EE Times Connecting the Global Electronics Community

designlines PROGRAMMABLE LOGIC

Blog

Wrapping One's Brain Around Metastability

Adam Taylor, Director, Adiuvo Engineering & Training

11/20/2013 11:20 AM EST

2 comments post a comment

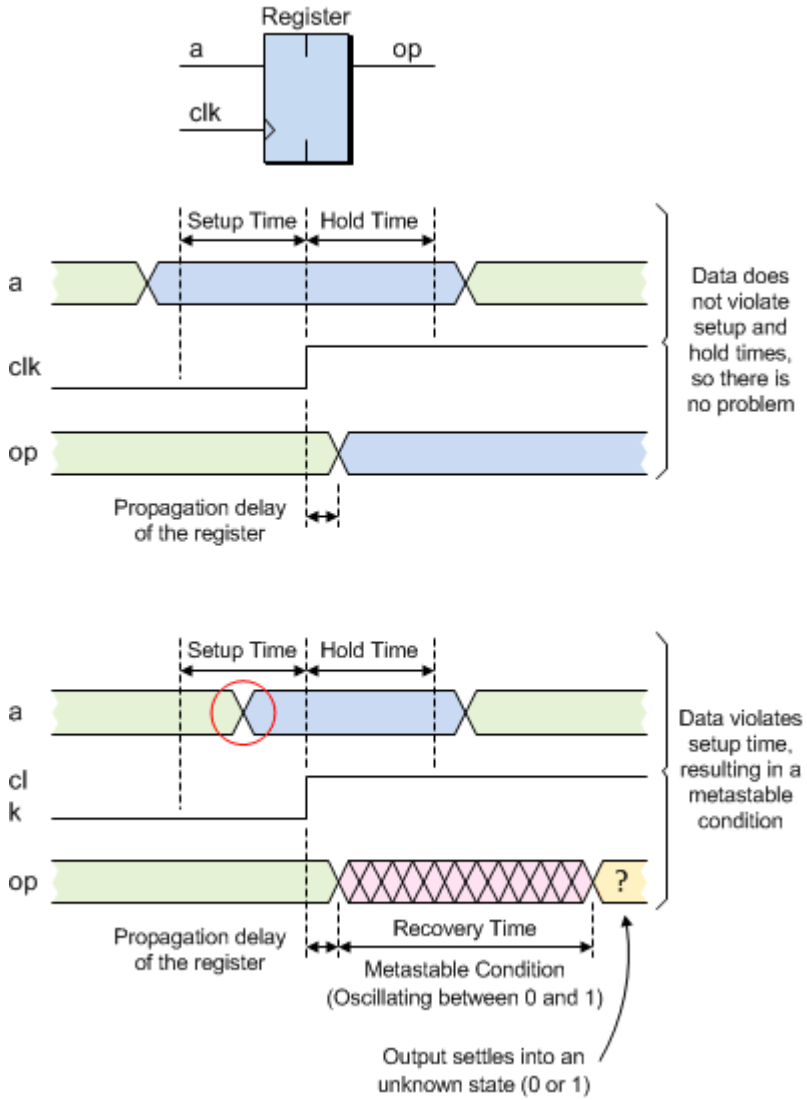
Tweet Share 8 G+

Design expert Adam Taylor explains the mysteries of VHDL-based FPGA design. In this column, he explores how metastability can be triggered in registers.

Part 1: The metastable condition

I recently noticed some mention of the topic of metastability in Handling Asynchronous Clock Groups in SDC. Since the authors only touched on this topic, I decided to more fully explain what metastability is, what causes it, and how we can learn to live with it, since its occurrence cannot be totally prevented.

As illustrated below, metastability can happen to registers when their setup or hold times are violated; that is, if the data input changes within the capture window. As a result, the output of the register may enter a metastable state, which involves oscillating between logic 0 and 1 values. If not treated, this metastable condition may propagate through the system, causing issues and errors. The register will eventually recover from its metastable state and "settle" on a logic 0 or 1 value; the time it takes for this to occur is called the recovery time.

