A New Approach to Easily Resolve the Hidden Timing Dangers of False Path Constraints on Clock Domain Crossings

Omri Dassa, Yossi (Joseph) Mirsky
Intel Corporation
Jerusalem, Israel

***Abstract*- The need for preventing data skew in digital designs is well known in the industry, and standard simulation or formal checks ensure that these functional bugs don't reach silicon. However, by labelling Clock Domain Crossing (CDC) signals as "false paths," different bits of a CDC bus or its qualifier may be routed with varying amounts of time delay. If the delay on certain bits is greater than 1 clock cycle, this can result in data skew or metastability in the sampling. Such issues can create silicon-killing functional bugs, and since these timing issues don't appear in RTL simulation, designers need to spend a large amount of effort to identify and constrain these paths for the back-end. This paper will describe new and improved methodologies for detecting and preventing such timing-analysis CDC bugs that also remove the overhead from designers.**

I. Introduction

To prevent metastability in silicon, huge resources are invested by the back-end teams to meet setup and hold times between sequential elements, otherwise known as "closing timing" [1]. However, certain sources of metastability are the front-end RTL designers' responsibility and include areas such as Clock Domain Crossings (CDCs) and Reset Domain Crossings (RDCs) [2]. This paper will describe an area of intersection between front-end CDC checks and back-end timing closure where glaring verification gaps have formed in many design teams' flows. Even properly synchronized (and random-delay simulated) CDC paths can still encounter functional bugs and metastability due to timing issues that arise from the blanket use of dangerous constraints such as *false paths*. This is a particular challenge in modern, large SoC designs with high-speed clocks. Signals across the chip from one IP or block to another necessitate huge delays of multiple clock cycles. There are several techniques for identifying problematic *false paths* on CDC paths and generating constraints for the back-end, but they are labor intensive and error prone [3]. In this paper we will present several examples of such metastability issues that can arise, briefly summarize the current solutions, and finally, introduce a new, silicon-proven methodology that greatly reduces the front-end effort and involvement in safely implementing these paths in the silicon.

II. Problem Statement

Standard timing analysis ensures that every signal propagates to the sampling sequential element within 1 clock cycle of the receive clock, and this ensures that data skew between bits will not occur. However, it is standard practice to label CDC paths as *false paths*, since closing setup and hold-time violations between asynchronous clocks are not possible. Instead, engineers rely on a synchronizer implementation to prevent any metastability from propagating into the receive domain. However, *false paths* or *multicycle* constraints on signals such as CDCs open the possibility for data skew to appear. The result can be similar to the known danger of separately synchronizing each bit of a functionally coherent bus that crosses clock domains (as shown in Fig. 1 below). In such cases, a change in a bus's value may propagate to the synchronizers just before the next rising edge of the receive clock. Slightly different datapath delays on each bit of the bus in the silicon can result in some of the data changes being sampled, while others are missed until the next clock cycle. This can induce temporary data skew and faulty data on the bus in the receive domain and create potential silicon-killing bugs.

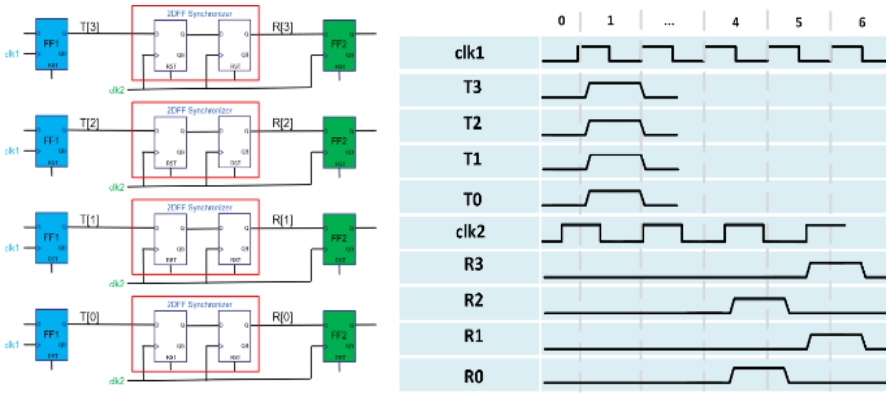

Figure 1. Bus crossing clock domain crossing, with each bit being sampled by a different 2DFF CDC synchronizer (left). Data skew can appear in the receive domain such as in clock cycle 4 (right).

