WHITE PAPER

Clock Domain Crossings in the FPGA World

Revision 1.0

2018-02-21

Alexander Gnusin

Sergei Zaychenko, PhD

[Level: Advanced]

Abstract

Clock domain crossing (CDC) issues cause significant amount of failures in ASIC and FPGA devices. As FPGA complexity and performance grows, the influence of CDC issues on design functionality grows even more. This paper outlines CDC issues and their solutions for FPGA designs. Various design techniques are presented together with real-life examples for Xilinx and Intel FPGA devices. More importantly, this paper summarizes the most important CDC guidelines for highly-reliable FPGA designs.

Table of Contents

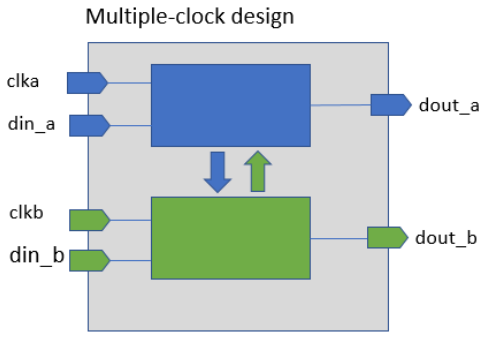
Introduction

As the entire electronics industry continues to add more peripherals to FPGA-based systems to address various consumer and business requirements, the metastability effects within the FPGA devices become increasingly notorious. The metastability effects can result to non-deterministic and non-functional errors that are impossible to catch with industry RTL simulators and static timing analysis tools. The resulting errors are typically found in FPGA test beds or in the field which means correcting them is quite expensive. If metastability effects can be properly mitigated earlier in the project lifecycle, then it can minimize verification costs and increase reliability of FPGA-based systems.

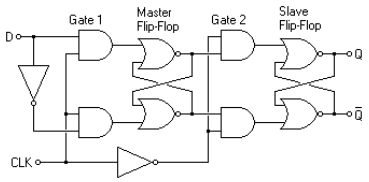
The following white paper explains metastability and clock domain crossing issues in hardware designs, outlining various design practices to make designs immune to metastability effects and functional non-determinism introduced by clock domain crossings.

The Metastability Effect

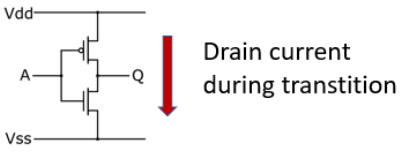
Metastability is a phenomenon that can cause system failure in digital devices, including FPGAs, when a signal is transferred between circuitry of unrelated or asynchronous clock domains. Therefore, this means it happens in systems with two or more asynchronous clock sources. Any sequential element in a design requires a certain period of stability for data to be properly captured; due to the unrelated clocks, this condition cannot be guaranteed in designs with multiple asynchronous clocks.

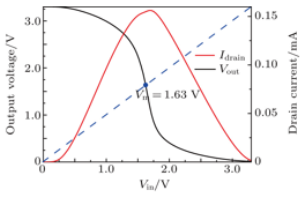


To better understand the metastability effect, there is a need to examine logic gates, which are the building blocks of sequential elements such as D flip-flops:



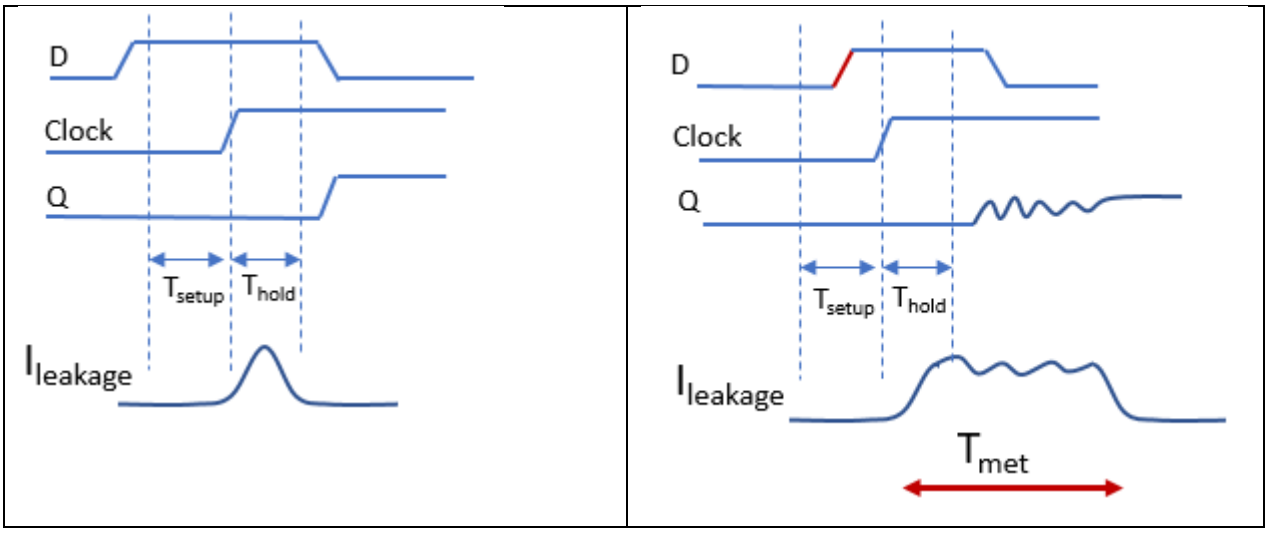
In fact, every flip-flop represents a negative feedback system with two stable states, representing two binary values. Each combinational gate contains one or many parallel or sequential N and P CMOS transistors in open or closed states. In a stable state, gates do not consume power other than through tiny leakages of current, as they must always be “closed” on one side of the power railing. However, when a value changes, there exists a certain amount of time when both N and P transistors are partially opened, leading to an increase in leakage current. ASIC and FPGA vendors do their best to minimize this amount of time as well as overall leakage power during the data change. The voltage level on gate A should change quickly enough to minimize leakage current. In the case that





