BY SHAKER SARWARY AND SAURABH VERMA • ATRENTA INC

Critical clock-domain-crossing bugs

AWARENESS OF CDC ISSUES, ALONG WITH THE USE OF GOOD DESIGN PRACTICES AND PROVEN EDA TOOLS FOR CDC VERIFICATION, CAN AVOID COSTLY SILICON RE-SPINS AND SIGNIFICANTLY IMPROVE TIME TO MARKET.

Today's SOC (system-on-chip) designs have dozens of clocks, many of which are asynchronous. This design approach facilitates the convergence of digital-audio, video, wireless, and networking applications in a single chip. CDCs (clock-domain crossings) can cause difficult-to-detect functional failures in SOCs involving multiple asynchronous clocks. Simulation and static-timing analysis often do not detect issues such as metastability and the coherency of correlated signals' CDCs; as a result, these issues often end up as bugs in silicon. Unfortunately, most relevant literature does not adequately cover some of these critical CDC issues, and designers learn about them only after making costly mistakes. Two of the most common and critical issues involving CDCs are improper sequencing of data/enable in enable-based synchronization and data coherency due to the convergence of signals.

ENABLE-BASED SYNCHRONIZATION

A receiver flip-flop output can become metastable if it violates the data/reset setup-and-hold times. This scenario can arise when the transmitter—the source of data—and the receiver flip-flop are in asynchronous-clock domains. To avoid such issues, designers use synchronizers that isolate metastability and deliver a clean signal to the downstream logic. A synchronizer can be a simple double flip-flop. Designers commonly use this technique for a control signal's CDCs. In a data transfer across clock domains, the data is first set up; then, a control signal that synchronizes with the destination domain travels to the destination to enable data capture. Although this data-transfer technique across clock domains is a common and proven technique, it involves pitfalls that require special attention. This technique relies on data to be stable when you assert an enable (**Figure 1**).

Having too low a margin between the data you are setting up and the enable you are asserting may corrupt the data transfer. A good way to prevent such problems is to design a full handshake when you set up the data. In this approach, you assert and synchronize the request in the destination domain and adequately assert an acknowledge to let the next data load occur. This approach might add a few cycles of latency, but it avoids functional failures.

Glitches are other sources of worry across clock domains. Typically, any combinational logic may be subject to short-lived glitches. These issues are generally harmless because they resolve themselves when you activate the next clock edge. Although these issues are not problematic for synchronous transfers, a glitch may occur with asynchronous crossings if you activate a destination clock. The design may therefore receive a glitch as a pulse, causing a functional failure. For this reason, it is important to avoid using any combinational logic that may cause glitches on a CDC path. You

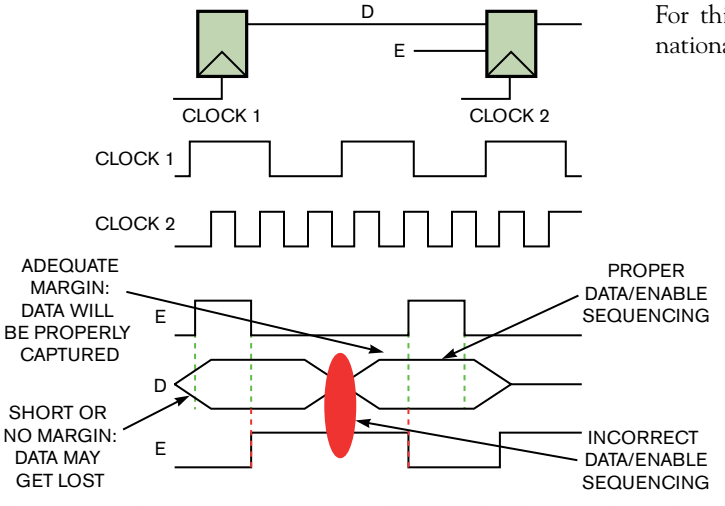

Figure 1 In a data transfer across clock domains, the data must be stable when enable is asserted. Too short of a margin between data setup and enable assertion can result in data corruption.