To prevent data skew on CDC crossings, more complex schemes than synchronizing every bit are employed, such as Gray code counters or bus qualifiers. However, less known is that these implementations carry their own unique risks and dangers. If the place & route tools are given a free hand, in large, challenging, highspeed, or congested designs, the CDCs' signals may be implemented with substantial path delays—greater than several clock cycles of the receive domain. This may have serious consequences as, for example, bits within a certain CDC bus may be implemented with different time delays and arrive in the receive domain in different clock cycles relative to each other. We will demonstrate several examples of how this can occur.

A. Gray Code

Gray code counters are specially designed to ensure that only 1 bit changes in each transition. This is useful for many digital design applications, such as asynchronous FIFOs, as they are an elegant solution to quickly synchronize memory pointers across clock domains while avoiding the risks (illustrated in Fig. 1 above) of skewing data across different clock cycles [4]. Fig. 2 below illustrates a standard Gray code counter implementation for an asynchronous FIFO. The counter's current value in the transmit clock domain is converted to Gray code and sampled by a FF. Each bit is then separately transmitted to the receive domain through a 2DFF synchronizer, where it is then translated from Gray code back to binary. Since only 1 bit can change in every clock cycle of the transmit clock, in effect, no more than 1 bit per clock cycle is synchronized through the 2DFF synchronizer to the receive domain, and there is no danger of skew between that changing bit and the rest of the bus. This is true as it is generally assumed that each bit of the Gray code bus will have a datapath delay shorter than 1 clock cycle of the fastest clock. However, if there are no timing constraints on the Gray code bus that crosses clock domains, 1 or more bits may be delayed greater than 1 clock cycle of the fastest clock. This can result in breaking the fundamental assumption of only 1 bit changing per clock cycle: several transitions with different datapath delays may arrive in the same clock cycle, resulting in an unintended, incorrect counter value. This is an illegal state which will not have been seen in simulation and may cause serious functional bugs.

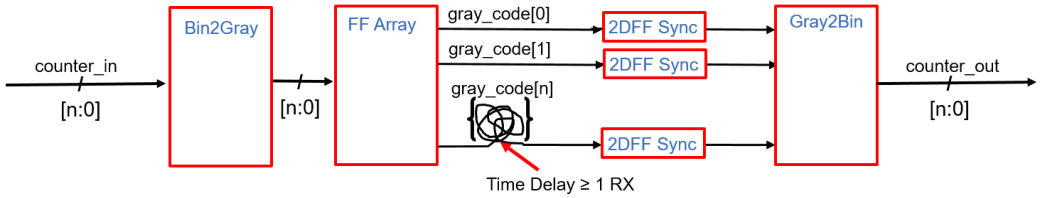

Figure 2. Async FIFO Gray Code Synchronizer with bit[n] in silicon implemented with a greater time delay compared to the rest of the bus.

B. DMUX Synchronizers

Fig. 3 below illustrates a standard DMUX CDC synchronizer scheme that uses a synchronized qualifier signal with an edge detector to safely sample CDC busses. We can outline three basic rules that must be met in order for the synchronization scheme to function correctly:

1) The data on the bus must always be stable before the synchronized edge-triggered qualifier signal goes high in the receive domain.
2) The data on the bus must remain stable after the qualifier has gone high, long enough to meet the hold-time requirements in the receive domain.
3) Due to the uncertainty of how many receive clock cycles it will take for the qualifier to go through the synchronizer, there is a further requirement that the data on the CDC bus must be stable for at least three receive domain clock cycles.

These rules can be verified in robust simulations, with the correct clock frequencies and random time delays added to the synchronizers.