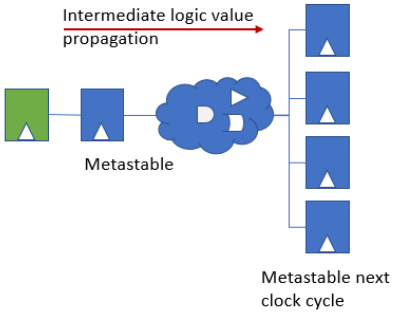
the gate voltage becomes stuck between logic values, both gates stay partially open, causing high current flow between power rails, device heating, and possible burnout.

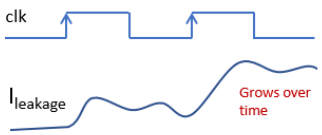
For proper operation, each sequential element has setup and hold timing requirements for data to be stable around an active edge transition (data capture). When these timing requirements are violated, the flip-flop, having negative feedback, starts bouncing at an intermediate (metastable) state. The bouncing time may be quite long, being comparable to, or even exceeding, the clock cycle period. Obviously, the leakage current and power dissipation grows significantly during this time. The metastability time (T_{met}) is a statistical parameter, and depends mostly on technological parameters of the current process.



The left picture presents valid timing on the flip-flop: the input data D is stable during pre-defined phases of the T_{setup} and T_{hold} periods around the active edge transition. The right picture presents the metastability effect because of an input data change close to the active edge transition. Because of the occurring metastability, the flip-flop output oscillates during T_{met}, producing a non-deterministic binary value, and experiences a high drain in current leakage.

As it is explained above, the metastability effect causes the occurrence of intermediate logic values at a flip-flop's output. When the sum of metastability time and propagation delay approaches the clock period, the intermediate logic value advances to later flip-flop stages, spreading the effect throughout the design. In the worst case, this may cause the "metastability avalanche effect". Also, combinational logic gates between the flip-flops consume much more power while propagating intermediate logic values.





If timing violations occur often, the influence of the metastability effect may be critical, causing non-determinism in functional operations and heating up certain sections to chip burnouts.

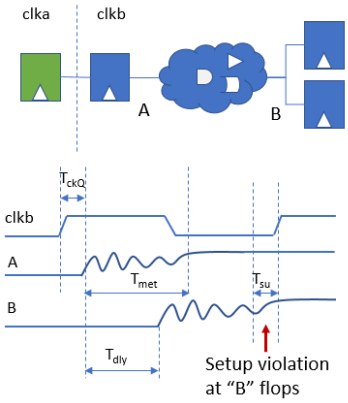
Let's take a closer look at metastability propagation conditions.

Assuming all other inputs to the combinational logic in the receiving clkb domain are stable, we may write the following condition for preventing metastability propagation between register stages:

$$T_{ck\text{-}Q} + T_{met} + T_{dly} < T_{period} - T_{su}$$

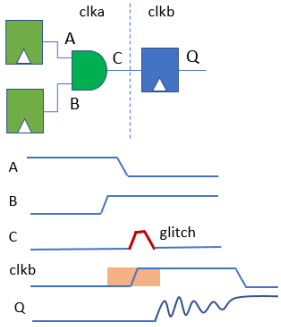
Here, the formula members are:

- $T_{ck\text{-}Q}$: clock-to-output flip-flop delay
- T_{met}: metastability time
- T_{dly}: propagation delay through wires and combinational gates
- T_{period}: clock period
- T_{su}: flip-flop setup time



In this formula, T_{dly} is the only parameter controllable by the designer. Designers are able to minimize T_{dly} by removing logic gates between flip-flops, as well as by reducing wire delay by placing A and B flip-flops close to each other.