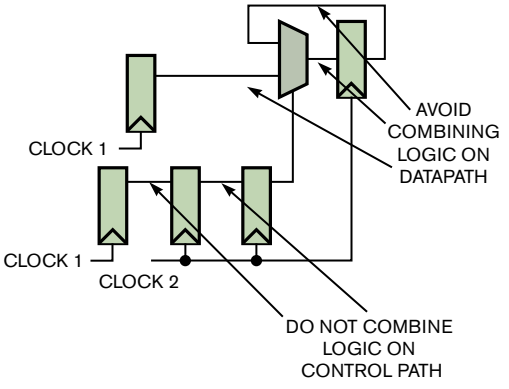

Figure 2 A good design practice is to avoid using any logic, except the recirculation-multiplexer logic, which is part of the enable flip-flop, on the datapath CDCs.

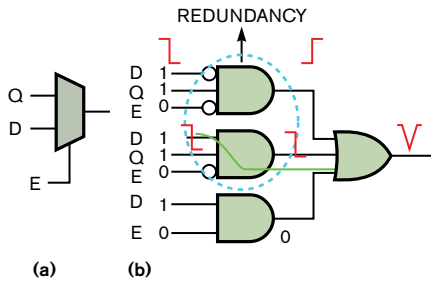

Figure 3 You can map a simple, glitch-free multiplexer (a) with AND and OR gates that can create glitches (b).

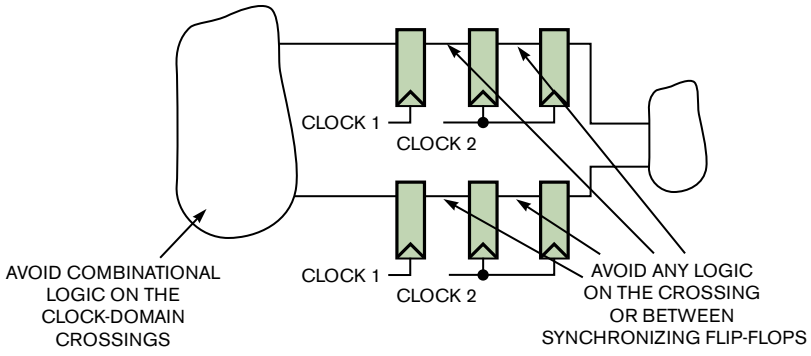

Figure 4 Any glitch in the Gray encoder may cause a functional failure in the design.

should perform any computation either before crossing clock domains or after the destination domain captures the signals.

Glitches may affect both control and data CDCs. In a data transfer, a glitch may affect the enable line or the data line; both present risks affecting safe data transfer. You must synchronize the enable logic in the destination domain and avoid using combinational logic after synchronization. Glitches on the datapath may be harmful, too. A good design practice is to avoid using any logic, except the recirculation-multiplexer logic, which is part of the enable flip-flop, on the datapath CDCs (**Figure 2**).

Although this data-synchronization scheme is the most common, many variations of enabled-data crossing involve an enable signal with combinational logic. Occasionally, designers use an enabled AND instead of a multiplexer or combine the multiplexer with other combinational logic on the datapath. They rely on the enable signal to ensure that data synchronously transfers to the destination and that glitches do not occur. As designers become more creative and use extra logic in enabled-data crossings, they expose their designs to glitch risks that are difficult to detect. To comprehend these risks, consider a simple example of a glitch-free multiplexer; you can implement this multiplexer so that it can create a glitch. Downstream tools, such as synthesis, optimization, and technology mapping, can transform the circuit and introduce logic that can cause a glitch and thus cause a functional failure. You can map a simple, glitch-free multiplexer with AND and OR gates that can create glitches (**Figure 3**).

Although this transformation may seem unlikely with a stand-alone multiplexer, it may well occur if you introduce more logic on the datapath. Synthesis and optimization tools may identify opportunities to increase timing performance, reduce area, or decrease power consumption by combining multiplexer logic with other logic on the path; however, these tools may also create a final implementation prone to glitches. To avoid such problems, you should control the use of these tools to avoid such transformations. Unfortunately, designers often fail to consider these details when creating and implementing a design. Furthermore, a glitch is not an easily predictable event; simulation or static-timing verification cannot detect a glitch on an asynchronous crossing. Once the symptom appears in silicon, it is difficult to perform a root-cause analysis. It takes significant effort and time to link silicon failures to a glitch on a CDC. Static-CDC analysis is better for systematically catching and reporting such issues and avoiding costly silicon re-spins.