However, by labeling the CDC bus and its qualifier as a *false path*, three distinct classes of silicon/timing bugs may occur:

1) If the timing delay of the entire bus crossing the clock domain is greater than 1 receive domain clock cycle, then the qualifier signal might arrive too early—i.e., before the data is stable and safe to sample—creating a setup violation.
2) Furthermore, as illustrated in Fig. 3 below, if 1 bit of the CDC bus has a time delay of 1 receive domain clock cycle—greater than the rest of the bus—then the delayed bit may not reach the receive domain's FF2 in time to be captured in the 3-clock window before the pulse from the qualifier arrives.
3) Finally, as illustrated in Fig. 4 below, if the timing delay of the qualifier prior to the synchronizer is greater than 1 receive domain clock cycle, then as a result of the delayed qualifier, the data being sampled in the receive domain might change too early while the receive domain is in the middle of sampling it, resulting in a hold violation.

All three scenarios may cause false or metastable logic values to be sampled and serious functional bugs in the silicon.

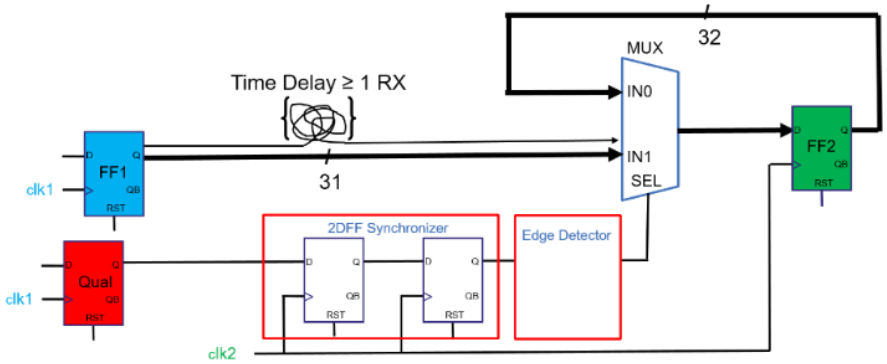

Figure 3. Illustration of 1 clock cycle timing delay (relative to the rest of the bus) on 1 bit of CDC bus resulting in skew or metastability in the receive domain.

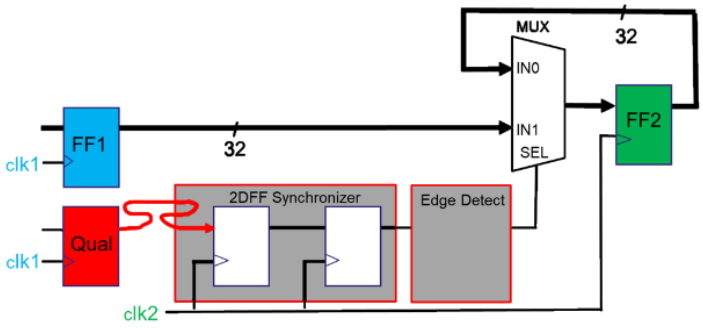

Figure 4. Illustration (in red) of timing delay on the qualifier relative to the bus and the simulated scenario.

III. Previous Solutions

A. Gate Level Simulation

From the front-end perspective, the examples given in Fig. 2, 3, and 4 behave correctly in simulation. The silicon bugs are introduced in the back-end stage due to blanket use of *false path* or *multicycle* constraints provided by the front-end, without the possible timing impacts being simulated. Performing Gate Level Simulation (GLS) with a good level of random time delay generation and high coverage should expose any functional issues on the CDC paths due to datapath delays. However, GLS is a costly, time-intensive flow and most engineering teams in the industry use other tools/flows such as Logic Equivalence Checking, CDC front-end checks, and X-propagation instead.