Glitches and hazards have an impact on the metastability effect too. The following picture presents potentially glitchy logic in a sending clock domain. Because the receiver may capture the signal at any time, glitches may violate the input signal stability requirement during setup and hold time, causing metastability effects at the receiving flip-flop. Removing all combinational logic between driver and receiver flip-flops will eliminate any possibility of CDC-related glitches, and this is recommended practice for multiple-clock designs.



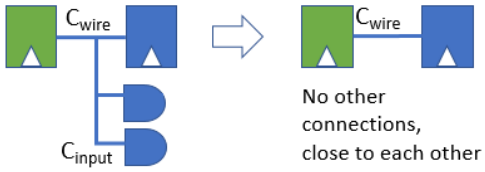
In addition to glitches, it is important to reduce the transition slew of the signal crossing clock domains. During signal transition, an intermediate value is sensed by the receiving flip-flop, and there is a higher chance for a metastable occurrence if a signal transition takes more time than the clock transition period.

The transition period is defined by the driver strength and node capacitance:

$$C_{node} = C_{wire} + \sum Ci$$

where:

- C_{wire}: wireload capacitance
- Ci: sum of input capacitances

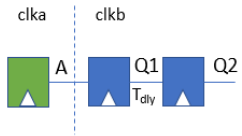


To reduce the transition slew effect on metastability, it is recommended to reduce cross-domain signal capacitance and wireload. The best design practice is to keep one-to-one connections between driver and receiver flip-flops, without any intermediate combinational logic or other connections (keeping fanout to be only one).

Two Flip-Flop Synchronizer as a Common CDC Solution

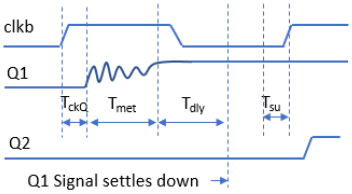
The main responsibility of a synchronizer is to allow sufficient time such that any metastable output can settle down to a stable value in the destination clock domain. The most common synchronizer used by designers is the two-flip-flop (2-FF) synchronizer. Usually the control signals of a design are synchronized by 2-FF synchronizers.

Let's take a look at the timing path between the Q1 and Q2 flip-flops of this synchronizer pair.



Assuming:

$$T_{ckQ} + T_{met} + T_{dly} < T_{ck\_period} - T_{su}$$



The signal at the Q2 flip-flop input settles down prior to being captured by the rising edge of clkb. In this case, the Q2 flip-flop never experiences metastability issue and always captures proper binary logic values, passing them to other logic elements in the design. To ensure the absence of the metastability effect on the second flip-flop (Q2) of the synchronization pair, there is a need to make sure that each of the synchronizer flip-flops share a direct connection only (no other connections allowed) and are placed closer to each other.

In the case that metastability time T_{met} is not small enough to fit into the above equation, there is a need to build the synchronizer with three of even more flip-flop stages.

In summary, the recommended synchronizer pair design requirements are:

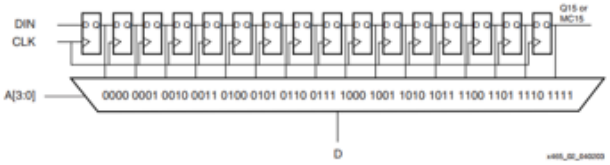
No	Requirement	Reason
1	Do not place any combinational logic between sending (A) and receiving (Q1) flip-flops	Reduce transition time, avoid glitches
2	Do not connect the sending flip-flop (A) output to any other gates	Reduce transition time
3	Keep the sending flip-flop (A) closer to the receiving one (Q1)	Reduce transition time
4	Do not place any combinational logic between synchronizer flip-flops (Q1, Q2, …)	Reduce probability of metastability effect by reducing propagation delay between synchronizer flip-flops
5	Do not connect synchronizer flip-flops to any other logic gates (except from the last one)	Reduce probability of metastability effect by reducing propagation delay between synchronizer flip-flops
6	Keep synchronizer flip-flops closer to each other	Reduce probability of metastability effect by reducing propagation delay between synchronizer flip-flops

