Understanding Clock Domain Crossings

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

Neena Chandawale
(Host)

Ahishek Bora
(Presenter)

Agenda

- Introduction
  - Clock Domain
  - Clock Domain Crossing
  - Metastability
- CDC Synchronization Techniques
  - Synchronizers
  - Two Flip-Flop Synchronizer
  - Closed Loop Synchronizer
  - Handshake Synchronizer
  - Asynchronous FIFO Synchronization Technique
- CDC Support in IDesignSpec^{TM} Design
  - HW Interface CDC Techniques
  - SW Interface CDC Techniques
- Summary
- References

Introduction

- Clock Domain
- Clock Domain Crossing
- Metastability

Introduction

- All modern System-on-Chip (SoC) designs face increasing size and complexity challenges
- Speed and power requirements lead to designs with multiple asynchronous clock domains employed at different I/O interfaces and data being transferred from one clock domain to another
- In a Clock Domain Crossing (CDC) design one clock is either asynchronous to, or has a variable phase relation with respect to, another clock

Clock Domain

A "Clock Domain" refers to a portion of design that is processed by the same edge of the same clock

- Different edge of the same clock forms a different clock domain

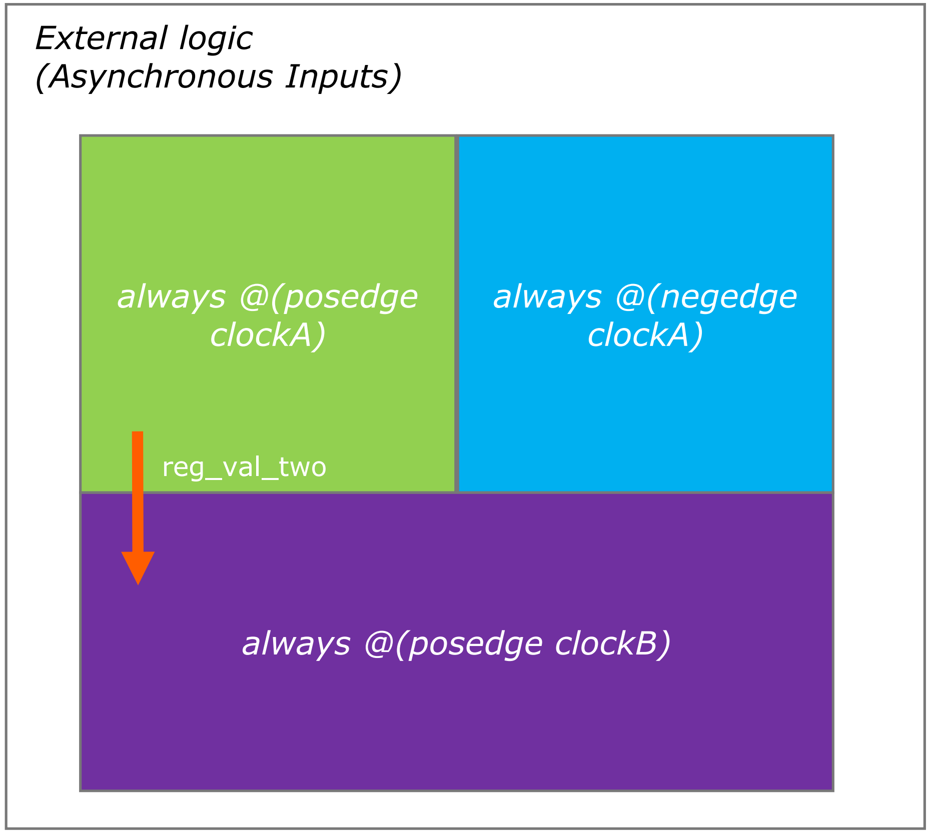
Clock Domain Crossing

- Figure shows a typical CDC design having four clock domains
- All registers and their combinational logic, accessed by the same edge of the same clock, form a single clock domain
- A CDC signal is a signal latched by a register or flip-flop (FF) in one clock domain (sending clock domain) and sampled in another asynchronous or phase variable clock domain (receiving clock domain)

```
reg reg_val_one;
always @(posedge clockA)
            reg_val_one <= some_logic;

reg reg_val_two;
always @(posedge clockB)
            reg_val_two <= some_function_of (reg_val_one);
```

CDC Design



Clock Domain Crossing

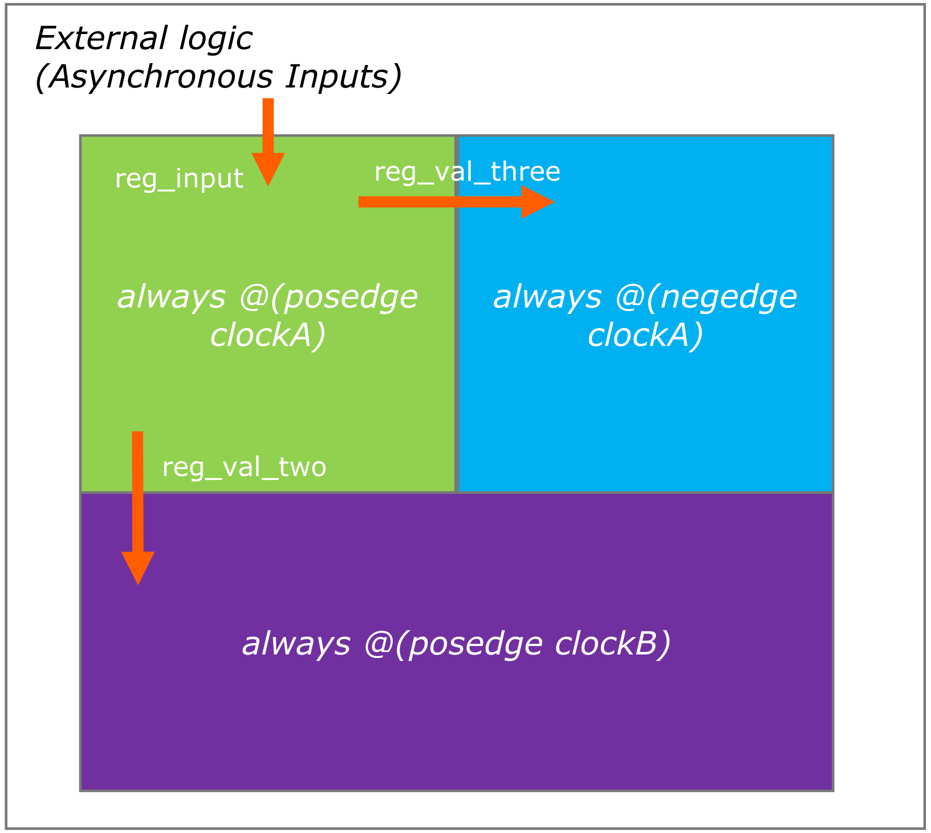
- This definition also applies to signals crossing from one edge to another edge of the same clock

```
reg reg_val_three;
always @(negedge clockA)
          reg_val_one <= some_function_of (reg_val_one);
```

- The third scenario occurs when a register is set in a clock domain with an asynchronous input

```
reg reg_input;
always @(posedge clockA)
          reg_input <= i_input;
```

CDC Design


External logic
(Asynchronous Inputs)
reg_input
reg_val_three
always @(posedge clockA)
always @(negedge clockA)
reg_val_two
always @(posedge clockB)

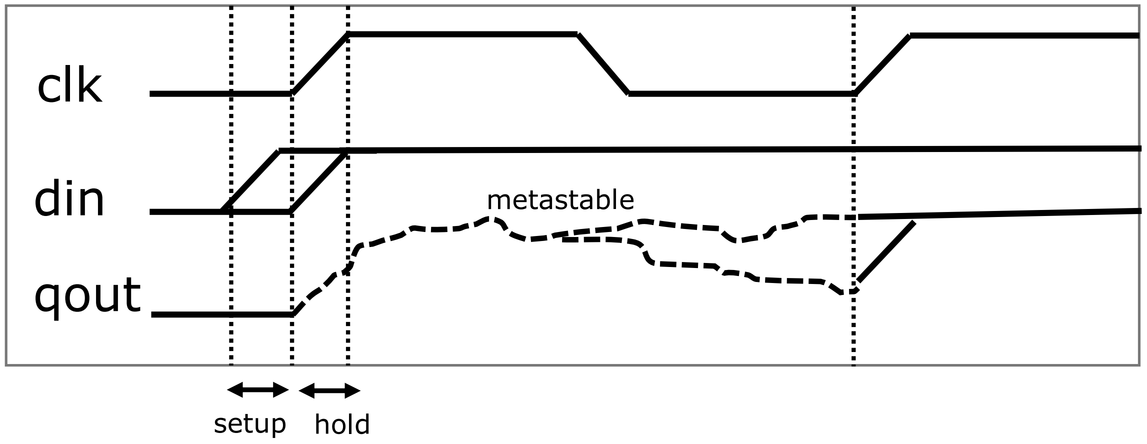
Metastability

- Transferring signals between asynchronous clock domains may lead to setup or hold timing violations of the flip-flops in the receiving clock domain
- These violations may cause CDC signals to be metastable

