BY ASHISH GOEL AND ATEET MISHRA • FREESCALE

# Latches and timing closure: a mixed bag

LATCHES HAVE THE EDGE OVER FLIP-FLOPS IN HIGH-FREQUENCY DESIGN. HERE ARE SOME HINTS ON APPLYING THEM TO YOUR NEXT DESIGN.

Digital blocks contain combinational and sequential circuits. Sequential circuits are the storage cells with outputs that reflect the past sequence of their input values, whereas the output of the combinational circuits depends only on the current input. Latches and flip-flops are common storage elements for these blocks.

A latch is a level-sensitive storage cell that is transparent to signals passing from the D input to the Q output and that holds the values of D on Q when the enable signal is false. Depending on the polarity of the enable input, latches have either a positive level or a negative level. A flip-flop, on the other hand, is an edge-triggered device that changes state on the rising or the falling edge of an enable signal, such as a clock. In a rising-edge-triggered flip-flop, the flip-flop samples its input state only at the rising edge of the clock. It then maintains this sampled value until the next rising edge of the clock. Designers typically prefer flip-flops over latches because of this edge-triggered property, which simplifies the behavior of the timing and eases design interpretation.

Latch-based designs, however, have smaller dice and are more successful in high-speed designs in which the clock frequency is in the gigahertz. In flip-flop-based high-speed designs, maintaining clock skew is a problem, but latches ease this problem. Hence, the use of flip-flops can limit the design's performance when the slowest path limits the frequency of the design. When you consider process variation, latch-based design is dramatically more tolerant of variations than is flip-flop-based design, resulting in better yield, allowing more aggressive clocking than the equivalent design with flip-flops, or providing both of these benefits.

## USING LATCHES TO BORROW TIME

Latches' biggest advantage is that they allow a sufficiently long combinational path, which determines the maximum frequency of the design, to borrow some time from a shorter path in subsequent latch-to-latch stages to meet its timing goal. A level-sensitive latch is transparent during an active clock pulse. The time-borrowing technique can also relax the normal edge-to-edge timing requirements of synchronous designs (**Figure 1**).

A sample circuit has two timing paths (**Figure 2**). Path 1 goes from a positive-triggered register (1) to a negative-level latch (2). Path 2 goes from the latch to a positive-edge-triggered register (3). In the **figure**, borrowing compensates for the delay through the logic cloud (A). The logic in Path 1 incurs a delay, and, depending on the length of that delay, two possible scenarios of timing analysis can emerge. These scenarios decide how much time the design can borrow (**figures 3** and **4**).

In **Figure 3**, data arrives from Logic A at Latch 2 before the falling edge of the clock at the latch. In this case, the behavior of the latch is similar to that of a flip-flop, and the analysis is simple. You need not borrow any time to achieve your timing goal. In **Figure 4**, the negative clock edge enables the latch before the arrival of the signal from Logic A at the input of the latch, so the latch enters transparent mode and for a time transmits an undefined state from Logic A through to Register B. It is important that the new state from Logic A reaches and passes through Logic B in time to meet the setup requirements of Register 2. So, if Logic B has a short propagation delay, you can, in effect, let Logic A have some of the time you reserved

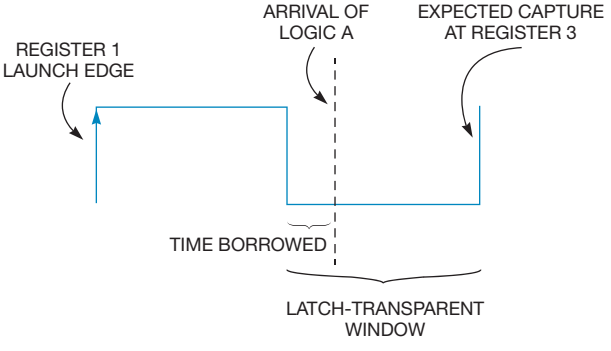


Figure 1 The time-borrowing technique can relax the normal edge-to-edge timing requirements of synchronous designs.