Safe Synchronizer Implementation in FPGA

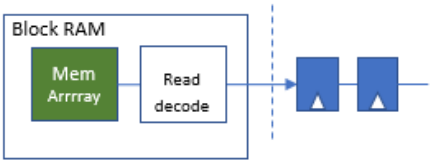
Just like ASICs, FPGA devices suffer from the metastability effect and its consequences. The following guidelines must be considered when implementing cross-clock domain transitions in FPGA:

1. NDFF (2 or more flip-flop stages) synchronizers may be built from non-resettable flip-flops. Here, we treat these flip-flops as "pipeline" registers, whether they relate to control logic or not.
2. NDFF synchronizers, as well as sending flip-flops, should be preserved (not optimized) by synthesis tools. In some cases, synthesis tools perform inter-register optimization, moving some combination logic between neighboring register stages.
3. NDFF synchronizers (as well as sending flip-flops) should be implemented from flip-flop FPGA resources only. No shift registers or BRAMs are allowed, as they do not provide direct-only flip-flop to flip-flop connections:

SRL shematics, flip-flops have extra Connections to output decoder:



MUX-based BRAM Read Decode Logic may generate glitches:

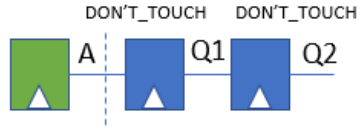


4. By placing NDFF synchronizer flip-flops in the same slice, we achieve the smallest inter-flop propagation delay, reducing the chance of metastability effects.

Xilinx uses the ASYNC_REG attribute placed on all NDFF flip-flops. This attribute:

- Automatically adds the "DONT_TOUCH" attribute on synchronizer flip-flops, preventing them from synthesis optimization and from mappings to SRL blocks.
- Instructs placement tools to keep synchronizer flip-flops close together, preferably in one slice. However, there is no guarantee that all synchronizers will be implemented in slices.
- Instructs the gate-level Xilinx ISIM simulator to avoid producing X's in synchronizer flip-flops when setup/hold violations happen. Please note that other non-Xilinx simulators miss this feature, requiring special gate-level simulation constraints to avoid X generation in the synchronizer's flip-flops.

In addition to constraint timing between NDFF synchronizer flip-flops, there is a need to constrain the timing between the driver flip-flop and NDFF synchronizer. The "*set_false_path -to Q1*" command excludes all cross-clock domain timing paths from synthesis



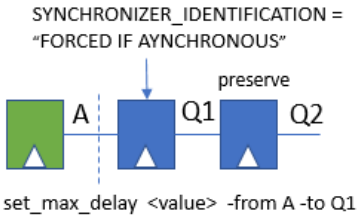
set_max_delay -datapath_only <value>
-from A -to Q1

optimization and timing analysis. However, long timing paths to the synchronizer input impacts transition time as well, causing metastability effects to happen more often. It is preferable to reduce these timing paths using the "*set_max_delay -datapath_only*" command. The "*-datapath_only"* option removes the clock skew from slack computation, as clock skew is not applicable for constraining signals between asynchronous clock domains.

The following code example shows the Xilinx-specific NDFF synchronizer implementation for a 1-bit control signal:

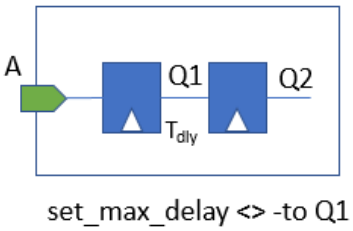
```
(*ASYNC_REG = "TRUE") reg [1:0] control_sync_ff;
always @(posedge clk)
  control_sync_ff <= {control_sync_ff[0], control};
....

# Timing Constraints
set_max_delay -datapath_ony -from control -to control_sync_ff[0]
```



Intel FPGA uses the Intel-specific "SYNCHRONIZER_IDENTIFICATION" attribute with the "FORCED IF AYNCHRONOUS" value. This attribute should be placed on the first NDFF flip-flop only. The attribute prevents synthesis optimization of first NDFF flip-flop as well as provides metastability optimization during placement, trying to place the NDFF flip-flops closer to each other. However, there is still a need to set the "preserve" attribute on the second NDFF flip-flop. As was the case with Xilinx, constraining timing paths to the NDFF input is a recommended practice.

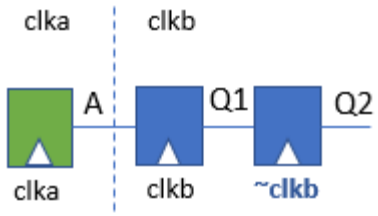
Please note that Intel FPGA does not provide any means to eliminate X's generated in NDFF synchronizers during gate-level simulation.