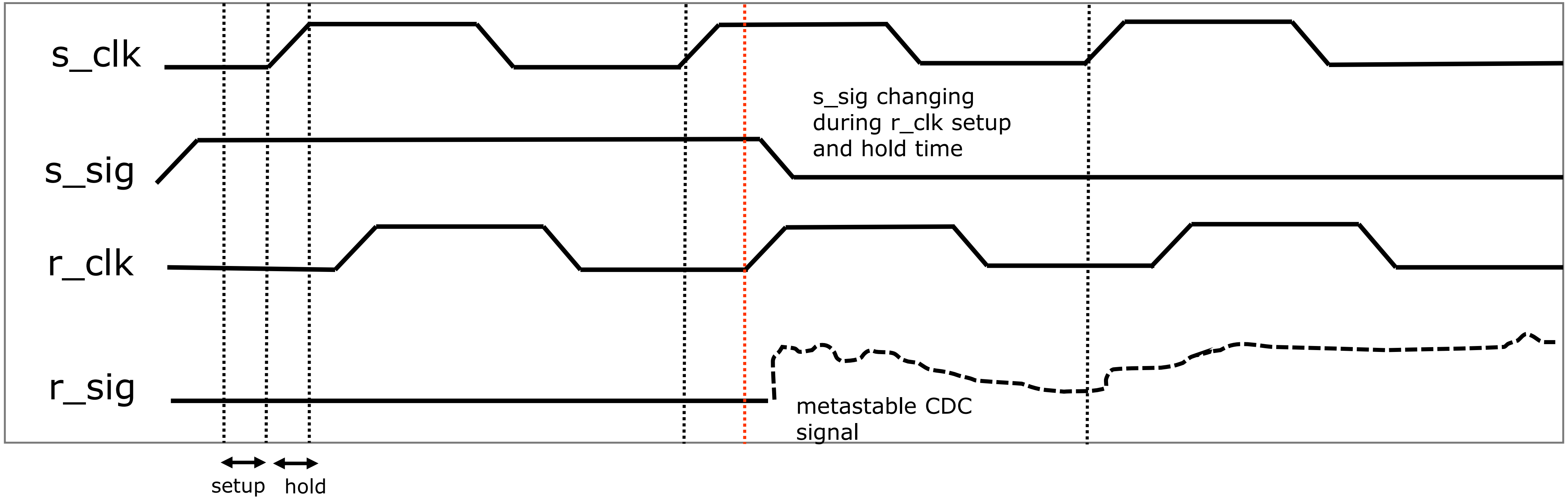
“Metastability” refers to signals that do not assume stable 0 or 1 states for some duration of time at some point during normal operation of a design [1]

- When applied to flip-flops in digital circuits:

“Metastability” refers to a state of indecision where the flip-flop’s output has not settled to the final expected value



Metastability in CDC Signal



- Metastable CDC signal leads to synchronization failure between sending and receiving clock domains
- If a signal (s_sig) generated in the sending clock domain (s_clk) is sampled within the setup and hold time of the active clock edge in the receiving clock domain (r_clk), the output (r_sig) may go metastable and not converge to a legal stable state by the time the it is sampled again

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

The Metastability Problem

- A metastable output that traverses additional logic in the receiving clock domain can cause illegal signal values to be propagated throughout the rest of the design
- Since the CDC signal can fluctuate for some period, the input logic in the receiving clock domain might recognize the logic level of the fluctuating signal to be different values and hence propagate erroneous signals into the receiving clock domain

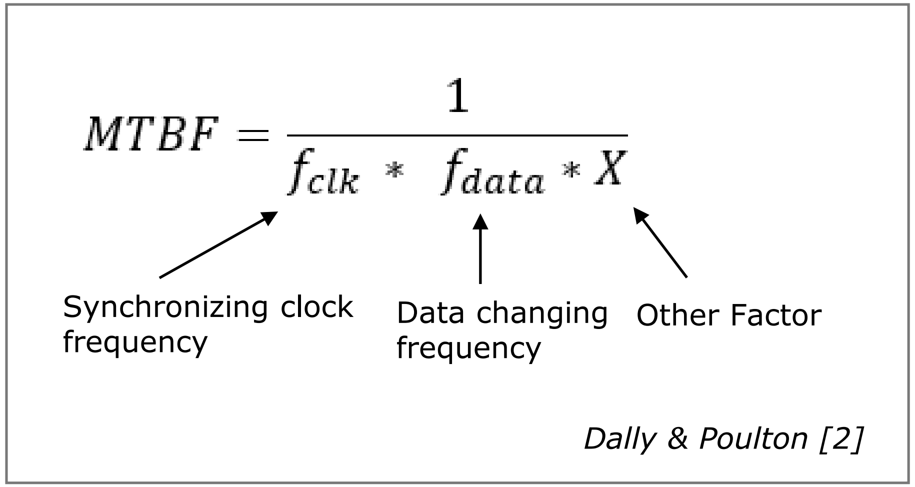
In a multi-clock domain design, metastability is almost "inevitable"

CDC Synchronization Techniques

- Synchronizers
- Two Flip-Flop Synchronizer
- Closed Loop Synchronizer
- Handshake Synchronizer
- Asynchronous FIFO Synchronization Technique

MTBF (Mean Time Between Failures)

- For most CDC design, it is important to run a calculation of the Mean Time Between Failures (MTBF) for any signal crossing a CDC boundary
- MTBF is inversely proportional to the sending clock frequency and the data change frequency
- So high speed designs are more prone to synchronization failures

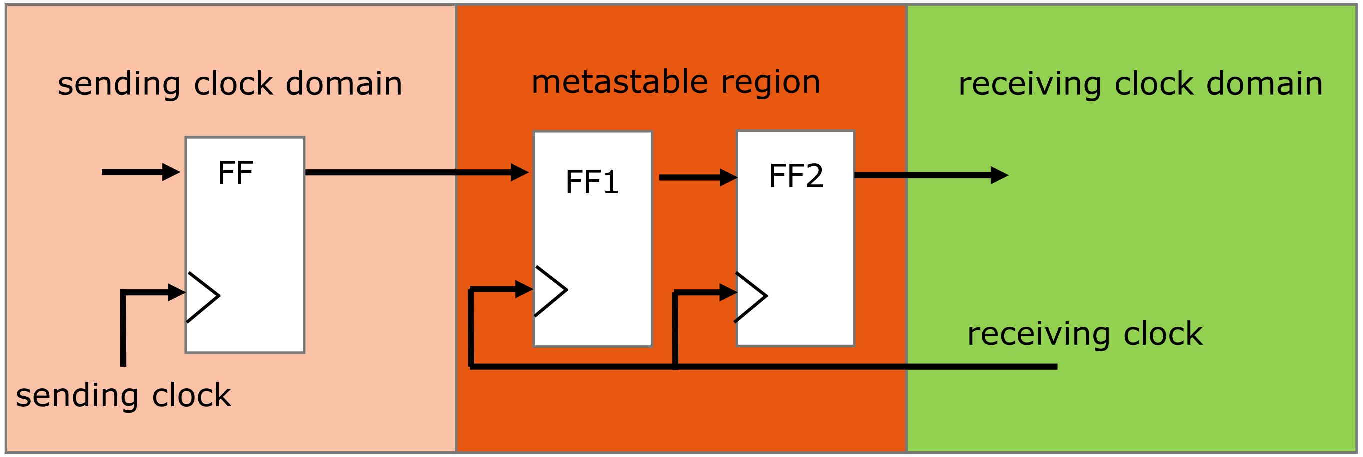


$$MTBF = \frac{1}{f_{clk} * f_{data} * X}$$

Dally & Poulton [2]

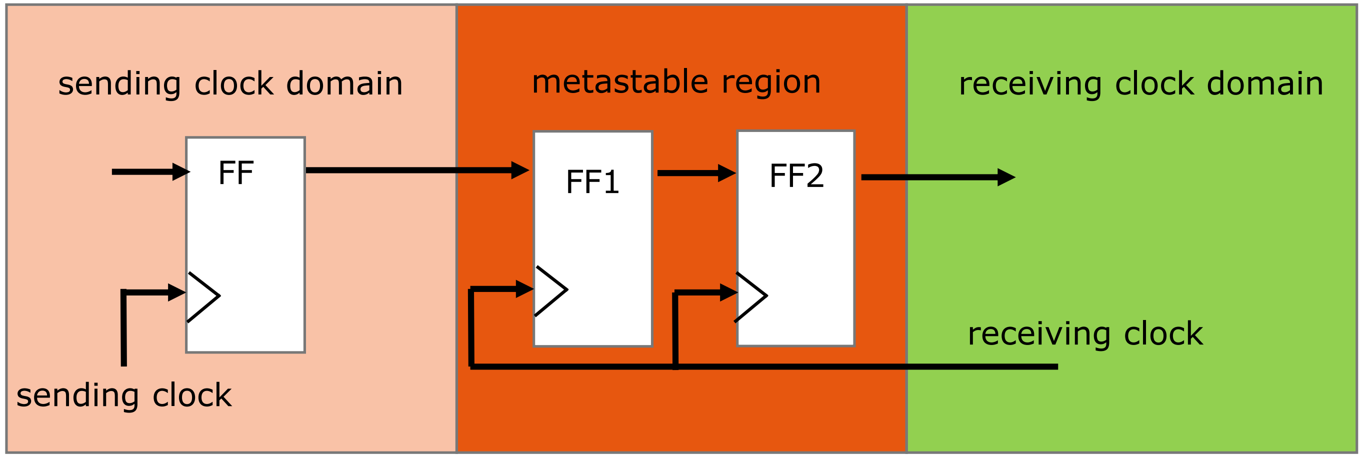
Two-Flip-Flop Synchronizer

- A synchronizer is a device that samples an asynchronous signal and outputs a version of the signal that has transitions synchronized to a local or sample clock [1]
- Two-flip-flop (2-FF) synchronizers are the most common synchronizer used by designers
- Mainly used to synchronize control signals in a CDC design
- In this a CDC signal is passed through a two flip-flops chain clocked with the receiving clock



Two-Flip-Flop Synchronizer

Two-Flip-Flop Synchronizer



Two-Flip-Flop Synchronizer

- The first flip-flop samples the asynchronous input signal into the receiving clock domain and waits for a full clock cycle to permit any metastability on the FF1 output signal to decay, then the FF1 signal is sampled by the same clock into a second stage flip-flop, with the intended goal that the FF2 signal is now a stable and valid signal synchronized and ready for propagation within the receiving clock domain