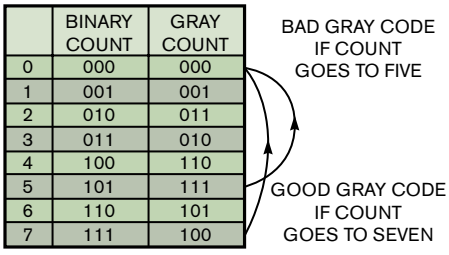

Figure 5 A Gray encoder targeting counting from zero to seven for a full 3-bit counter will fail when the pointer moves from five to zero.

DATA COHERENCY

Another critical issue involving asynchronous clocks is the coherency problem due to convergence of independently synchronized signals. CDCs introduce latency and cycle-level uncertainty, even with synchronized crossings. Although synchronizers isolate metastability and ensure that a "clean" signal travels to downstream logic, they cannot prevent latency. Coherency problems occur when two correlated, separately synchronized signals cross clock domains; each synchronizer introduces a different latency factor due to the CDC. If one of the signals captures a transition, metastability settles to the correct value in the first cycle, whereas the other signal captures a transition in the next cycle. That is, metastability settles to an incorrect value, and you must wait for the next clock cycle to capture the transition. Then, you will observe an incorrect set of values at the destination for at least one cycle. If the signals represent a state variable, then you will observe an unknown or unwanted state at the destination. This unknown state causes a functional failure in the design.

This problem is one of the most common in CDC, and it is becoming more important as designs become larger. Design reuse and IP (intellectual-property) integration may create convergences of which designers may be unaware. To avoid coherency problems—assuming that you know the convergences—you should use correlated signals so that they change values at different times. You must use Gray encoding to correlate signals that are CDCs. This scenario occurs when FIFO point-

ers cross clock domains to compute empty and full flags. You Gray-encode the binary counters, transfer to the other domain, and then convert the counters back to binary before using them. Occasionally, designers access pointers in a FIFO block to do empty/almost-empty or full/almost-full flag calculations. This practice may create CDCs, convergences, or both that a designer may overlook. Adopting standard practices prevents the introduction of CDC bugs into the design.

MORE AT EDN.COM

Go to www.edn.com/ms4271 and click on Feedback Loop to post a comment on this article.

Gray-encoding circuitry seems simple; however, errors can easily slip into a design. You must Gray-encode and register the signals before crossing clock domains. Sending Gray-encoded signals directly to the destination domain defies the purpose. Furthermore, any glitch in the Gray encoder may cause a functional failure in the design (**Figure 4**).

Another subtle issue is mismatch between Gray-encoding assumptions and the binary-counter range. Designs sometimes fail when a designer expects a Gray counter targeting the full range of a 4-bit counter to count to lower counts and loop back to zero. For example, a designer can build the write pointer of a six-layer-deep FIFO to count from zero to five and loop back to address zero. A Gray encoder targeting counting from zero to seven for a full 3-bit counter will fail when the pointer moves from five to zero (**Figure 5**).

Designing a Gray encoder may give a false sense of security if you fail to account for these details. Both junior and experienced designers may face such issues. There are a large number of corner-case problems in CDC, and it is difficult for any designer to pay attention to all the details, especially when under tight schedule pressure. The best way to catch these issues is to approach them with a systematic methodology that has concise metrics. Static-CDC verification has recently emerged as an accepted approach to achieve this goal. This approach targets metastability, convergence, and other CDC issues that traditional verification tools, such as simulation and static-timing verification, do not cover. Static-CDC verification successfully targets corner cases that designers may overlook. Furthermore, it provides a systematic-verification approach that can fit into any design flow as part of the verification-sign-off tool suite. EDN

AUTHORS' BIOGRAPHIES

Shaker Sarwary is technology director at Atrenta (San Jose, CA). He has a doctorate from Paris University (France), and he has performed postdoctorate work at the University of California—Berkeley. He has held senior engineering positions in the areas of synthesis and verification at Lattice Semiconductor, Get2Chip, and Cadence. You can reach him at shaker@atrenta.com.

Saurabh Verma is an engineering manager at Atrenta. He has a bachelor's degree from Indian Institute of Technology Kanpur. He has rich experience in formal technology and rule-based-design verification. You can reach him at saurabhv@noida.atrenta.com.