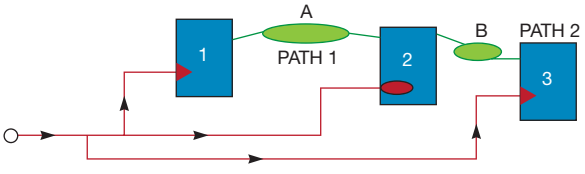


Figure 2 Path 1 goes from a positive-triggered register (1) to a negative-level latch (2). Path 2 goes from the latch to a positive-edge-triggered register (3).

for Logic B, and the circuit will still work. Logic A borrows this extra time to complete its propagation delay. When Path 2 is timed, the timing analysis considers the end of the borrowed time as the starting point for analyzing Logic B's delay.

Static-timing analysis generates timing reports according to the examples shown in **figures 3** and **4**. However, the timing when the latch is enabled is the same as if the latch were simply a transparent delay element (**Figure 5**).

## TIME-BORROWING IN OCV

In an ideal scenario, the time at the starting point should equal the time the latch borrows. Due to shrinking process technology, however, OCV (on-chip variation), signal-integrity, and other factors come into play. To increase the accuracy of the analysis, you can also use CPPR (common-path-pessimism-removal) techniques. These factors complicate the relationship between time-borrowing and time for the starting point. As a result, the timing analysis of latches becomes more challenging.

Returning to **Figure 4**, you'll note an interesting relationship between time-borrowing and the starting-point time. The variables include clock uncertainties, clock-path pessimism due to OCV, and clock derating. During the timing of Path C, $T_{SP}=T_B-U+CPPR$, where $T_{SP}$ is the time for the starting point and $T_B$ is the time Path 1 borrows when constraining Logic A.

Applied uncertainty in Path 1 is the uncertainty for the clock path of the latch, which is not part of pessimism when the latch is transparent. You thus remove that pessimism about the latch clock's uncertainty from the start time. Similarly, you recover pessimism due to CPPR in the time for the starting point because the same early or late path type of latch-launch clock path is in Path 2. If you want to apply clock derating in the design during the timing of Path 2, you should consider using early rather than late derating to make the path the same as the capture clock of Path 2.

EDA tools usually exhibit pessimistic behavior when timing Path 2 because they don't consider CPPR, but they should not apply that pessimism. Path 1's clock-path pessimism ends during calculation of the start-point time, and again, you should not retain this pessimism. The latch is transparent, so it acts as a combinational cell. In this case, you should consider using CPPR between the starting point of Path 1 and the ending point of Path 2. These tools yield extremely pessimistic results because they fail to consider that the use of pessimism is acceptable.

You can also consider using the smallest value between the CPPR of Path 1 and that of Path 2. This approach is not the most accurate, but it provides another level of pessimism removal. Comparing the common clock path of the register and the latch in timing Path 1 versus the common clock path of the latch and the endpoint—the second register—in timing Path 2 can give an idea of the minimum possibility of the clock path between the register and the final endpoint.

Once you ensure that the latch will be transparent during path timing, the least preferred, most accurate, and best way to judge the timing of latches is to make the latch transparent by using a case analysis on the enable pin of the latch. After this step, the EDA tool can time the two segments as one complete path. This method is the least preferred because the latch may not always be transparent when timing Path 1 in the best-case condition: when time borrowing is unnecessary.

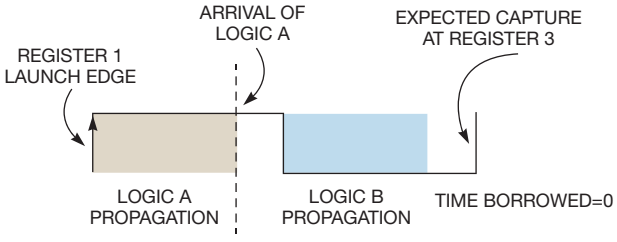


Figure 3 When Logic A is fast enough, no borrowing is necessary.