- Theoretically it is possible that the FF1 signal is still sufficiently metastable when it is clocked into the FF2 to cause its output signal to also go metastable, however its probability is very small

Synchronizing Fast Signals into Slow Clock Domains

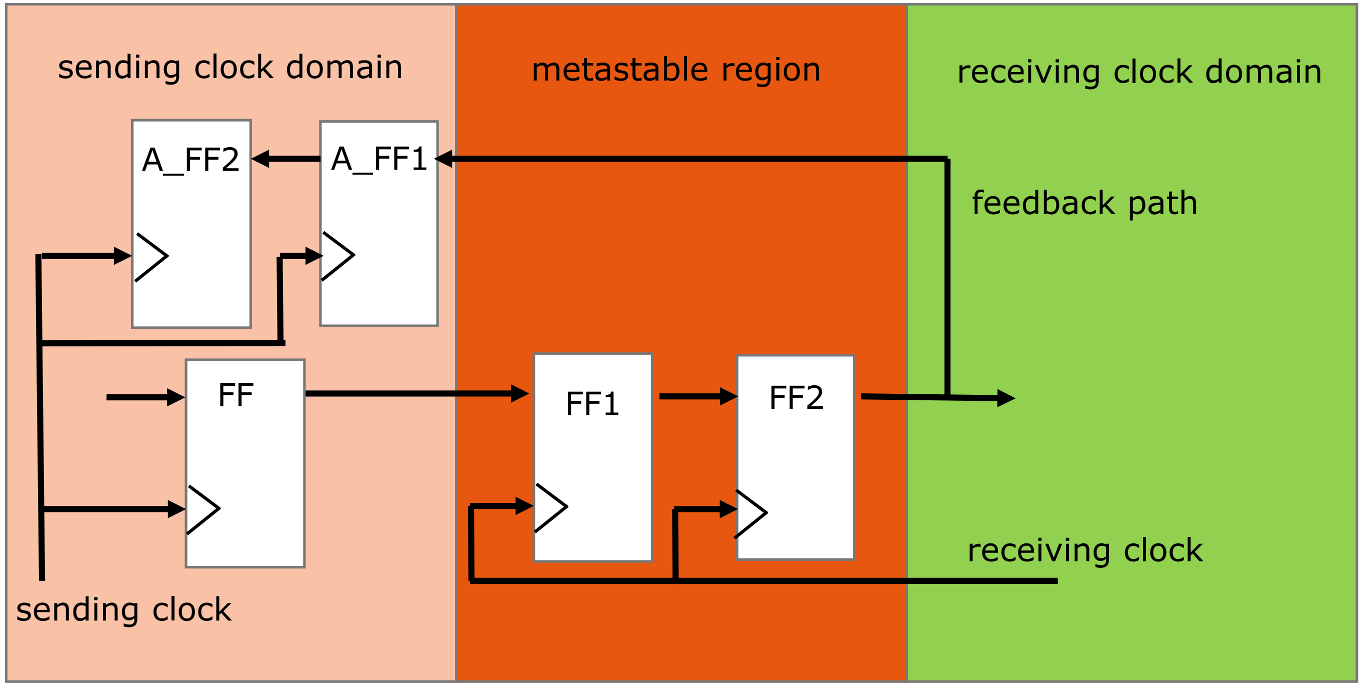
- Certain Multi-Clock Domain designs require that every CDC signal must be sampled
- Fast signals synchronized into slower clock domains may change value before synchronization
- To overcome this, the minimum pulse width for the CDC signal must be 1.5X the period of the receiving clock frequency

This is referred as the “three edge” requirement

- In Open Loop synchronizers, the “three edge” requirement must be met for proper synchronization into the slower clock domain
- Another, better approach is to use a Feedback (Closed Loop) Synchronizer

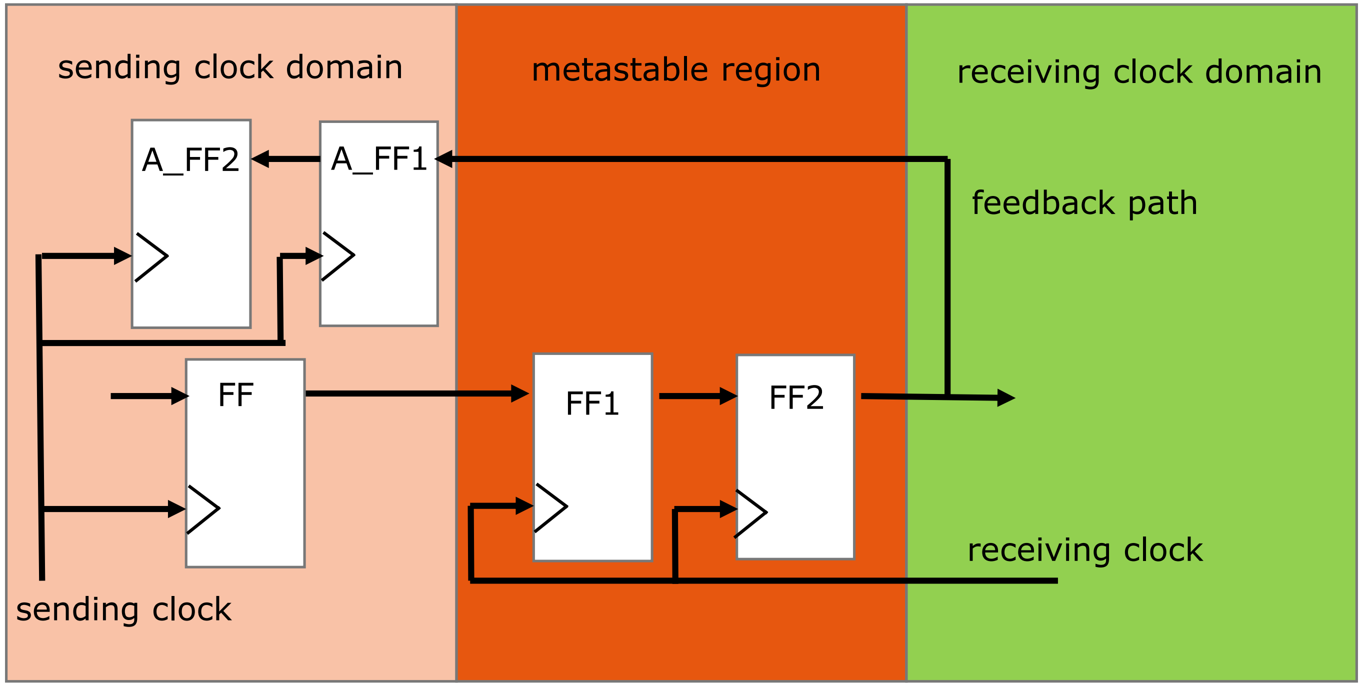
Feedback (Closed Loop) Synchronizer

- In a Feedback Synchronizer a CDC signal is synchronized into the receiving clock domain using a 2-FF chain and then this synchronized signal is passed back through another 2-FF chain to the sending clock domain as an acknowledge signal
- The CDC signal can change only after the acknowledgement is received



Feedback (Closed Loop) Synchronizer

Feedback (Closed Loop) Synchronizer

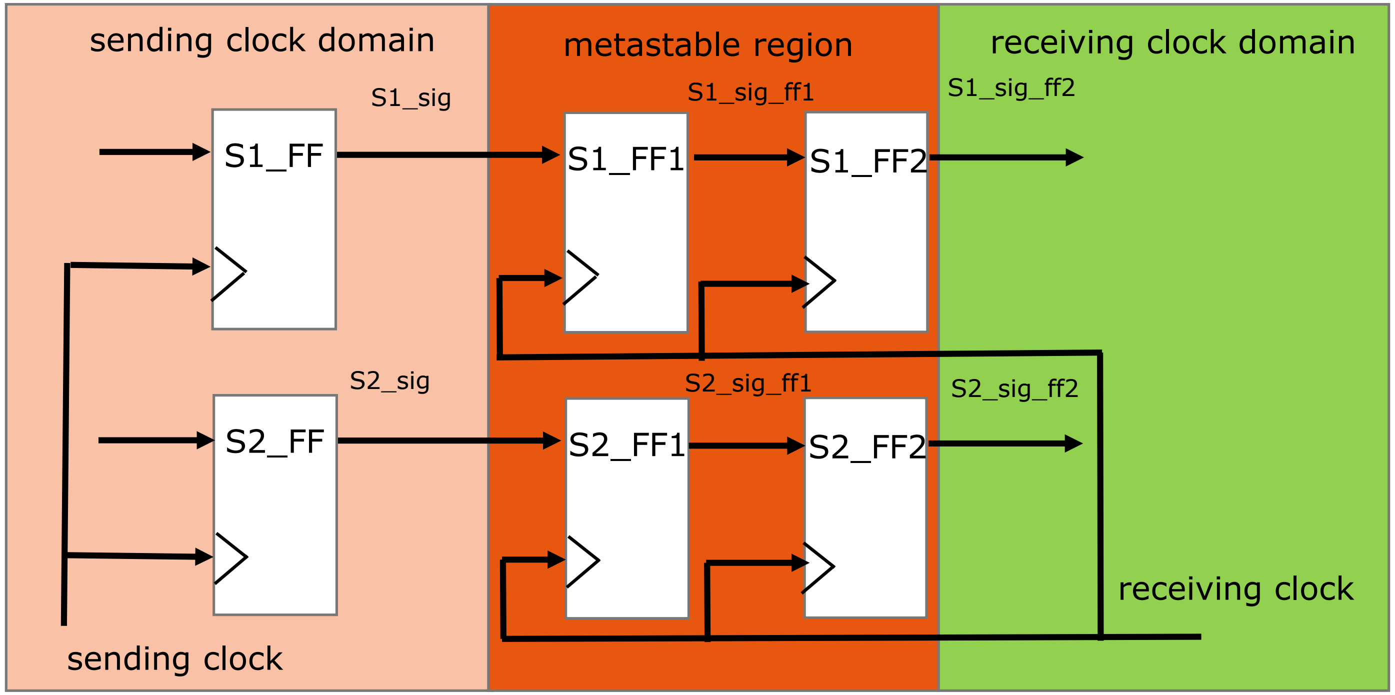


Feedback (Closed Loop) Synchronizer

- A Closed Loop Synchronizer ensures that every change in the CDC signal is sampled into the receiving clock domain
- But this approach could lead to considerable delay as signals are synchronized in both directions before allowing the CDC signal to change