B. Group Data Skew

Another approach described in detail in reference [3] required intensive analysis from the front-end team to identify all related groups of CDC signals. CDC tools are used to identify all the relevant signals in a CDC group (for example, a DMUX bus and its qualifier or a Gray code bus); the front-end RTL names of the group are mapped to the post-place-and-route netlist (a very labour-intensive task!); and only then can the relevant back-end tool commands finally be generated and the datapath delay of the associated group of signals analyzed. If there is a gap of 1 or more receive clock cycles between any bits in the group, buffers are added to push/delay all the rest of the signals to arrive at the destination in the same clock cycle to ensure that the silicon behaves as was simulated. This approach requires a lot of manual effort, and the whole process must be done from scratch for each new RTL/release and synthesis cycle (due to new CDC paths and name changes, multi-flop implementation, etc. in the netlist). This approach is also risky, as certain kinds of CDC groups can be missed by CDC tools (a lot depends on the setup of the functional mode of the CDC tool). Until now, the solutions to what is essentially a back-end timing closure problem for large high-speed designs rested on front-end designers. We will now present a novel approach which enables the entire issue to be resolved by the back-end.

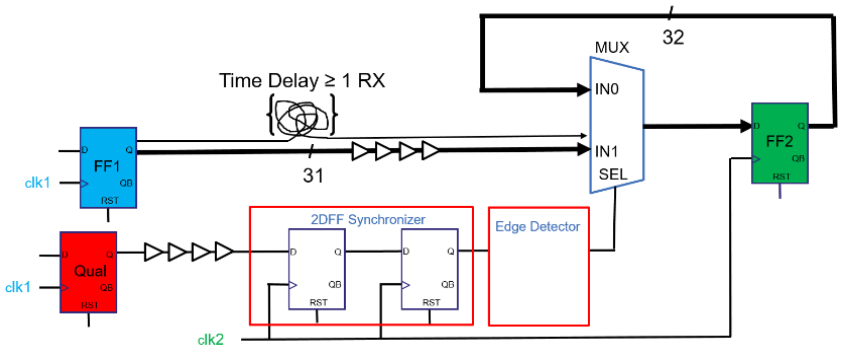

Figure 5. Buffers used to delay all other signals in the group to match timing with a bit that was delayed by more than a clock cycle.

IV. Solution

From the front-end perspective, the examples given in Fig. 2 and 3 behave correctly in simulation. The silicon bugs are introduced in the back-end stage due to blanket use of *false path* or *multicycle* constraints provided by the front-end. Therefore, a new collaborative approach is required to identify and prevent any timing/data skew. Previous solutions to this problem required intensive effort by the front-end team to identify all CDC paths in the design, laboriously group them together, and then painstakingly map all the related RTL signals in the group to the equivalent net names in the netlist. At the full chip level, add in multiple IPs/providers, synthesis optimizations, and multi-bit flops and this becomes a very time-intensive, error-prone task.

The new approach presented in this paper greatly simplifies and reduces the overhead from the process. Please note that due to Intel policy, we are prohibited from explicitly mentioning the vendor and tool commands which we used in our flows, so the solutions will be presented in pseudo-code. The same approach can be generically applied to any back-end vendor/tool flow.

Large chips/designs are divided at the RTL level into multiple partitions, and each partition is synthesized separately and integrated at the top. Our goal is to provide constraints per clock crossing path to the back-end team that will maintain the skew requirements stated in the problem statement. In general, timing issues are minimal at the partition

level; by adding the following constraints for each possible pair of clock crossings in the partition, we can ensure that no data skew or metastability issues will occur:

Partition Constraint: $For\ all\ paths\ between\ clk\_a\ and\ clk\_b, min\_delay = 0, max\_delay = \frac{1}{\max(clk\_a \mid clk\_b)}$

The *max_delay* must be the smallest clock period between the transmit and receive clocks. This ensures that only 1 transition per receive clock cycle is sampled in the Gray code counters (such as in Fig. 6 below). Regarding the rest of the paths between *clk_a* and *clk_b* in the design, it is an overconstraint to set $max\_delay = \frac{1}{\max(clk\_a \mid clk\_b)}$, as some paths only need to be constrained according to the transmit clock and others according to the receive clock. However, it is much easier to apply one global constraint to the entire partition instead of different constraints to different/specific CDC paths. Even though some CDC paths in the design may be legitimate proper *false paths*, we still prefer to broadly apply the strict constraint in order to easily catch and constrain the Gray code pointers. Any paths not meeting the strict timing constraint can be reviewed; if they are approved by the front-end as a true *false path*, then the timing violation can be waived—or, if identified as a real issue, the timing path can then be fixed. In our experience, using the stricter constraint did not cause any noteworthy timing closure issues or significant overhead.