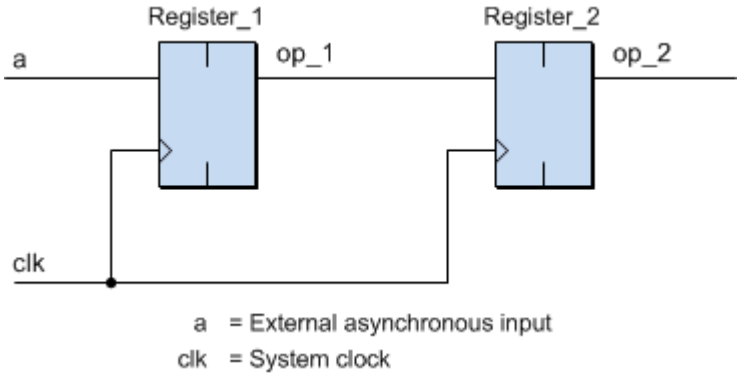


Metastability within an FPGA design will typically occur in one of two ways:

1. When an incoming signal is asynchronous with regard to the clock domain. This may be an external input signal or a signal crossing between clock domains. In this case, the design engineer is expected to resynchronize the signal to address metastability, which is certain to occur eventually. This is where a multi-stage synchronizer is typically employed as discussed below.

2. When multiple register elements in a synchronous design are using the same clock, but phase alignments or clock skew issues mean that the output from one register violates another register's setup and hold time. This may be addressed by modifying the place-and-route constraints or by changing the logic design itself.

Let's consider the case of an incoming signal that is asynchronous with respect to the system clock. It is the engineer's responsibility to create the design in such a way as to mitigate against any resultant metastability issues. Many engineers will be familiar with the concept of a two-stage synchronizer, but I wonder how many really understand just how it performs its magic?

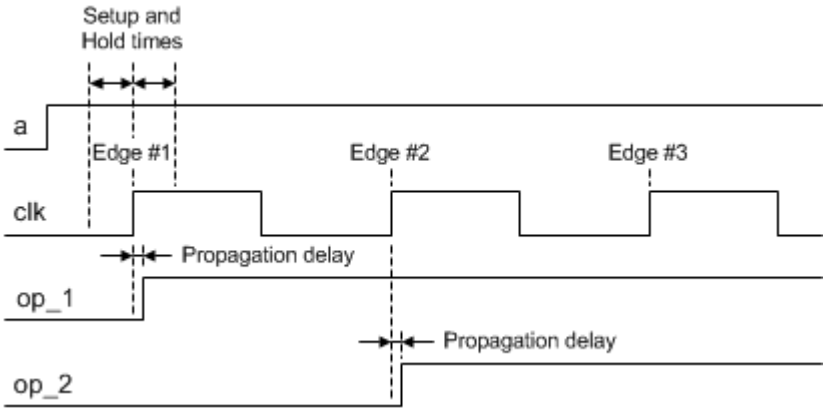


A two-stage synchronizer.

In fact, the two-stage synchronizer works by permitting the first register to enter a metastable condition. The idea is that the system clock is running -- and therefore "sampling" the external signal -- significantly faster than the external signal is changing from one state to another. If it should happen that a transition on the asynchronous signal causes the first register to become metastable, then ideally this register will have recovered by the time the next clock edge arrives and loads this value into the second register.

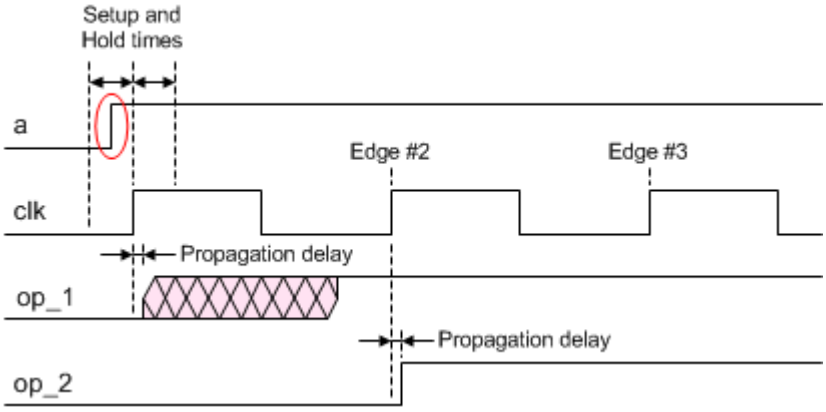
Now, this is where some people become confused. Let's assume that the original value on the asynchronous signal was a logic 0, and that this value has already been loaded into both of the synchronizer registers. At some stage the asynchronous signal transitions to a logic 1. Let's explore the possibilities as follows:

The first possibility is that the transition on the asynchronous signal *doesn't* violate the first register's setup or hold times. In this case, the first active clock edge (shown as "Edge #1" in the illustration below) following the transition on the asynchronous signal transition loads its new value into the first register, and the second active clock edge will copy this new value from the first register into the second as shown below:



Transition on input doesn’t cause any problems.

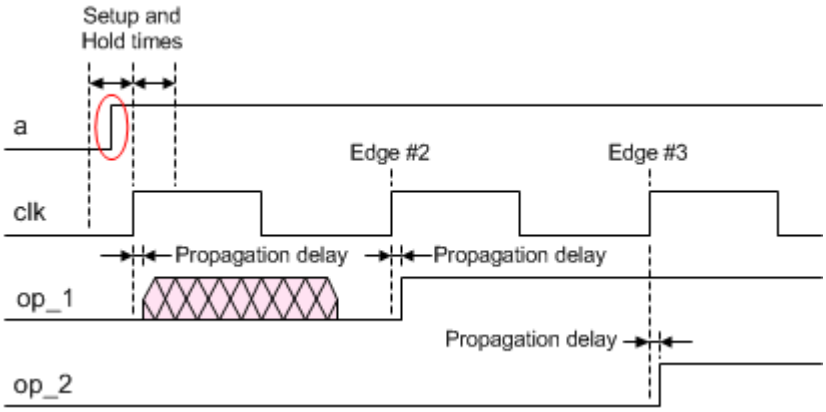
The second possibility is that the transition on the asynchronous signal *does* violate the first register's setup or hold times, which means the first active clock edge causes the first register to enter a metastable state. At some stage -- hopefully before the next clock edge -- the first register will recover, by which we mean it will settle into either a logic 0 or a 1 value. Let's assume that the first register ends up settling into a logic 1 as shown below:



Metastable state settles on logic 1 value.

This is, of course, what we wanted in the first place. In this case, the second active clock edge will load this 1 into the second register (which originally contained a logic 0). Thus, the end result -- as seen at the output from the second register -- is exactly the same as if the first register had not gone metastable at all.

The final possibility (at least, the last one we will consider in this column) is that, following a period of metastability, the first register settles into a logic 0 as shown below:



Metastable state settles on logic 0 value.

In this case, the second active clock edge will load this 0 into the second register (which already contained a 0). At the same time, this second active clock edge will load the logic 1 on the asynchronous signal into the first register. Thus, it is the *third* active clock edge that eventually causes the second register to be loaded with a logic 1.

The end result of using our two-stage synchronizer is that -- in a worst-case scenario -- the desired output from the synchronizer is delayed by a single clock cycle.

Of course, nothing is simple. There is a slight chance that the first register will not recover in time, which might cause the second stage of the synchronizer to enter its own metastable condition. This will be the topic of my next column. In the meantime, I would be delighted to hear of your experiences with metastable conditions -- the problems they caused and the ways in which you resolved them.

Related posts:

- Slideshow: 5 Things I Wish I'd Learned at University
- Creating a Prototype? Consider This
- Alphasat Mega Satellite Launches Successfully
- Continued Professional Development in Engineering
- Design West 2014: Big Shoes & Loud Shirts to Fill
- 3 Things to Consider for Your Prototype

- 6 Things to Consider When Designing Your PCB

EMAIL THIS PRINT COMMENT

Copyright © 2017 UBM Electronics, A AspenCore company, All rights reserved. Privacy Policy | Terms of Service