Passing Multiple Signals Across Clock Domain Crossing

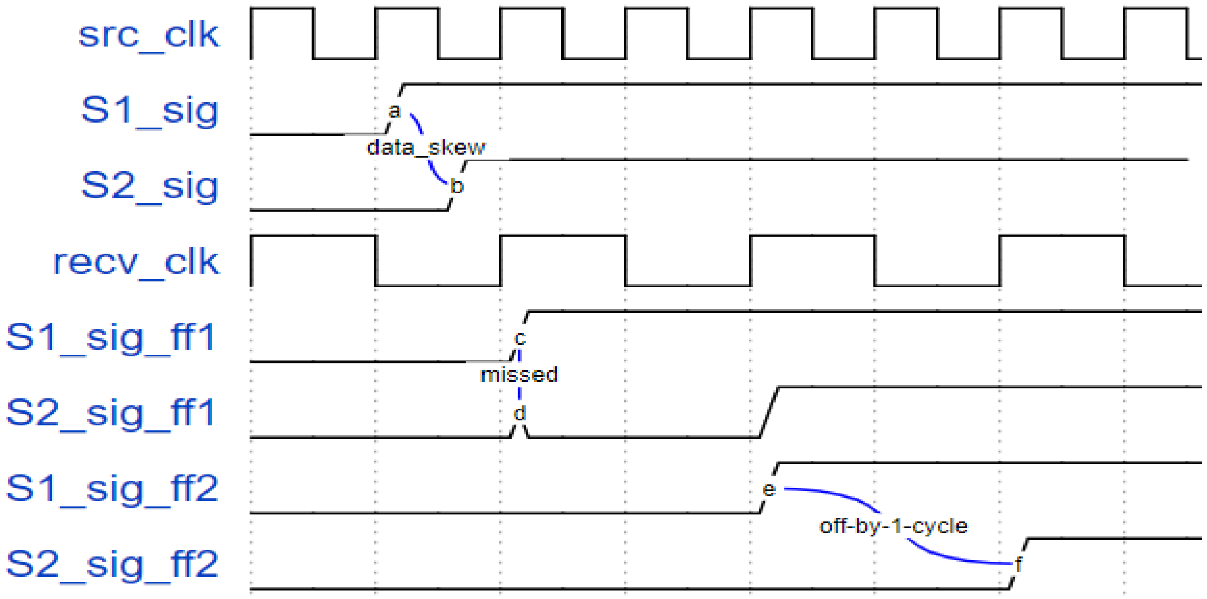
- Multiple signals that are synchronized to one sending clock can experience small data changing skews that result in them being sampled on different clock edges in the receiving clock domain
- Since in an asynchronous clock domain crossing (CDC) design the receiving clock can have every possible alignment relative to the sending clock, the receiving clock can sample at a time when not all the bits are at their stable final values



Multiple Signals crossing CDC boundary

Two Simultaneously Required CDC Signals

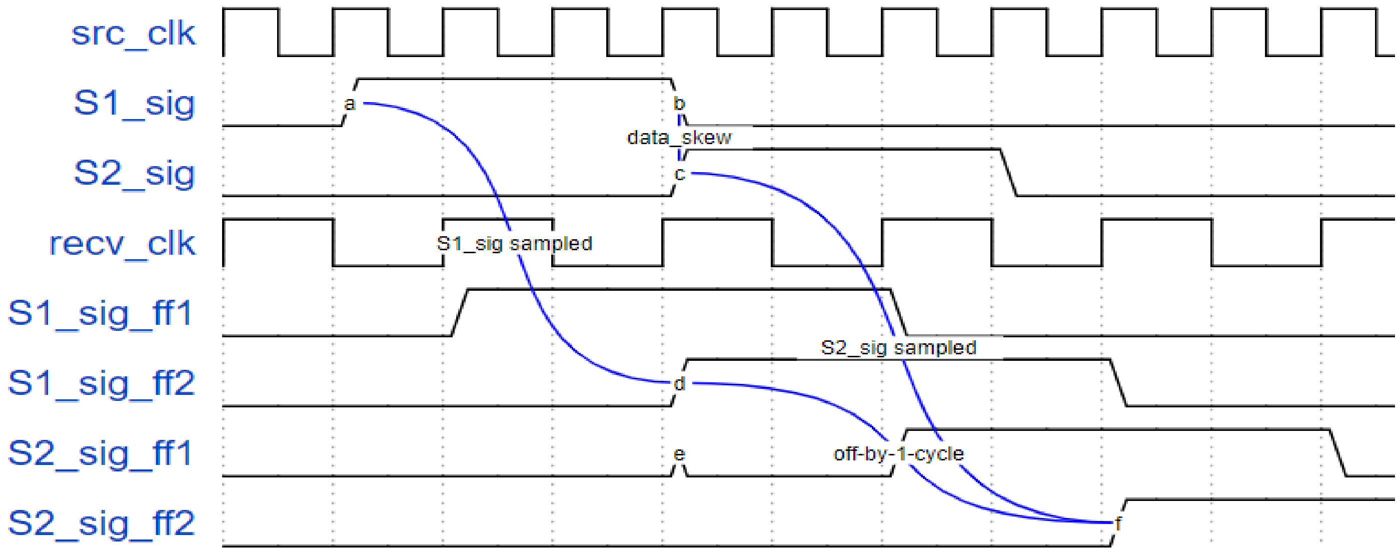
- A small skew from the sending clock domain between simultaneously required CDC signals can cause signals to arrive in different clock cycles in the receiving domain, if they are synchronized *individually* using two flip-flop synchronizers



- If possible, consolidating multiple CDC signals into one is a good design practice

Two Sequenced CDC signals

- Individually synchronizing two CDC signals that require precise sequencing across an asynchronous clock domain crossing (CDC) can result in incorrect design
- Two separate signals that are intended to arrive 1 cycle apart in the receiving domain, can arrive either 1 or 2 cycles apart depending on data skew

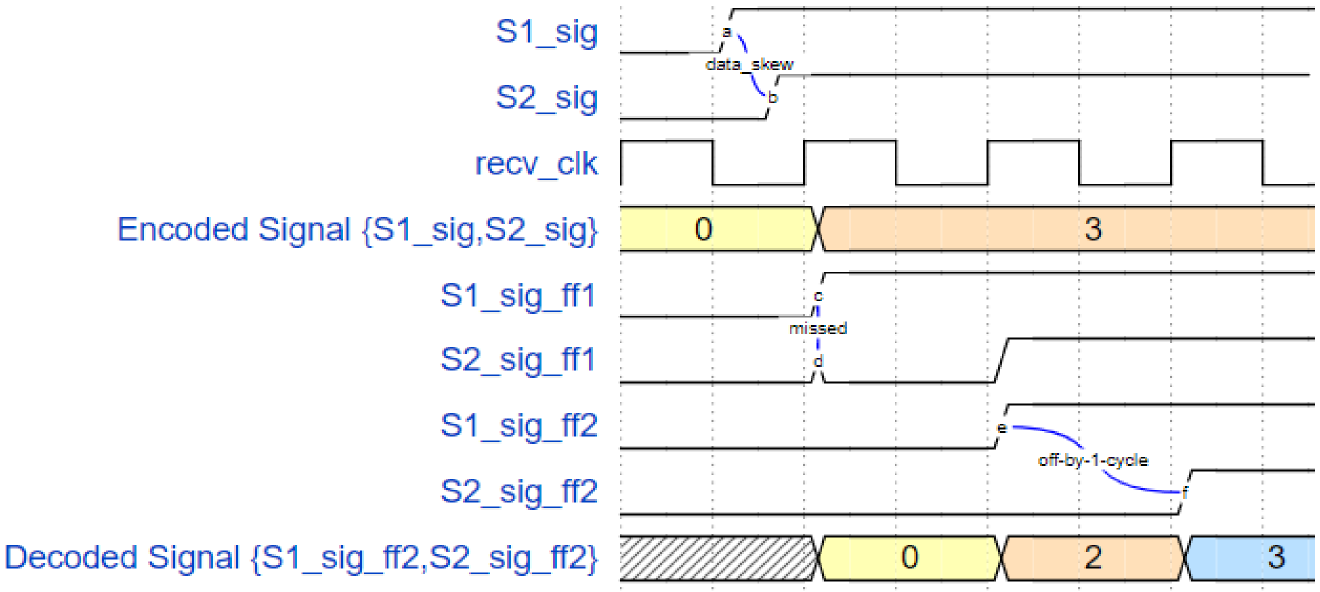


- The solution is to send only one CDC signal into the receiving clock domain and generate the second 1-cycle apart sequenced signal within the receiving clock domain

Solutions for Passing Multiple CDC Signals

- Multiple encoded CDC signals that are slightly skewed and are individually synchronized can result in an incorrect decoded output for one clock cycle when sampled in the receiving clock domain

- Two widely accept solution to resolve multiple CDC signal synchronization issues are:
  - Multi-cycle path (MCP) formulation
  - Asynchronous FIFO