Some designs contain cross domain clocking paths from input design ports. It is not recommended for CDC paths to cross design boundaries. The only exception could be the designs with NDFF synchronizers organized in separate entities/modules. In the case that a NDFF synchronizer connects directly to the input port, there is a need to constrain the timing path from the input port to the NDFF using the "set_max_delay" command. However, the external-to-module part of the timing paths may not be controllable, causing potential CDC issues. For IP developers, it is highly recommended to hide all CDC transitions within IP designs, preventing any CDC issues during IP integration in customer designs.

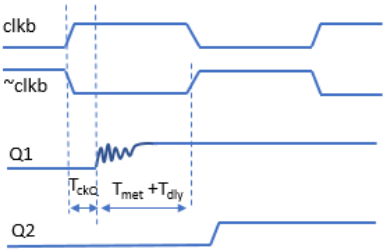
Half-Cycle Synchronizers in FPGA

In half-cycle synchronizers, the second flip-flop captures data using an inverted clock edge:



The purpose of inverting the clock at the second synchronizing flip-flop is to save a half clock cycle delay during the propagation delay of CDC signals. Sometimes, these synchronizers are implemented on timing-critical paths of ASIC designs, where propagation delay savings within CDC paths are critical for correct design functionality. This approach, however, adds significant constraints on metastability time: now, metastability time plus added propagation delay must fit into the half period of the receiving clock:

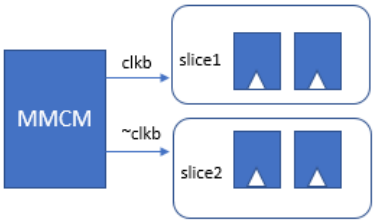
In addition, the clock inversion introduces extra clock uncertainty, constraining the metastability time even more.



Half-cycle synchronizer should be avoided in FPGA designs because of the following reasons:

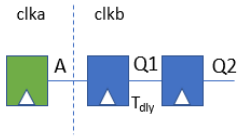
- In FPGA, clock inversion cannot be done using a local inverter gate so close to the second flip-flop. Inversion must be done through centralized FPGA clocking resources, for example, through the MMCM clock management component in Xilinx. Because of this, extra clock tree resources are required for the inverted clock. The mismatches between “direct” and inverted clock tree delays are more than the delay though the local clock inverter. So, the increasing clock uncertainty constrains the length of metastability time even more.
- In FPGA, registers belonging to different clock domains reside in different FPGA slices:

  Because of this, the interconnect path between synchronizer registers are not optimal from a CDC perspective.

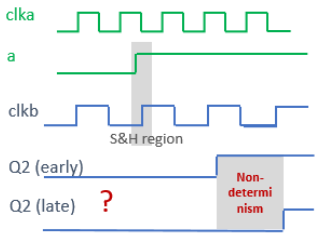


Functional Non-Determinism of CDC Signals

From a functional perspective, clock domain crossings introduce non-determinism into design functionality due to unpredictable NDFF delays for signals crossing clock domains:

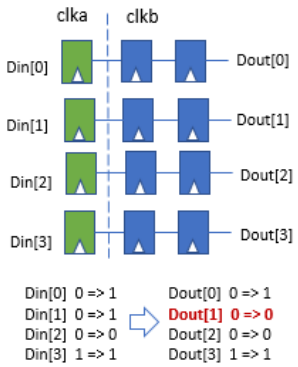


For a single control signal, this may not be an issue as 1-cycle delay unpredictability is not important for passing signals to the new clock domain.



The issue of unpredictable delay becomes critical when related signals (either control or data) cross clock domains through NDFF synchronizers. Therefore, it is dangerous to pass functionally related bit signals through NDFF synchronizer arrays because of potential data corruption at the receiving side.

The following example demonstrates vector corruption at a clock domain crossing through a NDFF synchronizer array:



Due to delay variations, the Din[1] bit arrives later than the rest of Din bits, causing Din vector corruption at the receiving side.

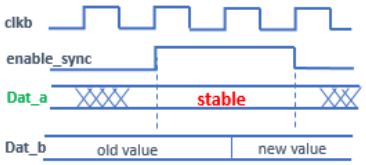
In the above example, both bits Din[0] and Din[1] change simultaneously, with the Din[1] bit experiencing more delay than bit D[0]. There should be no corruption if only one bit in the Din vector changes at a time. In this case, the receiving side vector either changes to a new value or, in the case of extra cycle delays, stays at the previous value. Gray coding ensures only a 1-bit change, if it is applied on incremental counters with a counter limit of power two.

Data Synchronizers

There are two basic methods for transferring data signals across clock domain boundaries. The first is based on enable-controlled data capture in the receiving domain. The second is based on sequential writing and reading of data using a dual-port FIFO.

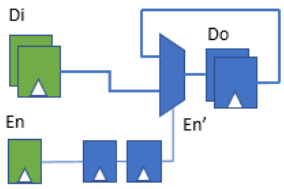
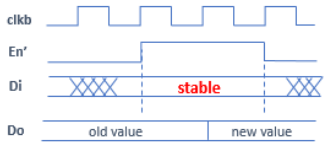
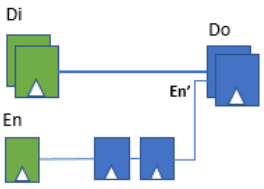
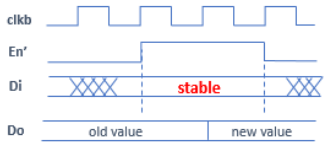
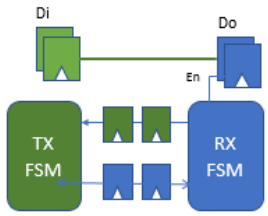
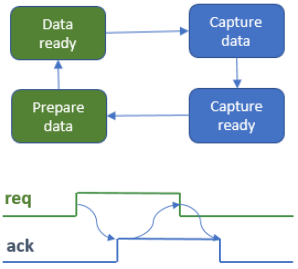
Control-Based Data Synchronizers

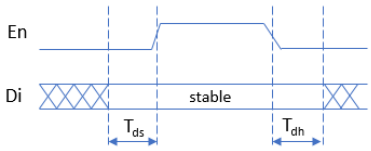
There are different types of synchronizers using enable-controlled data capture. In each type, however, the enable signal is responsible to inform the receiving domain that data is stable and ready to be captured:



The transmitter is responsible for keeping data stable over a time when the data enable signal, propagated to the receiving domain, stays asserted. The stability of all data bits during receiver data capture guarantees an absence of the metastability effect as well as correct data capture.

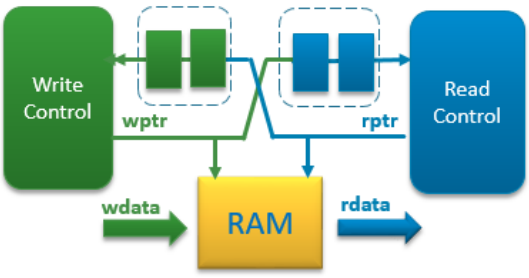
The following table summarizes the most commonly used control-based data synchronizer types:

Name	Schematics	Waveform / State diagram	Description
Mux-Based Data Synchronizer			Sender domain is responsible to keep Di data stable when captured by the receiving domain (when En' signal is asserted)
Enable-Based Data Synchronizer			Similar to the mux-based synchronizer, but based on flip-flops with built-in enable signals
Handshake-Based Data Synchronizer			The task of the TX/RX FSM pair is to ensure two-side communication between sender and receiver to guarantee Di signal stability while the En signal is asserted



To achieve safe data capture, the control-based data synchronizers should make sender data stable not only during the period of the enable signal assertion, but slightly wider, considering certain data setup and hold times relative to the data's stability region. This is important to eliminate possible data glitches during data capture. The handshake-based data synchronizer automatically implements these data stability margins during the cross-domain handshake communication.

FIFO-Based Data Synchronizers

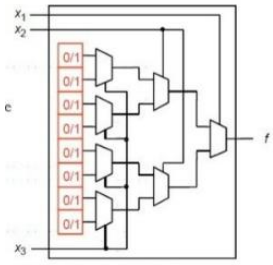


Limited bandwidth is one of the main drawbacks of control-based data synchronizers. Each data transfer takes time, since each data transfer comes along with control signal intercommunication. FIFO-based data synchronizers allow fast data communication through clock domain boundaries. In FIFO-based data synchronizers, data is pushed into the FIFO with the transmitter clock and pulled out from the FIFO with the receiver clock. The FIFO_FULL control signal controls the driver write frequency, while the FIFO_EMPTY signal controls the receiver read frequency.

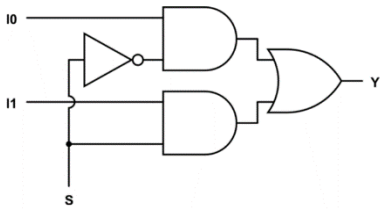
In asynchronous FIFO designs, the Gray-coded read and write pointers are passed into alternate clock domains through NDFF synchronizers to generate full and empty status flags. It is important to make sure that prior to coding, a counter runs from 0 to the power-of-two boundary, as only in this case can we achieve a one-bit-change of the coded counter signal during counter wrap-up.