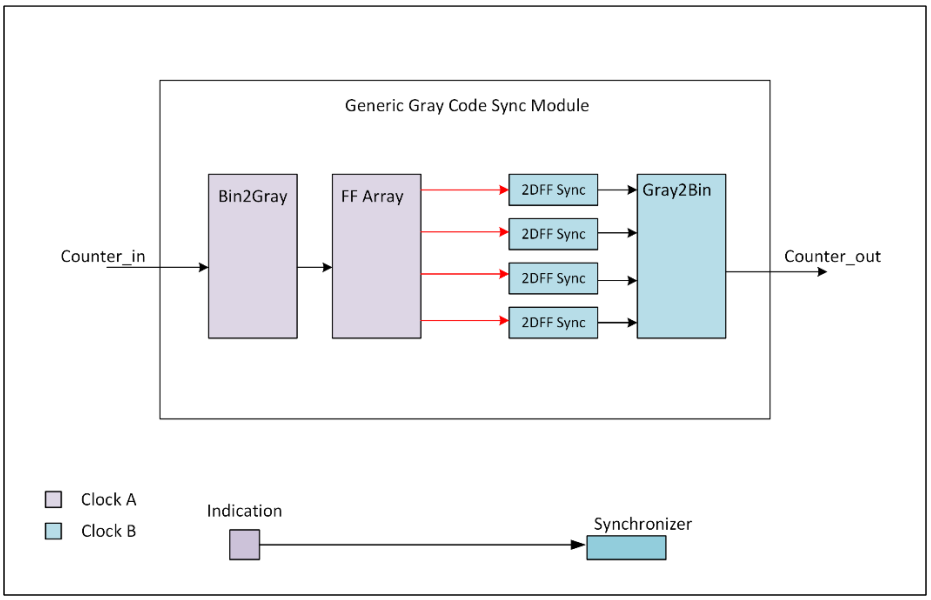

Figure 6. An example of different CDC schemes inside a partition which will have the strict *max_delay* timing constraint applied to them.

At the full chip or top level, the partition approach described above is generally not applicable. This is because many async paths routed across the die will indeed have a *max_delay* that cannot be reduced to be lower than a single clock cycle. Instead, we must map all the async paths between each partition and group them according to the transmit and receive clock domains. For example, as seen in Fig. 7, all async paths transmitting from *partition_x* in *clk_a* to *partition_y* using *clk_b* will be in the same group, regardless of the individual skew requirements of each path. This group will include different CDC signals and schemes, such as different data busses and qualifiers—each with their own skew requirements—yet we group them together. All the async paths from *partition_x* in *clk_b* to *partition_z* at *clk_c* will be grouped together in a second group. This is done even though there are signals in this group that go directly to a synchronizer and are not qualifiers. Creating/identifying such groups in back-end tools can be performed using very simple and quick commands.