Incorrect! {S1_sig_ff2,S2_sig_ff2} == 2

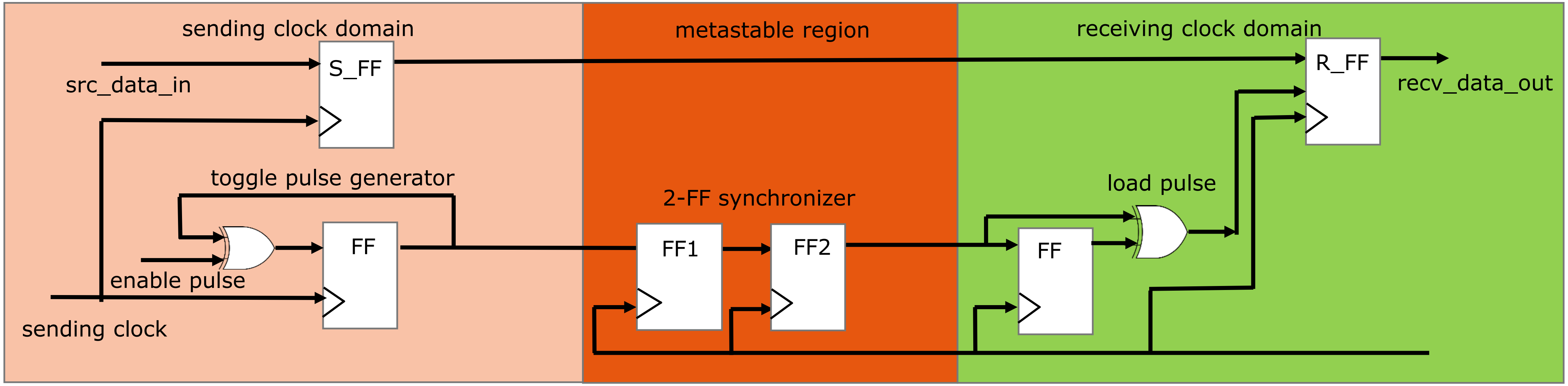
Multi-Cycle Path (MCP) Formulation

- In MCP formulation data is transferred over the CDC using a synchronized control (e.g., a load enable) signal
- This allows the data signals to settle (while the control signal is being synchronized) and be captured together on a single receiving clock edge
- Two Variations:
  - Multi-Cycle Path (MCP) formulation without feedback
  - Multi-Cycle Path (MCP) formulation with feedback (Handshake Synchronizer)

MCP Formulation without Feedback

- An MCP synchronizer consist of the following components
  - Logic that converts a synchronization event (enable/control pulse) from the sending clock domain to a toggle, which then passes across the clock domain crossing (CDC) through a 2-FF synchronizer
  - Logic that converts the toggle back into a load pulse in the receiving clock domain
  - Flip-flops to capture the unsynchronized multi-data bits
- A key feature in this design is that the enable (control) pulse is converted into a single toggle (either low to high, or high to low) before being synchronized into the receiving clock domain
- This is done to make the design independent of the polarity of the enable/control signal

MCP Formulation without Feedback

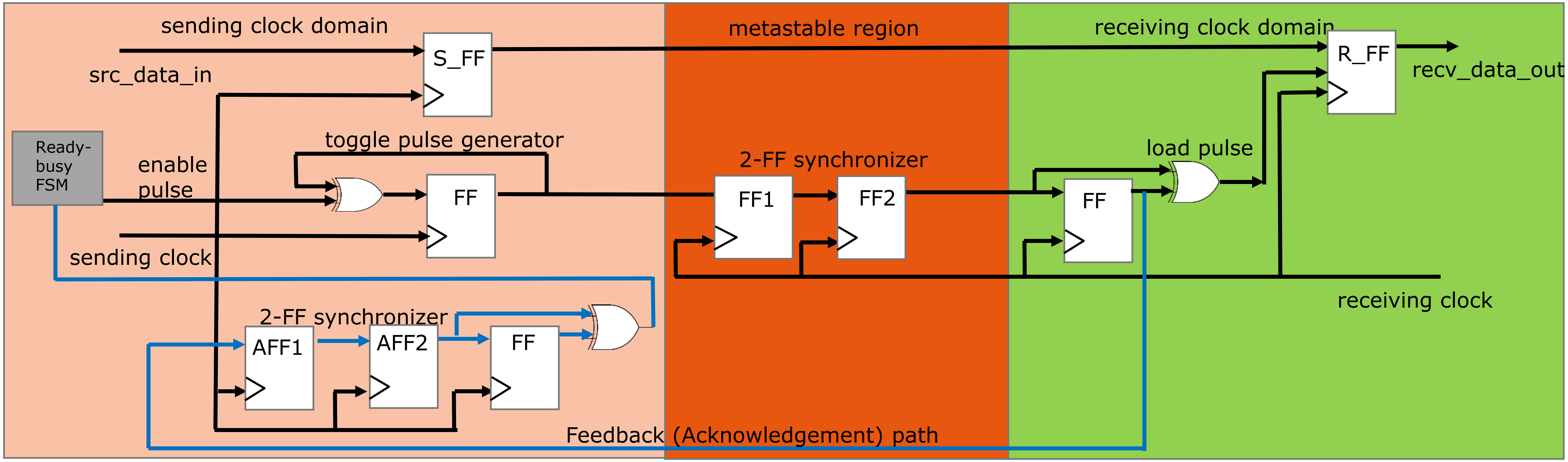


MCP formulation without feedback

- The multi-bit data signal (src_data_in) passes straight from sending clock flip-flop (S_FF) to receiving clock flip-flop (R_FF)
- A single enable/control pulse is synchronized to allow time for the multi-bit data to settle from a possible metastable state
- The enable pulse from the source clock domain first gets converted into a toggle, which is synchronized across the clock domain crossing (CDC), and then gets converted back to a load pulse in the receiving clock domain
- Finally that load pulse is used to load the multi-bit data signal into flip-flops in the destination clock domain

MCP Formulation with Feedback

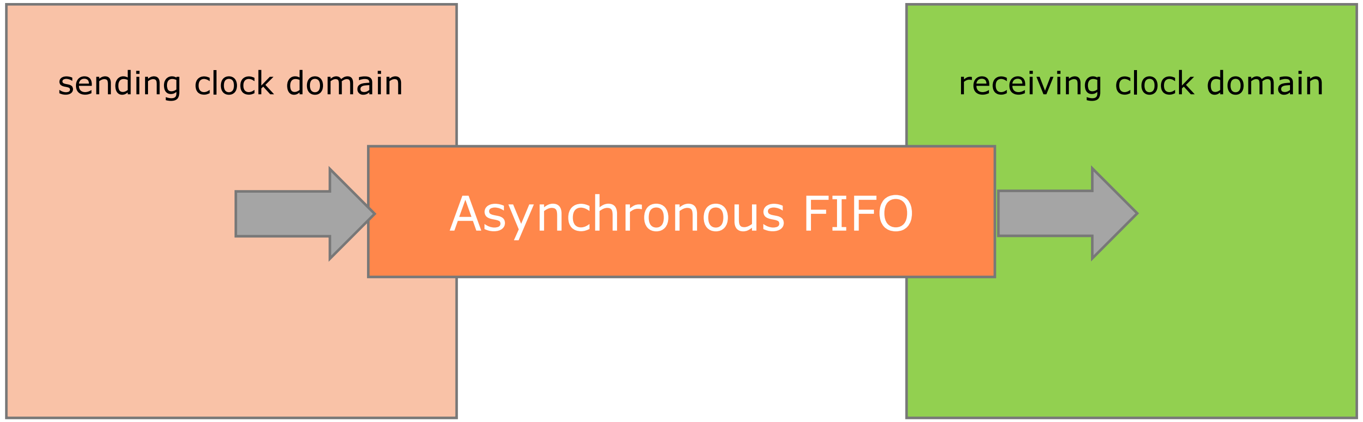
- Commonly referred as Handshake Synchronizer
- In open loop MCP formulation the input data must be held until the synchronization pulse loads the data in the receiving clock domain, to avoid data loss
- To avoid this a feedback acknowledgement path is added
- Feedback path + FSM logic allows the receiving clock domain to block any new data from the sending domain until the previous data is successfully loaded



MCP formulation with feedback (Handshake Synchronizer)

Asynchronous FIFO

- An asynchronous FIFO is a shared memory or register buffer where data is inserted from the write (sending) clock domain and data is removed from the read (receiving) clock domain



- Since both sender and receiver operate within their own respective clock domains, using an asynchronous FIFO is a safe way to pass multi-bit values between clock domains
- The sender puts data into the memory as fast as one data word per write clock
- Similarly, the receiver pulls data out of memory one data word per read clock
- For proper synchronization between the write-read operation, write and read pointers are passed and synchronized into opposite clock domains