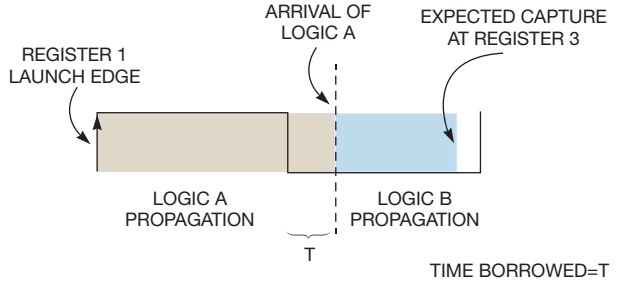


Figure 4 Logic A borrows time from Logic B.

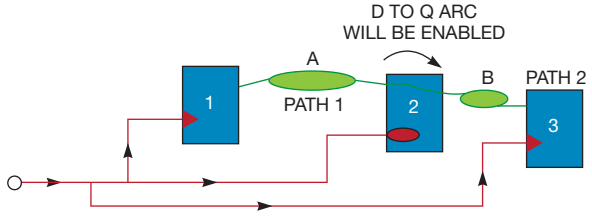


Figure 5 When the latch is enabled, it essentially becomes a passive delay.

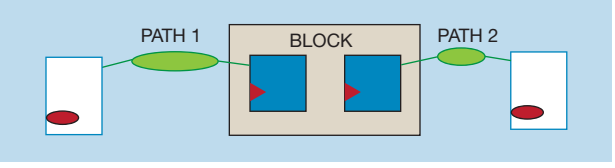


Figure 6 A block interfaces to external latches.

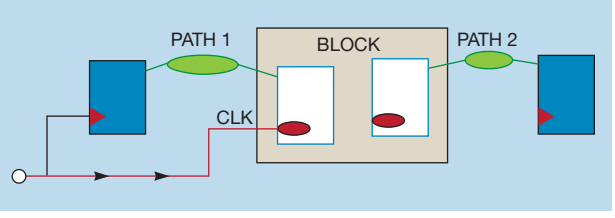


Figure 7 A block using latches interfaces to external registers.

The tool also misses all the paths that do not require time borrowing and holds a time check at the latch's endpoint.

## PARTITIONING CHALLENGES

Some challenges occur in hierarchical design when the blocks have latch-based interfaces. The timing tools require help to understand when it is possible to borrow time across a block boundary. The first challenge is to enable time-borrowing for the ports that you have budgeted for timing. When timing a block, you can model the ports that are entering or exiting latches at the top level of the SOC (system on chip) by using their proper I/O delays and the level-sensitive option in the EDA tool. Consider the case for Path 2 (**Figure 6**). Without the level-sensitive option, this path could be critical at the block level. By defining the output delay at the output port with the level-sensitive option, the timing tool can borrow time from the input stage of the next block, and this ability relaxes the timing on the output port.

Next, consider a case in which the latches are inside rather than outside the block (**Figure 7**). Path 1 has no special requirements for closing the block, but you must define all types of clock latency—rise, fall, minimum, and maximum times—for the CLK pin. This approach helps you correctly calculate the time of the starting point employing OCV and CPPR. In this way, you'll get no surprises when you merge the block at the top level. Another challenge arises when you use the timing models for top-level execution. You can enable time-borrowing through boundary latches by using gray-box ETMs (extracted timing models), which preserve the boundary latch and generate ETM libraries.

In summary, latches are beneficial for high-speed-SOC designs, but their use adds challenges in static-timing analysis, especially with hierarchical design. The limitations of EDA tools increase the complexities of latch-based design. You can employ latches in SOCs only after careful analysis. You can then apply some of these techniques, which can reduce the complexities of designing with latches. EDN


## ACKNOWLEDGMENT

*This article originally appeared on* EDN's *sister site, EDA Designline (http://bit.ly/p3aN3C).*


## AUTHORS' BIOGRAPHIES


*Ashish Goel is a lead design engineer at Freescale in India. He has 11 years of industry experience in static-timing analysis, RTL (register-transfer-level) design, physical design, and formal technologies. Previously, he worked at STMicroelectronics, Agilent Technologies, and Infineon Technologies. Goel holds multiple patents in FPGA architecture.*

*Ateet Mishra is a senior design engineer at Freescale in India, where he has worked for six years. He has experience in static-timing analysis, physical design, and synthesis. He has successfully taped out multiple SOCs.*