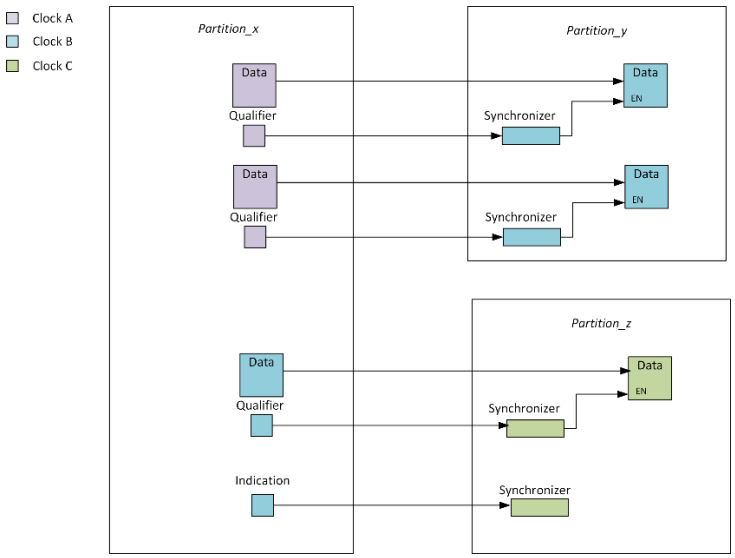

Figure 7. Typical Clock Domain Crossings at the SoC level and their groupings.

Once the groupings are formed, the back-end team can easily produce a report of all the signal delays for each asynchronous path in the group. This is reported equally for all bits, regardless of whether some signals are part of the same CDC bus or are, alternatively, a CDC qualifier or a real *false path*. Using our proposed method, the important information is only the groups of each partition/clock crossing pair, and we do not need to know the actual functionality or relationship that each signal bit provides. This greatly simplifies the analysis. The signal delay on each signal in the group can range from being small to multiple clock cycles in length. Since these are unconstrained paths, each path will have a different datapath delay that may be very spread out and uncorrelated, perhaps far longer that a timing window defined by *max delay = max(clk_a or clk_b)*.

The solution is therefore to take each group of signals and, using a simple Python script, look for a timing "window" the size of 1 cycle of the receive clock that maximizes the number of paths inside that window. From this window the script automatically creates constraints for the SD team to perform timing ECOs, adding buffers and delays in an attempt to move all the paths in every group to be inside their relevant *delay_window*. The assumption behind this grouping logic is that paths from the same group travel similar distances within the die; therefore, the data delay of those paths should be similar, meaning they can be reasonably expected to arrive within a single RX clock window. An example first iteration of one such timing analysis is presented in Table 1 and Fig. 8. Since the *End_Point_Clk* (or receive clock) frequency is known, the timing window can be easily identified and the best fit calculated by the script. Once the ideal timing window is chosen, all the paths that are already in this window still need to be constrained in order to ensure they remain inside it. The paths outside the window requirement will also be constrained to allow the timing ECO to move them into the window as well. After one or more iterations (see Fig. 9), all paths should be inside the timing windows as shown in Fig. 10.

TABLE I
DATAPATH DELAYS FOR ASYNCHRONOUS CROSS PARTITION PATHS

Start_Point_Partition	*End_Point_Partition*	*Start_Point_Clk*	*End_Point_Clk*	Datapath Delay (ns)
Par_A	*Par_C*	*Clk_D*	*Clk_B*	0.625305
Par_A	*Par_C*	*Clk_D*	*Clk_B*	0.701674
Par_A	*Par_C*	*Clk_D*	*Clk_B*	0.797138
Par_A	*Par_C*	*Clk_D*	*Clk_B*	0.848277

Each line of this table represents the timing delay of a single async bit that travels between different partitions.

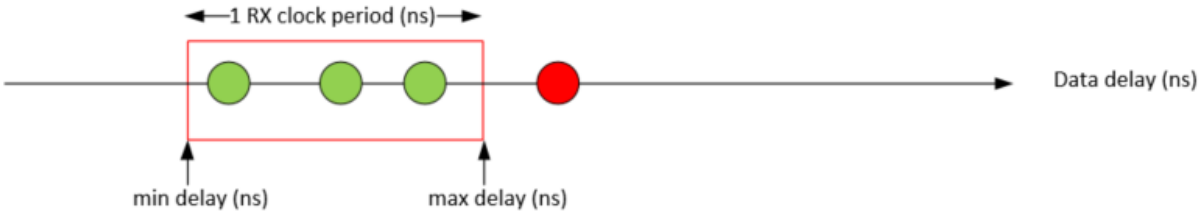

Figure 8. In the first iteration, the best-fitting timing window is 1 clock cycle of the receive clock. Timing paths in the window in green meet the initial timing window; those outside the window are labeled in red. Constraints to place all the timing paths in the window are generated and given to the back-end team.