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

Asynchronous FIFO

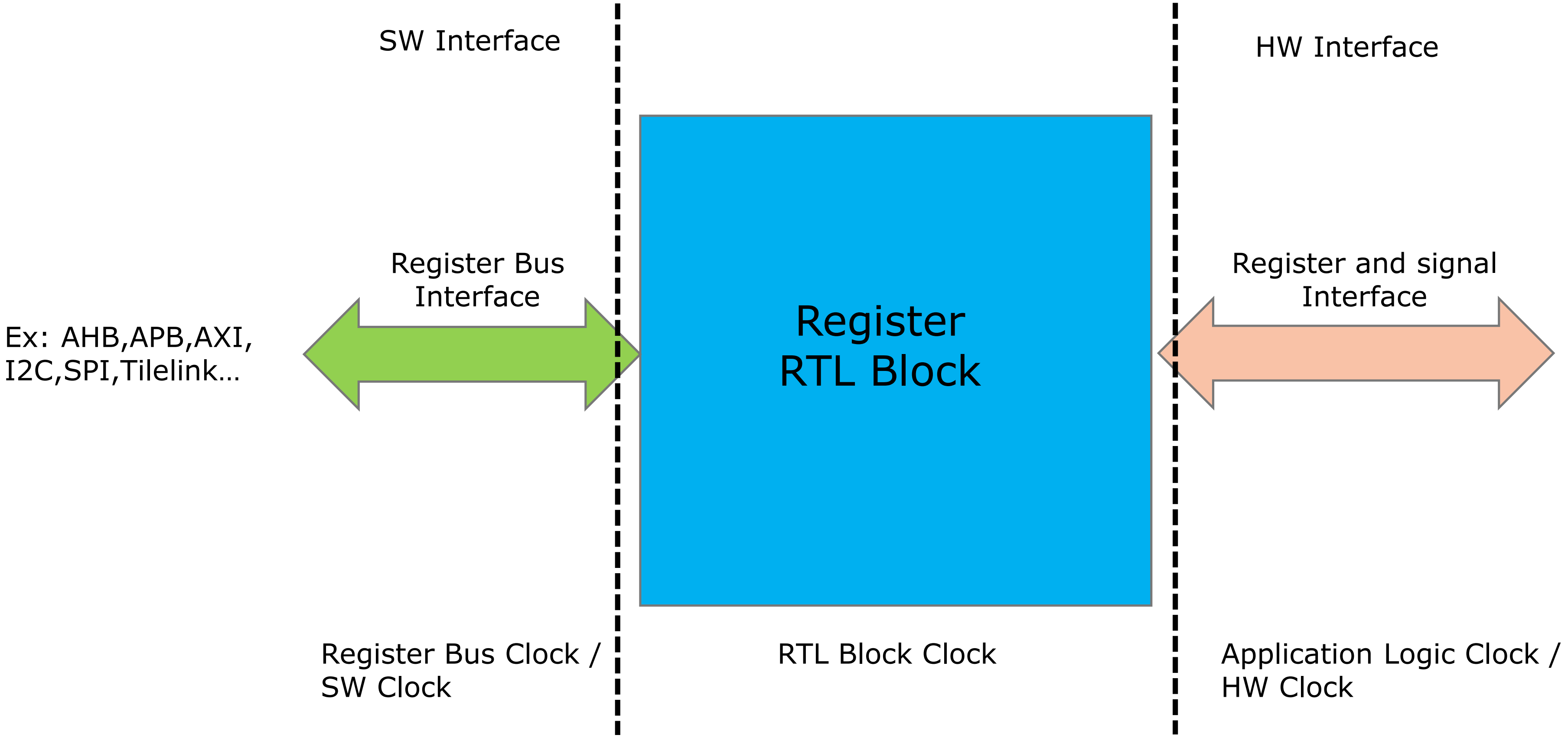
- Write pointer always points to the next FIFO word to be written, while the Read pointer always points to the current FIFO word to be read
- On reset, both pointers are set to zero, which also happens to be the next FIFO word location to be written
- On reset, the read pointer is pointing to an invalid data (because the FIFO is empty and the empty flag is asserted), but as soon as the first data word is written to the FIFO, the write pointer increments, the empty flag is cleared
- FIFO full occurs when the write pointer catches up to the synchronized and sampled read pointer
- FIFO empty occurs when the read pointer catches up to the synchronized and sampled write pointer
- A common approach to FIFO pointers, is to use Gray code counters
- In Gray codes two successive values differ in only one bit

CDC Support in IDesignSpec^{TM} Design

- HW Interface CDC
- SW Interface CDC

Possible Clock Domains in a Block

- A typical Register RTL block in a SoC design can have be accessed by a register bus on the SW interface and by application logic on the HW Interface
- This creates three possible clock domains


SW Interface
HW Interface
Register Bus Interface
Register and signal Interface
Ex: AHB,APB,AXI, I2C,SPI,Tilelink…
Register RTL Block
Register Bus Clock / SW Clock
RTL Block Clock
Application Logic Clock / HW Clock

Three Clock Domains in a Register RTL Block

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

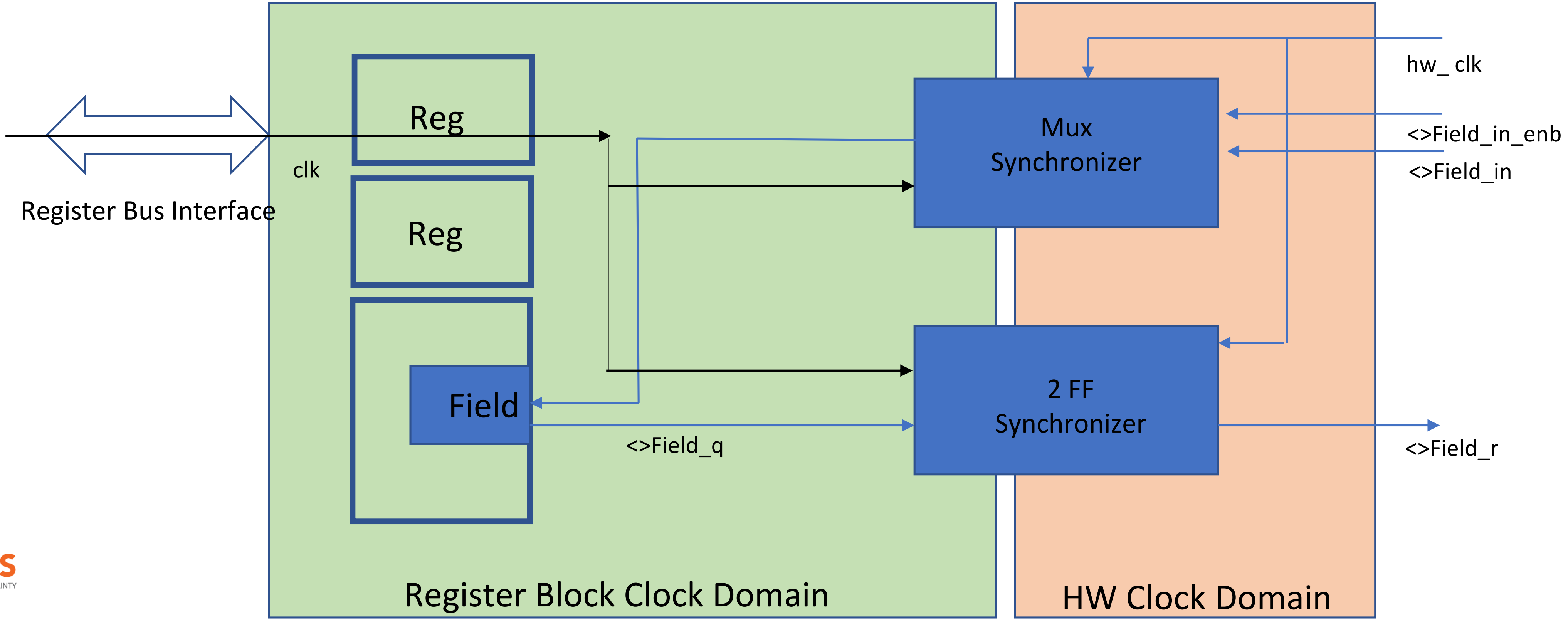
HW Interface CDC

- IDesignSpec^{TM} supports design flows in which Register RTL block and application logic are in different clock domains
- Application logic communicates with the Register block through a HW Interface, which comprises different data and control signals
- Both widely used Two Flip-Flop synchronization and sophisticated Handshake synchronization techniques are supported on the HW Interface to cater to different design requirements
- In a CDC design, HW Interface mainly consists of the following signals:

<>field_in_enb	: Register Field write enable signal
<>field_in	: Register Field write data bus
<>field_r	: Register Field read data bus
hw_clk	: Application Logic (Hardware) clock