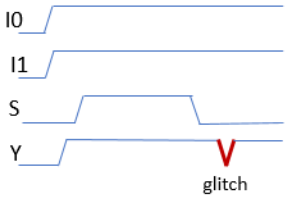
Data Synchronizers Implementation in FPGA

One of the more well-known FPGA issues is the use of multiplexors in the output decoders of lookup tables:



These multiplexors may produce glitches during certain combinations of input values. The basic multiplexor is built from AND and OR gates, with an inverted “sel” value supplied to the upper AND gate. When both multiplexor inputs stay “high” and that select signal changes from high to low, the glitch appears on the multiplexor’s output, as shown in the following picture:

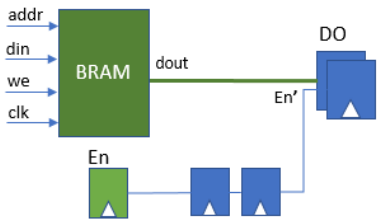




So, even though both multiplexer inputs remain stable and only the mux select signal changes, the multiplexor may experience glitches on its output. This glitch may be captured at the receiving clock domain, causing metastability and data value corruption. As multiplexors are used to implement a lookup table’s output decoder, any combinational logic in FPGA may produce glitches.

In timing-critical high-speed FPGA designs, it is important to avoid any combinational logic located at either control or data clock domain crossings. For this reason, mux-based data synchronizers should be avoided. The preferred data synchronizer in FPGA should be built on receiving flip-flops with built-in enable signals. Both Xilinx and Intel FPGA development tools support the “direct_enable” attribute set on these flip-flops. This attribute implements flip-flops with a built-in enable input. During the data synchronizer’s implementation in FPGA, it is important to mark the wire controlling the flip-flop capture array with the “direct_enable” attribute. Also, it is important to place data sending and data receiving flip-flop arrays close to each other in order to reduce cross-domain propagation delay as well as signal transition time. The “set_max_delay” constraint should be set on the data synchronizer’s timing paths.

For non-timing-critical FPGA designs, designers may use (with special care) combinational logic on clock domains crossings. For example, BRAMs may be used instead of driving a flip-flop array, being connected directly to receiving flip-flops from another clock domain:



To avoid glitches during data transfer, the outputs of BRAM should remain stable during En’ signal assertion, as well as adding sufficient setup and hold margins.

The following code examples illustrates data synchronizers’ implementations for Xilinx and Intel FPGA devices, along with timing constraints:

FPGA Vendor	Code	Constraints

Xilinx	`module data_sync_xilinx (input clk, enable, input [3:0] data_in,`
	`output reg [3:0] data_out);`
	
	`(* ASYNC_REG = "TRUE" *) reg en_meta_ff;`
	`(* ASYNC_REG = "TRUE" *) reg en_sync_ff;`
	
	`always @(posedge clk) begin`
	`en_meta_ff <= enable;`
	`en_sync_ff <= en_meta_ff;`
	`end`
	
	`(* direct_enable = "yes" *) wire data_enable;`
	`assign data_enable = en_sync_ff;`
	
	`always @(posedge clk)`
	`if (data_enable)`
	`data_out <= data_in;`
	
	`endmodule`
		set_max_delay
		
		-datapath_only <delay>
		
		-from …/data_in
		
		-to …/data_out
		
		set_max_delay
		
		-datapath_only <delay>
		
		-from …/enable
		
		-to …/en_meta_ff

Intel FPGA	`module data_sync_intel (input clk, enable, input [3:0] data_in,`
	`output reg [3:0] data_out);`
	
	`(* preserve altera_attribute = "-name SYNCHRONIZER_IDENTIFICATION ""FORCED IF ASYNCHRONOUS""" *)`
	`reg en_meta_ff;`
	`(* preserve *) reg en sync_ff;`
	
	`always @(posedge clk) begin`
	`en_meta_ff <= enable;`
	`en_sync_ff <= meta_ff;`
	`end`
	
	`(* direct_enable = 1 *) wire data_enable;`
	`assign data_enable = en_sync_ff;`
	
	`always @(posedge clk)`
	`if (data_enable)`
	`data_out <= data_in;`
	
	`endmodule`
		set_max_delay
		
		<delay>
		
		-from …|data_in
		
		-to …|data_out
		
		set_max_delay
		
		<delay>
		
		-from …|enable
		
		-to …|en_meta_ff

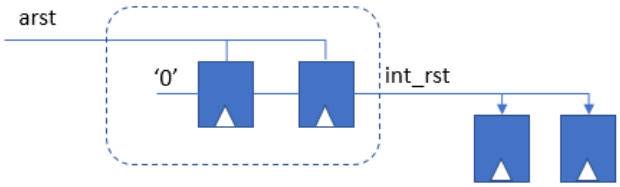
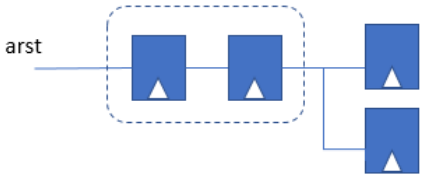
The safest way to implement FIFO-based data synchronizers in FPGA is using built-in FIFO generators, such as the LogiCORE IP FIFO Generator from Xilinx. The generated FIFO should be configured with independent clocks for read and write operations. For custom-built FIFOs, it is important to check that read and write pointers crossing clock domains are properly encoded, with only one bit changing at a time throughout device operation. Assertions should be used to validate one-bit-only changes of these pointers.

