition, the designer can opt to minimize synchronizing events by gating the asynchronous input with a low frequency qualifier to increase MTBF.

Ultimately, it would be desirable for vendors to supply an MTBF constraint which computes and reports the estimated MTBF for the synchronizer based on internal data for the transistors in their flip flops (given the appropriate design information.) Possibly even doing MTBF driven PAR as a component of a standard timing driven PAR.

References

[AN219] “A metastability primer”, Philips Semiconductor Application note 219, 1989-Nov-15.

[MEAD] Introduction to VLSI systems. Carver A. Mead. Copyright (C) 1980 by Addison-Wesley Publishing Company, Inc.

[KINNIMENT] Synchronization and arbitration in digital systems. David J. Kinniment. Copyright (C) 2007 John Wiley & Sons LTD.

[ALFKE] “Metastability Recovery in Virtex-II Pro FPGAs”. Xilinx application note XAPP094, (V3.0) Feb 10, 2005 by Peter Alfke. http://www.xilinx.com/support/documentation/application_notes/xapp094.pdf

[GROSSE] "Keep metastability from killing your digital design”, Debora Grosse, EDN magazine, 2004-06-23.

Feedback

To provide feedback (of any type) to the author of this article please send an E-mail to Journal@AspenLogic.com. With respect to this article, all feedback emails are considered as being in the public domain prior to disclosure to Aspen Logic, Inc. so *do not* send any confidential or proprietary information to us.

About the Author

Tim Davis is the night time janitor at Aspen Logic where he practices “janitorial engineering” on designs he finds lying around the office. Drop him a line if you need something cleaned up.

Notes