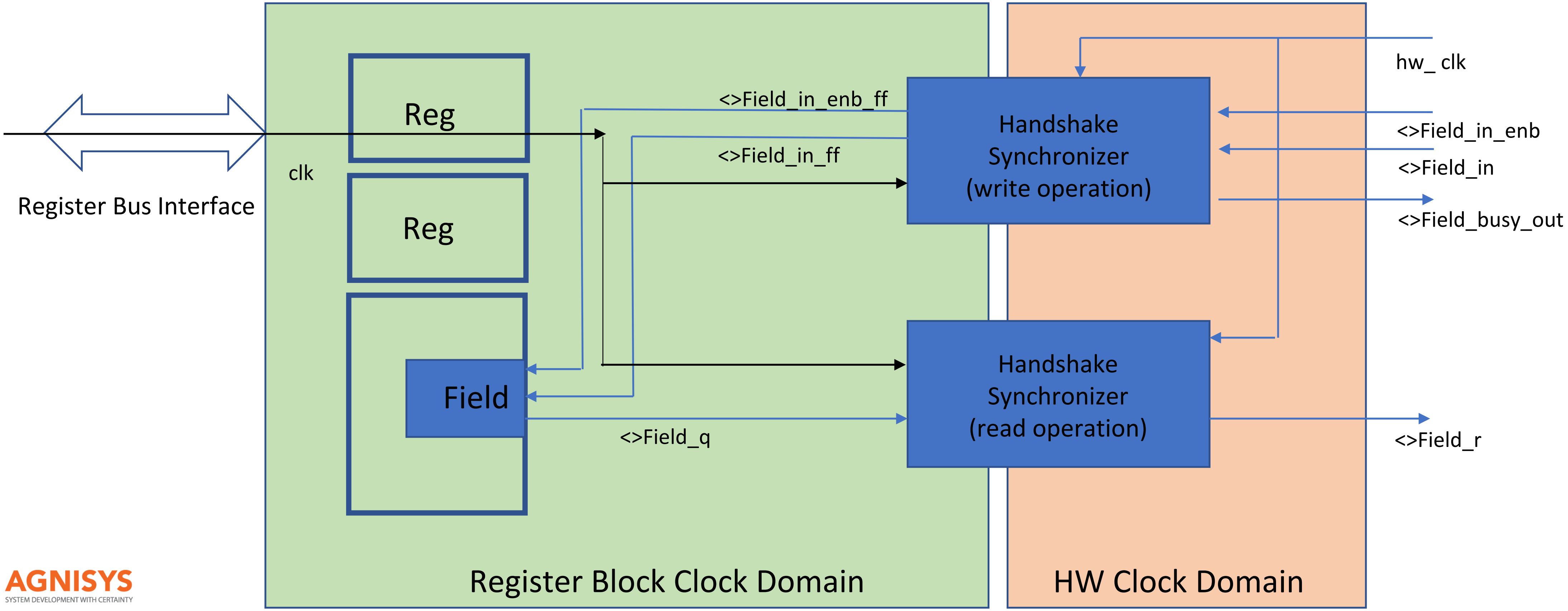
Two Flip-Flop and Mux Synchronization

- Mux synchronizer is a without feedback MCP formulation
- It is used to synchronize the write data bus from the HW clock domain into the Register block clock domain using the write enable control signal
- A Two Flip-Flop synchronizer is used to synchronize read data from the Register block clock domain to the HW clock domain



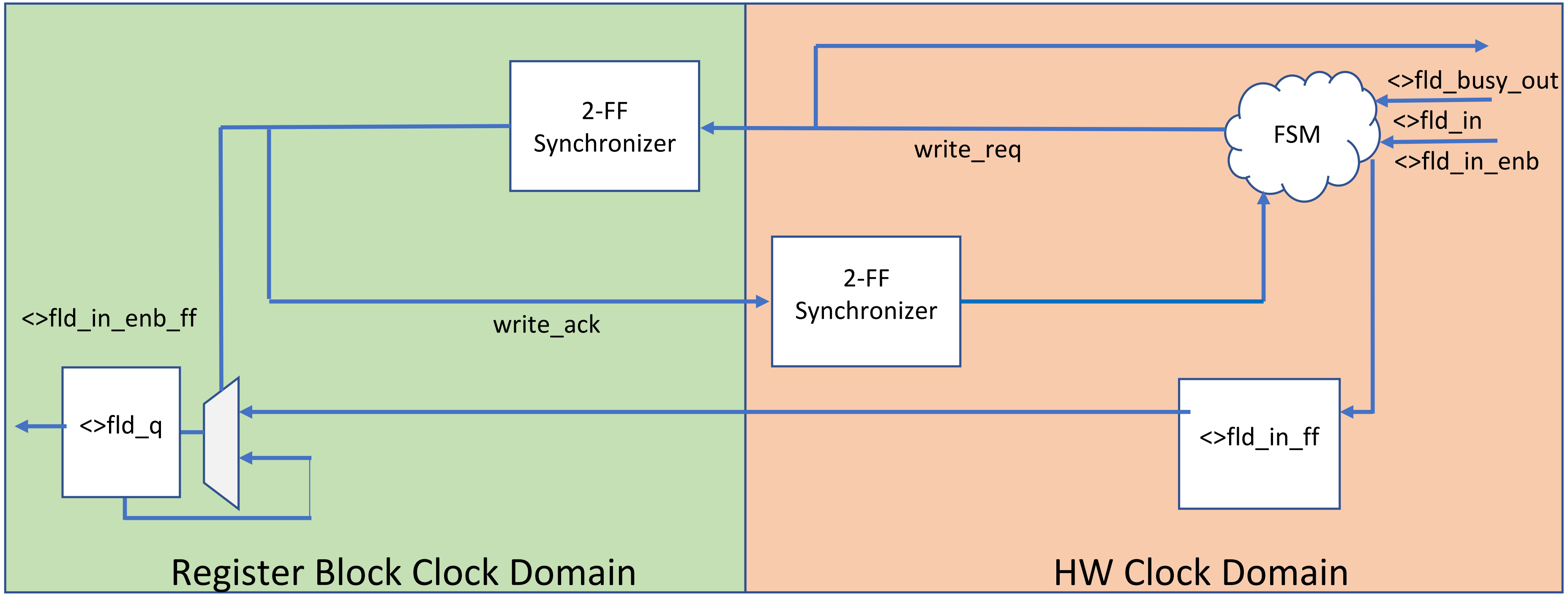
Handshake Synchronization

- Open loop synchronization techniques do not ensure that every data change is synchronized and sampled in the opposite clock domains
- To resolve this limitation Handshake synchronizers are used on both read and write data paths



Handshake Synchronization (Write Operation)

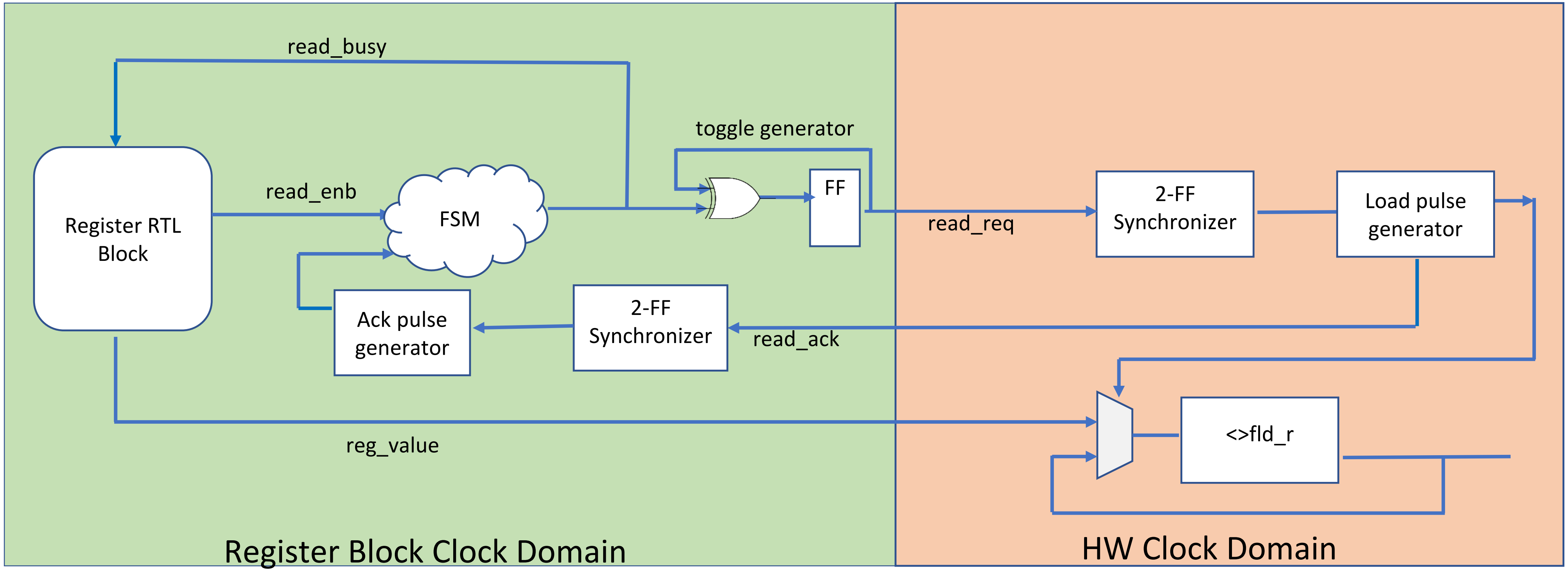
- The write request (<>fld_in_enb) is synchronized into the Register block clock domain using a 2-FF synchronizer and then used to load the registered write data (<>fld_in_ff) into the register storage (<>fld_q)
- The feedback acknowledge path is used to hold the field write busy (<>fld_busy_out) signal
- The write data value should only be changed if the write busy signal is low



AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

Handshake Synchronization (Read Operation)

- A value change event on the register field in the register block clock domain is synchronized into the HW clock domain using a toggle pulse, which is then converted back to load the read data onto the HW read bus <>fld_r
- A read_busy signal is set to restrict any value change during an ongoing read operation



HW Interface CDC Properties in IDesignSpec™

- CDC synchronization techniques can be configured on the HW interface of a register using the below properties (implemented as UDPs in SystemRDL):

```
property cdc_clk {type = string; component = reg;};
addrmap Block1{
  reg Reg1{
    cdc_clk = "hw_clk";
    field {}fld[0:0];
  };
  Reg1 Reg1;};
```