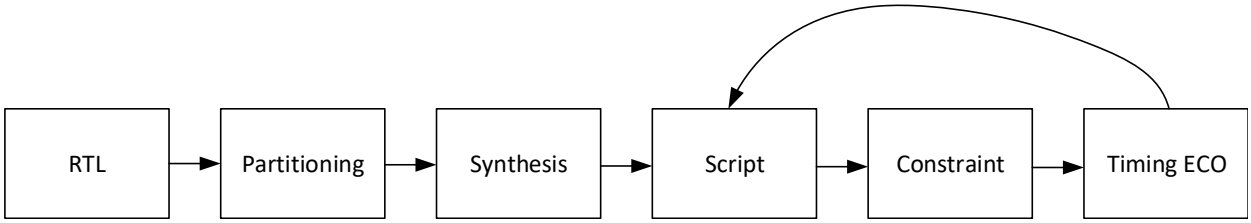

Figure 9. Flow for detecting and preventing data skew in CDC signals.

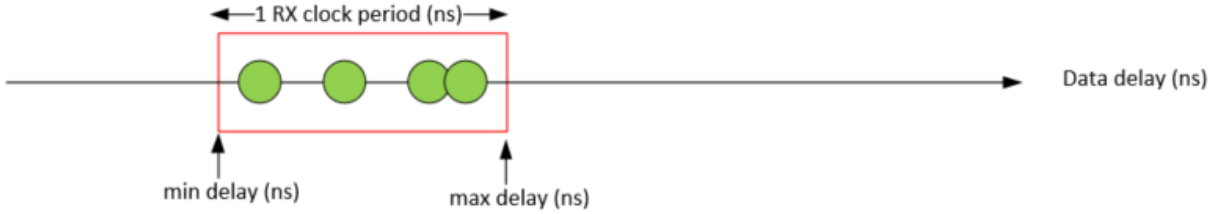

Figure 10. After several (N) iterations of the Python script and timing ECOs, all paths will be inside the window.

Groups of async paths from each partition/clk pair are not further subdivided into individual CDC busses and their related qualifiers in order to avoid the tremendous effort that was required to implement the solution described in reference [3]. This comes at the expense of several iterations of ECO timing cycles, but in our experience those ECO timing cycles would have been performed in any case to address other timing issues, and the extra *delay_window* constraints does not have any significant impact on back-end effort to close timing. By using this method, we can reduce the front-end effort by shifting it into a script that can be run by the back-end and reviewed by the front-end to address some corner cases such as true *false path* signals on very slow clocks. The additional effort from the back-end team was also found to be negligible.

V. Real World Results

This new approach has been used successfully to tapeout large complex chips/designs. Multiple real issues, which almost certainly would have caused serious silicon bugs, were detected and resolved in the design prior to tapeout. The approach was robust and certain to catch any potential real issues due to the overconstraint approach, yet both the front-end and back-end teams' effort was found to be negligible. All that was required was creating reports from the back-end tool of the timing groups and then running the timing window script to automatically generate constraints. The back-end team had no major issues performing the limited number of timing ECOs required. The entire flow is much quicker, simpler, less risky, and requires less overhead than previous tailored-per-CDC signal solutions that have been used in the past.

VI. Acknowledgments

Thank you to Shoshana Mirsky for editing this paper.

REFERENCES

[1] A. L. J. M. I. H. J. Kahng, VLSI Physical Design: From Graph Partitioning to Timing Closure, Springer Science, 2011.

[2] Y. Mirsky, "Comprehensive and Automated Static Tool Based Strategies for the Detection and Resolution of Reset Domain Crossings," in *DVCon USA*, San Jose, 2017.

[3] Y. Mirsky, O. Tsarfaty, D. Stein, & O. Winner, “Timing Analysis of Unconstrained Clock Domain Crossings – the Need and the Method,” SNUG Israel, 2018.

[4] C. E. Cummings, "Synthesis and Scripting Techniques for Designing Multi Asynchronous Clock Designs," in SNUG, San Jose, 2001.