Reset Synchronization in FPGA

In both ASIC and FPGA devices, reset signals have to be synchronized at de-assertion to prevent registers from going metastable with corrupted values right after reset de-assertion. In the case of multiple-clock devices, each clock domain should be supplied with properly synchronized reset signals. Sometimes, the common asynchronous reset signal is supplied to multiple-clock designs. In such cases, this signal should be properly synchronized to each one of the design clocks. In other cases, reset signals could be designed as pre-defined configuration register bits. Same as an asynchronous reset, it must also be re-synchronized to other clock domains.

Reset synchronizers may synchronize either one or both reset signal edges (full or partial synchronization). Sequential elements with asynchronous resets may receive either fully or partially synchronized reset signals. Sequential elements with synchronous resets should receive fully synchronized reset signals only.

The following pictures illustrate partial and full reset synchronizers (in fact, the full reset synchronizer is the same as a basic NDFF synchronizer):

Partial reset synchronizer (reset de-assertion synchronization)	
Full reset synchronizer, both reset edges	

As in NDFF synchronizer implementation, reset synchronizers should be designed using “ASYNC_REG” (Xilinx) or “FORCED IF AYNCHRONOUS” (Intel FPGA) attributes, as well as constrained by “set_max_delay” timing constraints.

Cross-Domain Clocking Techniques for Highly-Reliable FPGA Devices

Highly reliable FPGA devices require special attention to clock domain crossings. The following design requirements are essential for reliance within multiple-clock devices:

- Instantiate special metastability hardened flip-flop macros on all cross-domain boundaries. For example, use the “Hard_Sync” metastability hardened macro in Xilinx. These macros could be configured with two or more register stages. The usage of these macros significantly reduces the impact of metastability by reducing the metastability time and frequency of metastable events.
- Constrain CDC timing paths with “set_max_delay” instead of “set_false_path”. In a case of special metastability hardened FF macro usage, set “set_max_delay” constraints on timing paths between driver flip-flop directly connected to the inputs of metastability hardened FF macros.
- Never use combinational logic on clock domain crossings or between synchronizer stages.
- Ensure that single-bit control signals do not reconverge in the receiving domain, even after one or more register stages.
- In enable-based data synchronizer designs, implement sending data flip-flop arrays as registers-only (not BRAMS etc).
- In enable-based data synchronizers, use “set_max_delay” to constrain timing paths between sending and receiving data registers.
- Ensure data synchronizers contain receiving flip-flop arrays with built-in “enable” signals. For this purpose, ensure that the “direct_enable” attribute is set on the receiver flip-flops’ enable nets.
- Prefer using handshake synchronizers for non-frequent data transfers. Handshake synchronizers ensure that data signals are stable long enough for proper capture within the receiving domain.
- For frequent data transfer, prefer using FIFO-based data synchronizers, generated with vendor-specific FIFO generation tools. For example, use the LogiCORE IP FIFO Generator to generate dual-port FIFO components for Xilinx devices.

Advanced linting tools are capable of ensuring safe operation of complex multiple-clock designs, as their capabilities go far beyond the checks implemented in FPGA vendor-specific design tools.

Summary

This article outlined cross domain clocking (CDC) issues and their solutions in FPGA designs. Various design techniques were presented, and illustrated with real-life examples for Xilinx and Intel FPGA devices. The CDC rules for highly-reliable FPGA designs were outlined as well. As advanced linting tools are capable of automatically verifying these rules, the usage of these CDC linting tools is critical to ensure safe operation of FPGA devices.

About Aldec, Inc.

Aldec Inc., an industry leader in Electronic Design Verification, provides a patented verification technology tool suite including: RTL Design, RTL Simulation, Hardware-Assisted Verification, SoC/ASIC Emulation & Prototyping, Design Rule Checking, CDC Verification, IP Cores, Requirements Lifecycle Management, DO-254 Functional Verification, Embedded Solutions, High-Performance Computing and Military/Aerospace solutions. *www.aldec.com*