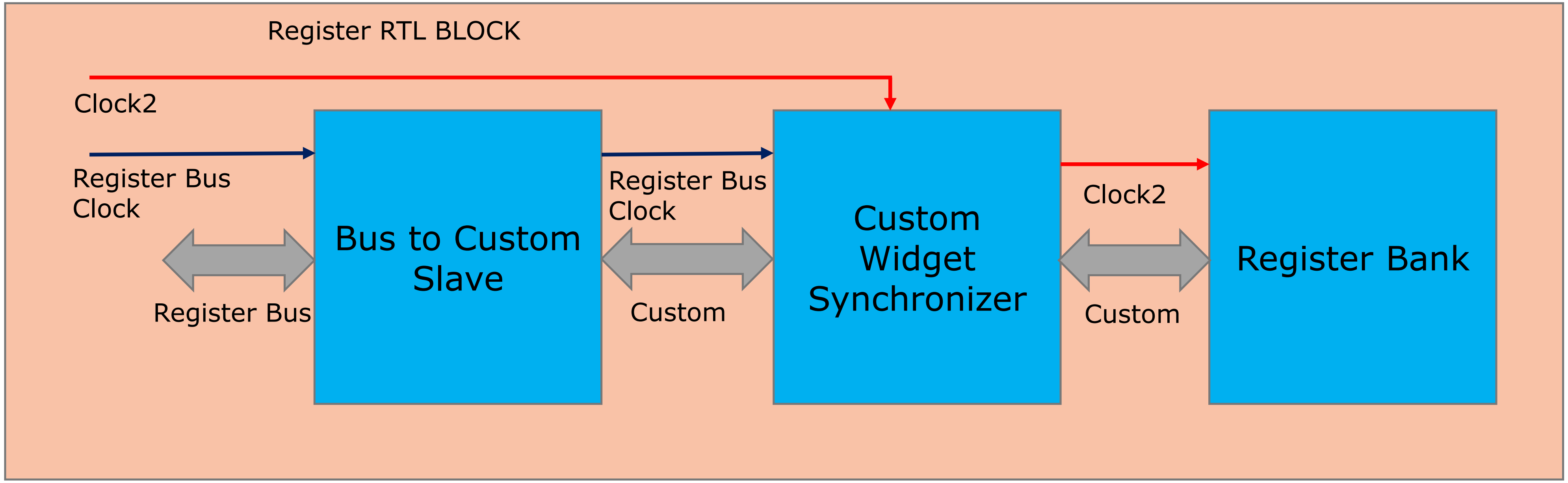
SystemRDL Specification defining 2-FF and Mux synchronization technique for Reg1

SystemRDL Specification defining Handshake synchronization technique for Reg1

```
property cdc_clk {type = string; component = reg;};
addrmap Block1{
  reg Reg1{
    cdc_clk = "hw_clk:handshake";
    field {}fld[31:0];
  };
  Reg1 Reg1;};
```

SW Interface CDC

- Sometimes it is required that the Register RTL block operates on a clock which is different from the Register bus clock
- An additional input clock is created, on which read and write operations on registers take place, thus the Register bus and the RTL block are in two different clock domains
- The standard bus transactions are first translated into a custom bus protocol, which is an internal IDesignSpec™ bus protocol, and then custom control and data signals are synchronized into the Register block bus domain using appropriate Handshake synchronization techniques



SW Interface CDC Properties in IDesignSpec™

- CDC synchronization techniques can be configured on the SW interface using the {reg.clk= <clock_name>} property (UDP)

```
property reg_clk {type = string; component = block;};
addrmap Block1{
  reg_clk = “Clock2";
  reg Reg1{
    field {}fld[0:0];
  };
  Reg1 Reg1;};
```

SystemRDL Specification for placing Block1 on another clock domain, “Clock2”, w.r.t the register bus

Summary and References

Summary

- In multi-clock domain designs metastability is inevitable and Clock Domain Crossing (CDC) errors can cause serious design failures
- These expensive failures can be avoided by following critical design guidelines and using correct and well-established synchronization techniques

References

[1] Clifford E. Cummings, “Clock Domain Crossing (CDC) Design & Verification Techniques Using SystemVerilog,” SNUG 2008

[2] William J. Dally and John W. Poulton, Digital Systems Engineering, Cambridge University Press, 1998

[3] Ran Ginosar ,”Metastability and Synchronizers: A Tutorial”, IEEE CS, 2011

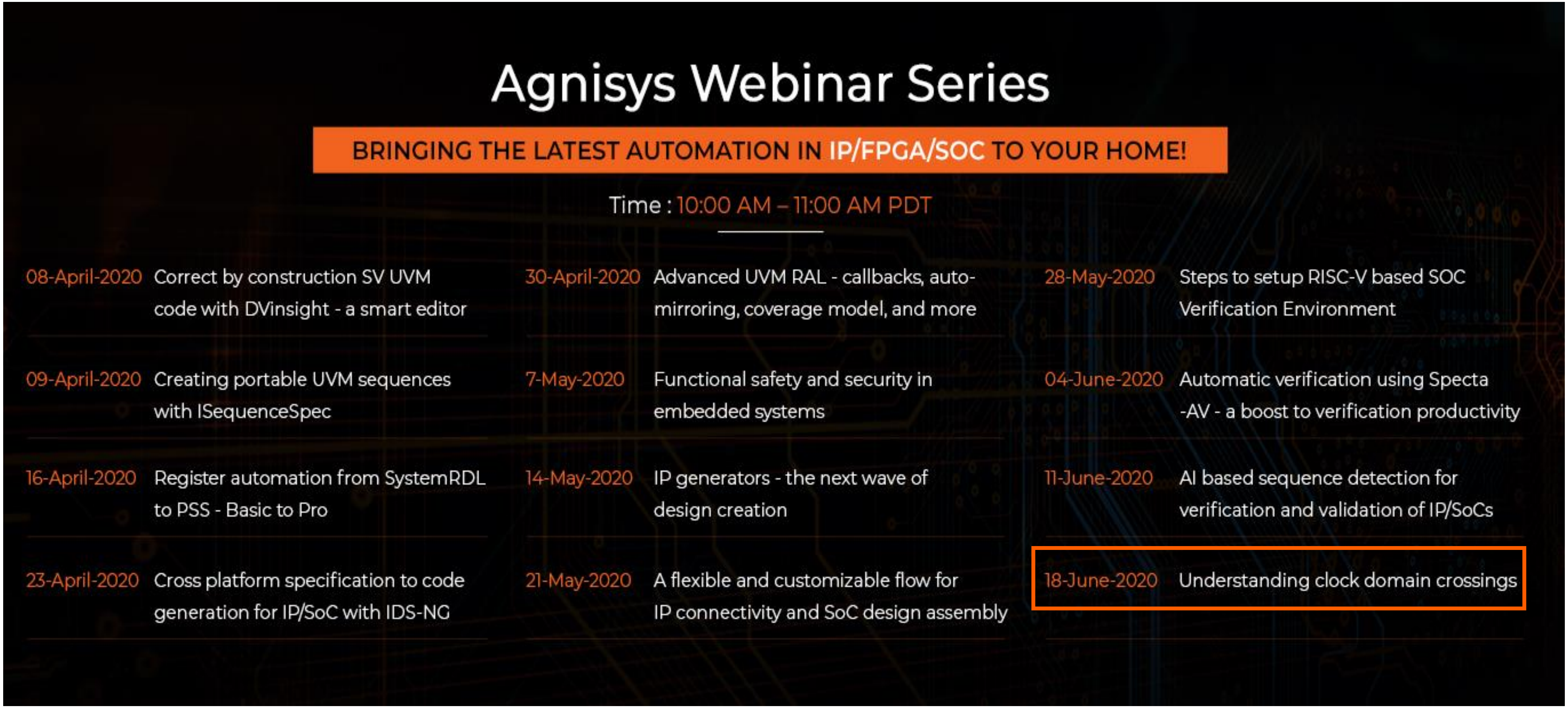
[4] “Some Simple Clock-Domain Crossing Solutions”, https://zipcpu.com/blog/2017/10/20/cdc.html

[5] Jason Yu, “Clock Domain Crossing Design – 3 Part Series”, https://www.verilogpro.com

[6] Clifford E. Cummings, “Simulation and Synthesis Techniques for Asynchronous FIFO Design,” SNUG 2002

[7] https://filebox.ece.vt.edu/~athanas/4514/ledadoc/html/pol_cdc.html

[8] “Clock Domain Crossing (CDC)”, https://www.agnisys.com/release/docs/ids/ClockDomainCrossing.html



Agnisys Webinar Series

BRINGING THE LATEST AUTOMATION IN IP/FPGA/SOC TO YOUR HOME!

Time : 10:00 AM – 11:00 AM PDT

08-April-2020	Correct by construction SV UVM code with DVinsight - a smart editor
09-April-2020	Creating portable UVM sequences with ISequenceSpec
16-April-2020	Register automation from SystemRDL to PSS - Basic to Pro
23-April-2020	Cross platform specification to code generation for IP/SoC with IDS-NG
30-April-2020	Advanced UVM RAL - callbacks, auto-mirroring, coverage model, and more
7-May-2020	Functional safety and security in embedded systems
14-May-2020	IP generators - the next wave of design creation
21-May-2020	A flexible and customizable flow for IP connectivity and SoC design assembly
28-May-2020	Steps to setup RISC-V based SOC Verification Environment
04-June-2020	Automatic verification using Specta -AV - a boost to verification productivity
11-June-2020	AI based sequence detection for verification and validation of IP/SoCs
18-June-2020	Understanding clock domain crossings

About Agnisys

- The EDA leader in solving complex design and verification problems associated with HW/SW interface
- Formed in 2007
- Privately held and profitable
- Headquarters in Boston, MA
  - ~1000 users worldwide
  - ~50 customer companies
- Customer retention rate ~90%
- R&D centers (US and India)
- Support centers - committed to ensure comprehensive support
  - Email : support@agnisys.com
  - Phone : 1-855-VERIFYY
  - Response time within one day; within hours in many cases
  - Multiple time zones (Boston MA, San Jose CA and Noida India)

IDESIGNSPEC™ (IDS)   EXECUTABLE REGISTER SPECIFICATION

Automatically generate UVM models, Verilog/VHDL models, coverage model, software model, etc.

AUTOMATIC REGISTER VERIFICATION (ARV)

ARV-Sim™ : Create UVM test environment, sequences, and verification plans, and instantly know the status of the verification project

ARV-Formal™ : Create formal properties and assertions, and coverage model from the specification

ISEQUENCESPEC™ (ISS)

Create UVM sequences and firmware routines from the specification

DVinsight™ (DVi)

Smart editor for SystemVerilog and UVM projects

IDS – Next Generation™ (IDS-NG)

Comprehensive SoC/IP spec creation and code generation tool

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY