Clock Domain Crossing Standard
Version 0.5 Draft for Public Review

April 14, 2025

Abstract: In-house and externally purchased IPs are often combined in SOCs. It can prove challenging to verify these SOCs because a mixture of verification tools and methodologies that do not integrate can be used. Ensuring that a common clock domain crossing interface standard that every tool can translate their native format to and from is the intent of this standard. With this interface standard, every IP developer's verification tool of choice is run to verify and produce collateral, and the standard format is generated for SOCs that used a different tool. And with this standard, efficiently translating from provided collateral into a tool of choice is possible for every SOC. A way to specify all the information necessary to do accurate clock domain crossing, reset domain crossing, and glitch structural analysis is afforded. Attributes for a block that can be used to facilitate a correct clock domain crossing and reset domain crossing integration of that block in an encompassing design are identified. The limitations of the set of attributes are also addressed; that is, the clock domain crossing and reset domain crossing schemes for which the set of attributes is sufficient and the schemes the defined set of attributes is not guaranteed to support are identified.

Keywords: CDC, clock domain crossing, glitch, IP-XACT, RDC, reset domain crossing, functional verification.

Notices

Suggestions for improvements to the Clock Domain Crossing Standard 0.5 Draft for Public Review are welcome. They should be posted to the Clock Domain Crossing (CDC) community forum at:

https://forums.accellera.org/forum/56-cdc-draft-lrm-release-discussion/

The current Working Group (WG) web page is:

https://accellera.org/activities/working-groups/clock-domain-crossing

Participants

The CDC WG operates on an entity-based model. The following registered members have made contributions to the standard at various milestones or participated in the workgroup as observers.

1 At the time of standardization, the CDC WG had the following eligible voters:

Agnisys, Inc.

ARM Limited

Blue Pearl Software

Infineon Technologies AG

Intel Corporation

Marvell Technology, Inc.

Microsoft

QualcommTechnologies, Inc.

Renesas Electronics Corp.

Siemens EDA

STMicroelectronics N.V.

Synopsys, Inc.

Introduction

The purpose of this standard is to provide the electronic design automation (EDA), semiconductor, and system design communities with a well-defined specification for unified handling of CDC by vendor tools used across IPs and SOCs.

Industry design style can be largely classified into two types (not exhaustive), namely:

Monolithic design: The entire product and its various hierarchies are all designed by one team. All aspects of the design are accessible (and editable) by that team. This type requires knowledge and expertise of all aspects of the design but provides control and autonomy over all aspects of the design (including tools and methodology). Depending on how extensive the product is, and how large (or small) the team is, this style can require a considerable time investment.

IP/SOC design: The product is composed of various IPs that can be designed in-house (by this team, or another team) or purchased externally. This means that the required knowledge and expertise of all aspects of the design are not available in the SOC team, and as a result, the SOC team has less autonomy and control over all aspects of the design. This style can significantly accelerate product development depending on the quality of the IPs and how easy it is to integrate into the product SOC.

NOTE—Most of the industry is shifting from monolithic design styles to IP/SOC design styles, and many IP companies provide their IP to multiple SOC product companies. In addition, there is no expectation that every IP and every SOC company is using the same tools and methodology for validating the design, including CDC analysis.

For most collateral types (for example, SystemVerilog, cluster test environment, low power, and so forth), there exist standards that govern its use; hence, integration for these types of collateral has been largely straight-forward. But for CDC collateral the industry has not had a clean way to handle tool and methodology differences across IPs and SOCs.

The following list describes various methods of handling the differences in the absence of this standard proposal:

a) Monolithic SOC design: Gives tool and methodology autonomy but delays time-to-market.

b) Black-boxing IP: Assumes the IP is clean and ignores checking for integration issues by black-boxing. This helps with initial time-to-market but introduces quality risks that can lead to silicon re-spin, which eventually impacts the product's time-to-market.

c) Re-verification of IP: Re-verifies IPs on CDC tools that might be different from those used by the IP provider. These teams lack the knowledge and expertise to do a thorough and efficient job, thus putting both time-to-market and quality at risk, even after employing considerable resources and effort.

d) Common tools between SOC and IPs: Requests all IP companies they purchase from provide collateral analyzed with their tool of choice. In this scenario, IP companies that provide IP to different SOC companies must run multiple tools to fit all SOC needs. Even if the IP company agrees, they push back on their delivery date to master new tools and collateral, which eventually impacts SOC time-to-market.

None of the approaches above is able to provide sufficient quality in reasonable time. However, defining a standard format to capture clock domain crossing (CDC), reset domain crossing (RDC), and glitch intent enables interoperability of CDC collateral generated by any CDC verification tool.

Contents

List of figures

List of tables

Clock Domain Crossing Standard Version 0.5 Draft for Public Review

1. Overview

This common CDC interface standard provides the following:

a) Every vendor’s tool can translate its native format to and from the standard to maintain its IP.

b) Every IP provider can run its tool of choice to verify and produce collateral and generate the standard format for SOCs that use a different tool.

c) Every SOC can quickly and safely integrate either native collateral or translate from the standard collateral into their tool of choice to ensure time-to-market goals and quality.

A limited feasibility study was conducted on a subsystem with multiple IPs connected by Arm® AMBA® interfaces across three vendor tools with limited support from the vendors.[1] It showed that 99.5% of what was identifiable in a flat run using any of the three vendor tools was also identifiable if the native abstraction collateral was replaced with an XML representation and translated across the vendor tools.[2]

1.1 Scope

The scope of work of the Clock Domain Crossing Standard is limited to:

a) Support tool-independent output collateral for CDC, RDC, and glitch structural analysis

b) Provide human-readable and machine-parsable attributes

c) Support customizable extensions (for example, to support complex user conditions)

d) Support hierarchical analysis

e) Support power-aware designs

f) Support multi-modal IP/SOC analysis

g) Support multi-instance IPs

h) Support parameterized IPs

i) Support multiple interface protocols (for example, AMBA, I2C, PCIe, UCIe, and so forth)

j) Provide extensibility to cover input collateral cases (for example, constraints, waivers, and so forth) to enable high-quality re-verification of an IP using alternate tools

k) Support assertions necessary to complement CDC, RDC, and glitch structural analyses

[1]AMBA® is a registered trademark of Arm® Limited (or its subsidiaries) in the US and/or elsewhere

[2]This feasibility study was only for CDC and did not include reset domain crossing (RDC) or glitch analysis.

l) Support other design styles (for example, FPGA and Analog) that follow similar standards

NOTE—The “interface protocols” item above is a way to ensure that multiple circuit styles (CDC crossing types) are supported to maintain good integration quality.

Using standard interfaces that can be verified independently (for example, with VIP from independent sources) for the integration of independently designed IPs limits the potential for bugs to be introduced. The above limitation does not prevent innovation between sub-blocks, but instead discourages any complications from these innovations from being spread across independent blocks that might not have been verified with the same tools. Using customizable extensions of the format can address exceptions, but these extensions are not guaranteed by the standard.

1.2 Purpose

The purpose of the Clock Domain Crossing standard is to:

a) Enable EDA vendors to develop CDC, RDC, and/or glitch analysis tools that meet the specification defined in this standard to generate tool-agnostic collateral
b) Enable IP companies to use their tool(s) of choice to perform CDC, RDC, and/or glitch analysis on their IP and generate said collateral
c) Enable SOC companies to consume collateral generated by different IP vendors from their tool(s) of choice, into the SOC company’s own tool(s) of choice

1.3 Conventions used

The conventions used throughout the document are included here.

1.3.1 Circuit drawing conventions

Use Figure 1 as a guide to understanding the drawings in this LRM.

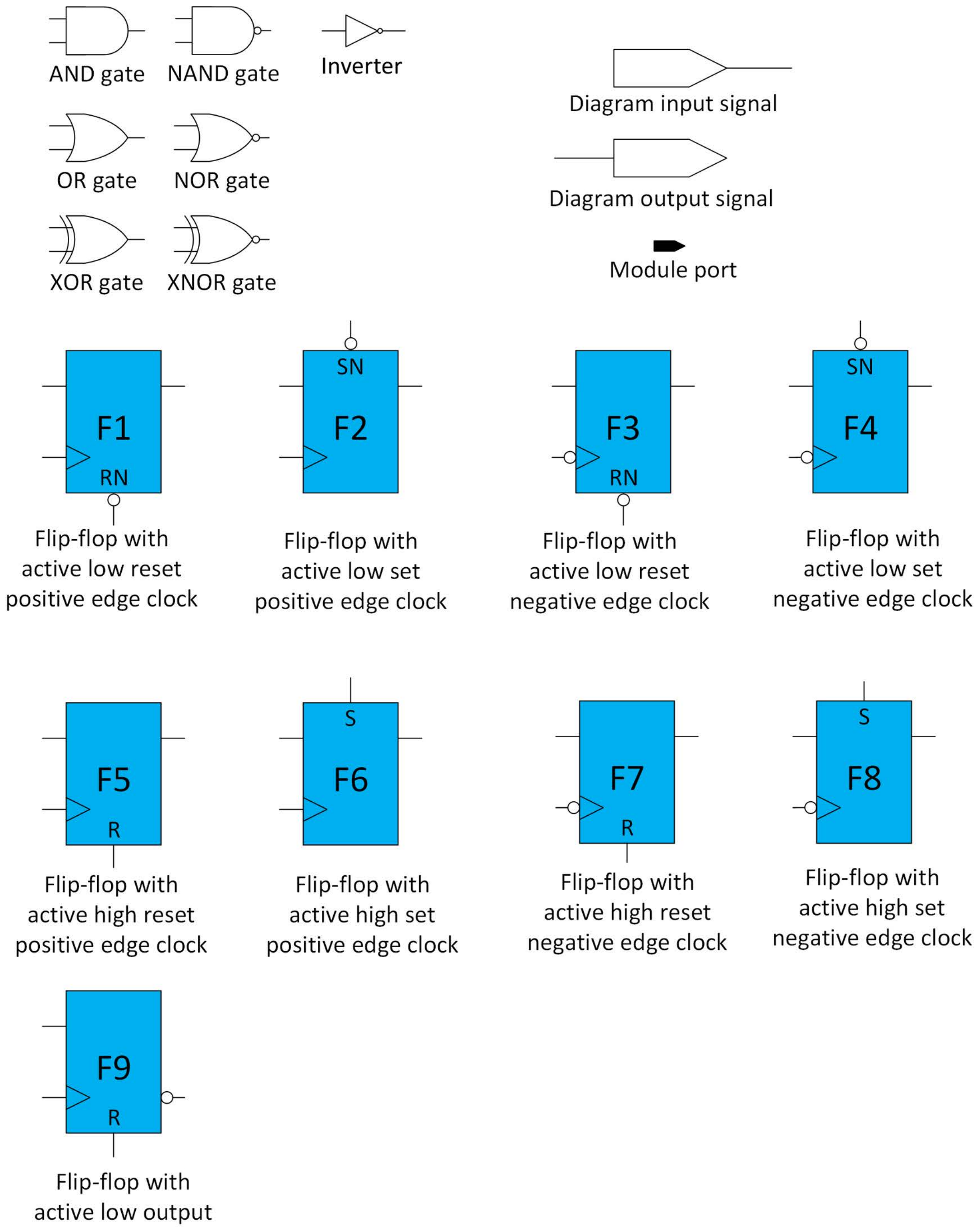

AND gate
NAND gate
Inverter
OR gate
NOR gate
XOR gate
XNOR gate
Diagram input signal
Diagram output signal
Module port
F1
RN
Flip-flop with active low reset positive edge clock
SN
F2
Flip-flop with active low set positive edge clock
F3
RN
Flip-flop with active low reset negative edge clock
SN
F4
Flip-flop with active low set negative edge clock
F5
R
Flip-flop with active high reset positive edge clock
S
F6
Flip-flop with active high set positive edge clock
F7
R
Flip-flop with active high reset negative edge clock
S
F8
Flip-flop with active high set negative edge clock
F9
R
Flip-flop with active low output

Figure 1—Circuit drawing template

1.3.2 Word usage

The word *shall* indicates mandatory requirements strictly to be followed in order to conform to the standard and from which no deviation is permitted (*shall* equals *is required to*).[3],[4]

The word *should* indicates that among several possibilities one is recommended as particularly suitable, without mentioning or excluding others; or that a certain course of action is preferred but not necessarily required (*should* equals *is recommended that*).

The word *may* is used to indicate a course of action permissible within the limits of the standard (*may* equals *is permitted to*).

The word *can* is used for statements of possibility and capability, whether material, physical, or causal (*can* equals *is able to*).

1.4 Use of color in this standard

This standard uses a minimal amount of color to enhance readability. The coloring is not essential and does not affect the accuracy of this standard when viewed in pure black and white. The places where color is used are the following:

— Cross references that are hyperlinked to other portions of this standard are shown in underlined-blue text (hyperlinking works when this standard is viewed interactively as a PDF file).

— Illustrations. (The color can clarify, but it is not essential.)

1.5 Contents of this standard

The organization of the remainder of this standard is as follows:

— Clause 2 provides references to other applicable standards that are assumed or required for this standard.

— Clause 3 defines terms and acronyms used throughout the different specifications contained in this standard.

— Clause 4 lists CDC attributes along with their type, accepted values, whether they are mandatory, and clarifying comments.

— Clause 5 lists RDC attributes and provides scenarios to describe support for an RDC interface at the top level without resynthesizing the RTL of the IP.

— Clause 6 describes the CDC format that is based on the TCL language for CDC specification from an output collateral perspective.

— Clause 7 describes the CDC format that is based on the IP-XACT standard.

— Clause 8 describes SVA Requirements for black box CDC integrity verification.

— Annex A provides an informative bibliography.

— Annex B provides a list of unit testcases for early testing by EDA vendors.

[3] The use of the word *must* is deprecated and cannot be used when stating mandatory requirements; *must* is used only to describe unavoidable situations.

[4] The use of *will* is deprecated and cannot be used when stating mandatory requirements; *will* is only used in statements of fact.

2. References

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments or corrigenda) applies.

IEEE Std 1685™-2022, IEEE Standard for IP-XACT, Standard Structure for Packaging, Integrating, and Reusing IP within Tool Flows

IEEE Std 1800™-2017, IEEE Standard for SystemVerilog Unified Hardware Design, Specification and Verification Language.[5], [6]

[5]The IEEE standards or products referred to in this clause are trademarks of the Institute of Electrical and Electronics Engineers, Inc.

[6]IEEE publications are available from the Institute of Electrical and Electronics Engineers, Inc., 445 Hoes Lane, Piscataway, NJ 08854, USA (https://standards.ieee.org/).

3. Definitions, acronyms, and abbreviations

For the purposes of this document, the following terms and definitions apply. *The Authoritative Dictionary of IEEE Standards Terms* [B1][7] should be referenced for terms not defined in this clause.

3.1 Definitions

Accellera Vendor extensions: Extensions to the IEEE Standard for IP-XACT that capture attributes related to clock and reset to support the CDC Standard.

component: A physical and logical construction that relates inputs to outputs.

glitch: A transient pulse on a signal that is caused by a race between signals in its fan-in. This includes asynchronous data path (for example, combinational logic going into synchronizers or reconvergence of parallel synchronizers) as well as clock and reset trees.

port: A connection on the interface of a SystemVerilog module or very high speed integrated circuit (VHSIC) hardware description language (VHDL) entity.

CDC control signal: A signal that is either tool-inferred or user-specified as a clock domain crossing control to a data path (scalar or vector) as it crosses clock domains.

3.2 Acronyms and abbreviations

CDC	clock domain crossing
Combo	combinational logic
DFT	design for test
EDA	electronic design automation
HDL	hardware description language
inout	bidirectional port
internal_sync	internal synchronizer
IP	intellectual property
IP-XACT	IEEE standard describing an XML metadata schema for documenting IP; used in the development, implementation, and verification of electronic systems
LRM	language reference manual
MTBF	mean time before failure
PWG	proposed working group
RDC	reset domain crossing

[7]The numbers in brackets correspond to those of the bibliography in Annex A.

QoR	quality of results
RTL	register transfer level
SDC	Synopsys Design Constraints
SOC	system on chip or products that consist of various IPs and glue-logic
SVA	SystemVerilog Assertions
STA	static timing analysis
SWG	sub-workgroup
TCL	Tool Command Language
VIP	validation IP or test environment used to test aspects of an IP
WG	working group
XML	eXtensible Markup Language

NOTE—CDC WG refers to CDC-, RDC-, and glitch-related (asynchronous data crossing) checks necessary for correct functionality of clock, reset, and data glitch (associated with clock and reset crossings).

4. CDC Attributes

The CDC attributes are expressed in case-sensitive TCL for the CDC tool-level format, but you have the option to translate this Tcl to IP-XACT for packaging the IP (see Clause 6 and Clause 7).

This version of the LRM includes case-sensitive TCL commands to support EDA vendors with tool development. (Also see Clause 6.) The case-sensitive TCL captures data required from input, output, and verification collateral in a human-readable and machine-parsable format to provide accurate CDC, RDC, and glitch structural analysis by any EDA tool. The CDC attributes format supports customizable extensions for complex user conditions. In addition, these attributes are applicable to other design styles (e.g., FPGA and Analog) that follow similar standards.

The CDC attributes facilitate a correct CDC and RDC integration of blocks in an encompassing design and address a specific set of known industry standard interfaces. The CDC standard identifies both the CDC and RDC schemes the attributes support and those schemes the defined set of attributes is not guaranteed to support.

Each table below is associated with an attribute domain (i.e., module, parameter, port, and so forth), and within each table, the applicable attributes are listed along with their type, accepted values, whether they are mandatory, and clarifying comments. (Also see Clause 5 for additional information about RDC attributes.) Use the drawings and accompanying case-sensitive TCL examples in the sections following the tables to understand clock relationships and various crossings and failure modes. These sections include the following:

a) Module attribute
b) Parameter attributes
c) Port attributes
d) async_reset
e) Modeling abstracted blocks
f) Clock definitions
g) Clock relationships
h) Failure modes

NOTE—Support for an `-update` or `-add` argument will be considered in the future for incrementally building commands. In the case of conflicting commands, the final command is recognized.

Table 1 lists details of the module domain attribute: name.

Table 1—CDC attributes: module domain

Domain	Attribute	Type	Values	Conditions of usage	Comments
module	**name**	string	{module name}	Mandatory	See also: 4.1

Table 2 lists details of the parameter domain attributes.

Table 2—CDC attributes: parameter domain

Domain	Attribute	Type	Values	Conditions of usage	Comments
parameter	**name**	string	{parameter name}	Mandatory	See also: 4.2
parameter	**value**	range-list	{values}	Optional	See also: 4.2
parameter	**type**	defined set	{string, Boolean*, number (hex, decimal, oct, binary)} * Boolean values 0/1 and true/false (case independent) are accepted.	Optional	See also: 4.2
parameter	**ignore**	Boolean	{0 or 1, true or false} (case independent)	Optional	See also: 4.2

Table 3 lists details of the port domain attributes.

Table 3—CDC Attributes: port domain

Domain	Attribute	Type	Values	Conditions of usage	Comments
port	**name**	string	{port name}	Mandatory	See also: 4.3.1
port	**direction**	defined set	{input, output, inout}	Mandatory	See also: 4.3.1
port	**type**	defined set	{data, clock, virtual_clock, async_reset, cdc_control, rdc_control, virtual_reset}	Mandatory Mandatory applies to async reset only.	For async resets, type is async_reset; for sync resets, type is data. Type virtual_clock defines a virtual clock port that does not match an actual port in the module. See also: 4.3.1, 4.3.4, 4.4, and Clause 5

Table 3—CDC Attributes: port domain *(continued)*

Domain	Attribute	Type	Values	Conditions of usage	Comments
port	**logic**	defined set	{combo, inverter, glitch_free_combo, internal_sync}	Optional	Without this attribute, the assumption is that the port directly reaches a sequential element. See also: 4.3.10
port	**cdc_data_from_clock**	string	{clock name}	Conditional Mandatory for ports of type cdc_control.	Supports only one clock. See the definition below. See also: 4.5, Figure 15

Definition: Associates a cdc_control to a clock domain where the data is coming from.

Example:

```
cdc_set_port sync_valid
    -direction input
    -type cdc_control
    -logic internal_sync
    -associated_from_clocks VCLK_tx_clock
    -cdc_data_from_clock VCLK_tx_clock
    -associated_to_clocks rx_clock

cdc_set_port sync_valid
    -direction input
    -type cdc_control
    -associated_from_clocks rx_clock
    -cdc_data_from_clock VCLK_tx_clock
    -associated_to_clocks rx_clock
```

Domain	Attribute	Type	Values	Conditions of usage	Comments
port	**associated_from_clocks**	space-separated list	{clock-names}	Conditional This attribute is mandatory for an input port unless the input port is not received by any clock domain.	See also: 4.3.2 and 4.5

Table 3—CDC Attributes: port domain *(continued)*

Domain	Attribute	Type	Values	Conditions of usage	Comments
port	**associated_to_clocks**	space-separated list	{clock-names}	Conditional This attribute is mandatory for an output port unless the output port is not generated from any clock domain.	For default values, see 4.3.3. See also 4.5 and 5.1.4
port	**associated_inputs**	space-separated list	{ports}	Optional	See the definition below. See also: 4.5
Definition: Used for feedthrough configuration where one or more inputs reach one or more outputs; applicable to outputs. Used also for the CDC control signal configuration to associate a cdc_control to a vector signal; applicable to inputs if considering a CDC control signal input.					
port	**associated_outputs**	space-separated list	{ports}	Optional	See the definition below. See also: 4.3.9
Definition: Used for a feedthrough configuration where one or more outputs trace from one or more inputs; applicable to inputs. The cdc_control signal configuration might be useful.					
port	**cdc_control**	space-separated list	{associated-ports}	Optional	See also: 4.5
port	**cdc_control_setup**	integer	value	Conditional	See the definition below. See also: 4.5
Definition: Sets a constant value on a port with the cdc_control attribute to define the setup margin between the cdc_control port and its associated_inputs data port. The value of this attribute is the number of destination clock cycles the associated_inputs data port must be stable before the cdc_control port enables the data crossing. If the value is 3, associated_inputs data port must be stable for 3 destination clock cycles before cdc_control enables data crossing. If the value is -2, associated_inputs data port can toggle at max up to 2 destination clock cycles after cdc_control enables data crossing.					
port	**cdc_control_hold**	integer	value	Conditional	See the definition below. See also: 4.5
Definition: Sets a constant value on a port with the cdc_control attribute to define the hold margin between the cdc_control port and its associated_inputs data port. The value of this attribute is the number of destination clock cycles the associated_inputs data port must be stable after the cdc_control port disables the data crossing. If the value is 3, associated_inputs data port must be stable for 3 destination clock cycles after cdc_control disables data crossing. If the value is -2, associated_inputs data port can change at most 2 destination clock cycles before cdc_control disables data crossing.					

Table 3—CDC Attributes: port domain *(continued)*

Domain	Attribute	Type	Values	Conditions of usage	Comments
port	**sampling_edge**	defined set	{pos, neg}	Conditional This attribute is mandatory if the IP provider requires minimum pulse width assertion at a single bit input that is received by a synchronizer in the IP. In such cases, the input signal should be also described by the attribute "-logic internal_sync".	See the definition below. See also: 4.5
Definition: Paired with cdc_control port type or a single-bit data port type that requires synchronization to define the sampling edge of the first receiving flop of the destination domain. The string "pos" is attributed to the positive edge of the destination clock sampling the source data. The string "neg" is attributed to the negative edge of the destination clock sampling the source data.					
port	**polarity**	defined set	{high, low, low_high}	Conditional This attribute is mandatory for async reset, cdc_control, and rdc_control; otherwise, it is optional.	For cdc_control and rdc_control, "low_high" is not allowed.
port	**ignore**	defined set	{blocked, hanging}	Optional	See also: 4.3.6
port	**cdc_static**	space-separated list	{clock-names}	Optional	See also: 4.3.7
port	**constant**	space-separated list	{binary, hex, and of any length}	Optional	See the definition below. See also: 4.3.8
Definition: Sets a constant value on a net, pin, or port for the current analysis. From the CDC model perspective, it is applicable to a port only. It is relevant for all signal types (data, control, etc.).					

Table 3—CDC Attributes: port domain *(continued)*

Domain	Attribute	Type	Values	Conditions of usage	Comments
port	**gray_coded**	Boolean	{0 or 1, true or false} Default: 0 or false (case independent)	Optional	See the definition below. Deferred to LRM v0.5
Definition: Implies mutually exclusiveness signals that are part of a bus corresponding to Gray coding (only 1 bit changing sequentially). gray_coded ports are used to eliminate convergence and glitch violations inside an IP. If a user has specified these on input ports, then an assertion is generated to check for non-gray coded transitions. There isn't any validation other than assertions that can be done on this constraint.					
port	**clock_period**	string	{clock period}	Conditional	See definition below.
clock_period is used to identify relative clock periods for recognizing a fast to slow CDC and where assertions could be created for detecting if violations could occur and invalidate the CDC structure.					
port	**associated_from_reset**	space-separated list	{reset-names}	Optional	Defines the driver reset of a port See also: 4.3.5
port	**associated_to_reset**	space-separated list	{reset-names}	Optional	`Defines the receiver reset of a port` See also: 4.3.5
port	**rdc_control**	space-separated list	{associated-port}	Conditional	See also: 5.2
port	**rdc_data_from_reset**	space-separated list	{reset-names}	Optional	Used with rdc_control. Defines the source reset of the RDC; i.e., the start point’s reset. This is the source reset being blocked by -rdc_control. See also: Clause 5
port	**rdc_data_to_reset**	space-separated list	{reset-names}	Optional	Used with rdc_control. Defines the desti-nation reset of the RDC; i.e., the end point’s reset. See also: Clause 5

Table 3—CDC Attributes: port domain *(continued)*

Domain	Attribute	Type	Values	Conditions of usage	Comments
port	**rdc_data_to_clock**	space-separated list	{clock-names}	Optional	Used with rdc_control. Defines the destination clock of the RDC end point's clock. This is the destination clock being gated by -rdc_control. See also: Clause 5
port	**rdc_clock_gate_location**	defined set	{external or internal}	Optional	Used with rdc_control. Defines the location of a clock gate; i.e., internal versus external. See also: Clause 5

Table 4 lists details of the tool domain attributes.

Table 4—CDC Attributes: tool domain

Domain	Attribute	Type	Values	Conditions of usage	Comments
tool	**name**	string	{tool name}	Conditional Mandatory if it is EDA-tool-generated. If no EDA tool name is captured, it implies hand crafted by user.	
tool	**version**	string	{tool version}	Conditional Mandatory if it is EDA-tool-generated. N/A if user generated.	

Table 5 lists details of the design domain attributes.

Table 5—CDC Attributes: design domain

Domain	Attribute	Type	Values	Conditions of usage	Comments
design	**version**	string	{design milestone}	Optional	
design	**date**	string	{collateral generation date}	Mandatory	
design	**username**	string	{user/tool that generated the collateral}	Optional	
design	**description**	string	{description}	Optional	

Table 6 lists details of the `set_cdc_clock_group` domain attributes.

Table 6—CDC Attributes: cdc_set_clock_group domain

Domain	Attribute	Type	Values	Conditions of usage	Comments
cdc_set_clock_group					See the definition below. See also 4.7

Definition: A set of clocks that are synchronous to each other in a design.
 	
Example:
`cdc_set_clock_group`
`    -name small_domain`
`    -clocks {ck}`
 	
`cdc_set_clock_group`
`    -name large_domain`
`    -clocks {gck0;gck1}`					

	clocks	space-separated list	{clock-names}	Mandatory	
	name	string	{group-name}	Optional	

Table 7 lists details of the `set_reset_group` domain attribute.

Table 7—CDC Attributes: set_reset_group domain

Domain	Attribute	Type	Values	Conditions of usage	Comments
set_reset_group					See the definition below. `See also` 5.2.3

Defines the reset relation. There are no RDC violations between resets that appear together in the same reset group.

Example:
```
set_reset_group
    -name reset_domain_1
    -reset {virtual_reset_a; rstb}
```

4.1 Module attribute

The attributes include one module attribute. Refer to the following case-sensitive TCL and Figure 2, which depict the `mod0` module.

```
cdc_set_module mod0              Name of the module
```

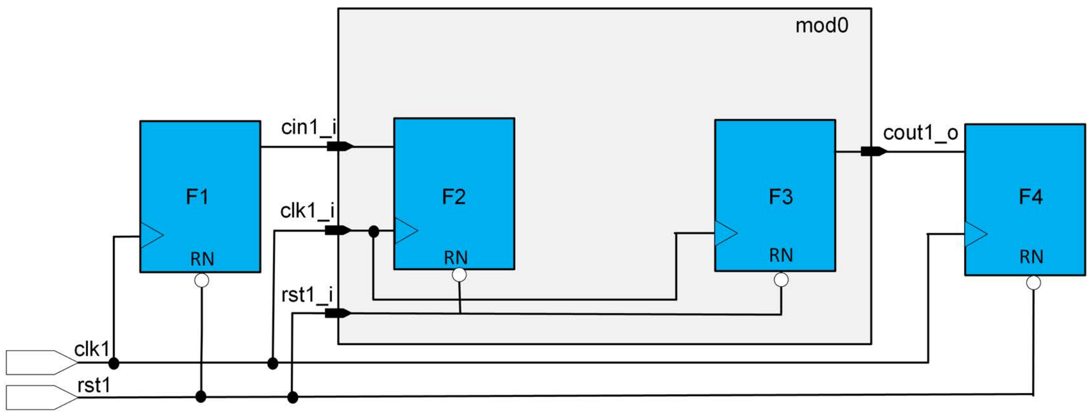


Figure 2—Module attribute

4.2 Parameter attributes

The `parameter` command is used to define parameters within the scope of the module, e.g.,

```
cdc_set_param    \
    -name PARAM1 \               Name of parameter
    -type int    \               Type of parameter
    -value 0;                    Value of parameter
```

Table 8 lists the attributes that will be used with the `parameter` command.

Table 8—Parameter attributes

Attribute	Values	Definition
-name	String	Defines the name of the parameter. Is Mandatory.
-type	Integer/string/Boolean Boolean values 0/1 and true/false (case independent) are accepted.	Specifies type of the value stored by the parameter

Table 8—Parameter attributes *(continued)*

Attribute	Values	Definition
-value	Range-list	Specifies the value of the parameter
-ignore	N/A	Tells that the parameter should be ignored. Any parameter that is ignored but still referred to in the model in following commands is a failure.

Table 9 contains usage examples for the `parameter` command with the cdc_set_param command.

Table 9—Parameter usage

Number	Scenario	Example command

1	Sample integer type parameter with value 32	`cdc_set_param`
		`    -name PARAM1`
		`    -type int`
		`    -value 32`

2	Sample string type parameter with value DEFAULT_CASE	`cdc_set_param`
		`    -name CASE_VAR`
		`    -type string`
		`    -value DEFAULT_CASE`

3	Sample Boolean type parameter with value “false”
	Boolean values 0/1 and true/false (case independent) are accepted.
		`cdc_set_param`
		`    -name SELECT`
		`    -type boolean`
		`    -value false`
		
		or
		
		`cdc_set_param`
		`    -name SELECT`
		`    -type boolean`
		`    -value 0`

Table 9—Parameter usage

Number	Scenario	Example command

4	A bus named DATA 7 down to 0 defined using the parameter for MSB and LSB.	`cdc_set_param`
		`    -name MSB`
		`    -type int`
		`    -value 7`
		
		`cdc_set_param`
		`    -name LSB`
		`    -type int`
		`    -value 0`
		
		`cdc_set_port DATA[7:0]`
		`    -type data`
		
		or
		
		`cdc_set_port DATA[MSB:0]`
		`    -type data`
		
		or
		
		`cdc_set_port DATA[7:LSB]`
		`    -type data`
		
		or
		
		`cdc_set_port DATA[MSB:LSB]`
		`    -type data`

5	Individual bits and slice of a bus named DATA 7 down to 0 defined using the parameter for MSB and LSB.
	Also using expressions with parameters.
		`cdc_set_param`
		`    -name MSB`
		`    -type int`
		`    -value 7`
		
		`cdc_set_param`
		`    -name LSB`
		`    -type int`
		`    -value 0`
		
		Only the 8th bit
		`cdc_set_port DATA[MSB]`
		`    -type data`
		Only the first four MSB bits
		`cdc_set_port DATA[MSB:MSB-3]`
		`    -type data`
		
		Only the last 2 LSB bits
		`cdc_set_port DATA[LSB+1:LSB]`
		`    -type data`
		
		3-bit slice from the middle of the bus
		`cdc_set_port DATA[MSB-2:LSB+3]`
		`    -type data`

6	Example of usage of a parameter for a port P1 to be tied to a parameterized constant	`cdc_set_param`
		`    -name SEL_VAL`
		`    -type boolean`
		`    -value true`
		
		`cdc_set_port P1`
		`    -constant SEL_VAL`

4.3 Port attributes

Several port attributes are detailed in this section.

4.3.1 Port attributes: name, direction, and type

Refer to the following case-sensitive TCL and Figure 3, which depict the `name`, `direction`, and `type` attributes. Also see 4.4 for additional information about the `type` attribute. If the name does not include a range, it applies to all bits of the port.

```
cdc_set_port cin1_i                        \    Name of port
    -direction input                       \    Direction of port
    -type data                             \    Type of port
    -associated_from_clocks virtual_clk \    External driver's clock domain
    -associated_to_clocks clk1_i                Internal receiver clock
```

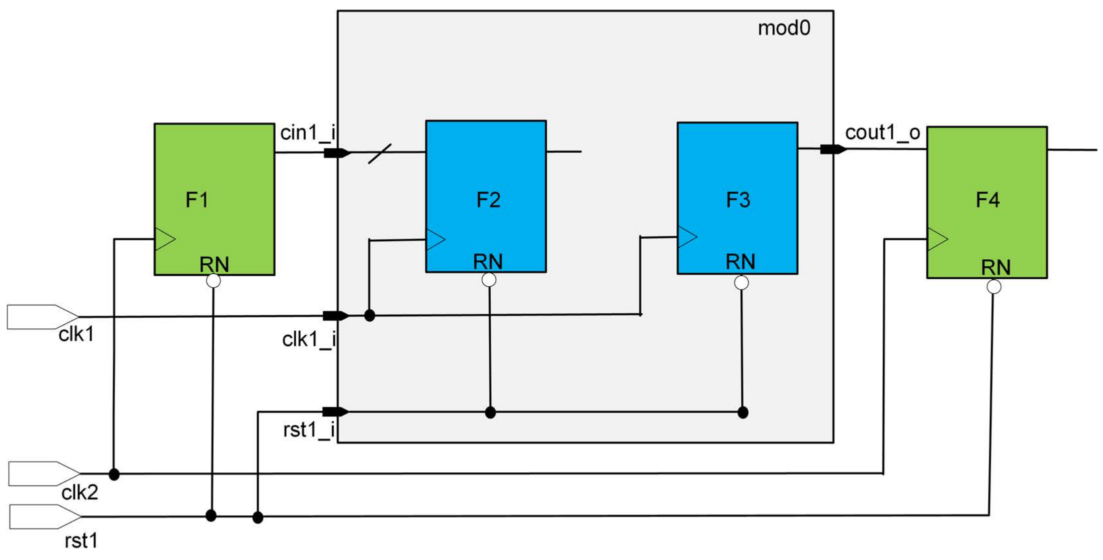


Figure 3—Port attributes: name, direction, type

For clock attributes see 4.3.2, 4.3.3, 4.3.4. For a description of the related testcase, refer to Annex B.

4.3.2 Port attribute: associated_from_clocks

The following case-sensitive TCL and Figure 4 depict the `associated_from_clocks` attribute.

For an input port, the `associated_from_clocks` attribute is part of the optional external model describing the driver of the input. For an output port, it is part of the mandatory internal model describing the driver of the output.

```
cdc_set_port cin1_i                \   Name of port
    -direction input               \   Direction of port
    -type data                     \   Type of port
    -associated_from_clocks clk1_i \   External driver’s clock domain
    -associated_to_clocks clk1_i       Internal receiver clock
```

```
cdc_set_port cout1_o  \                Name of port
    -direction output \                Direction of port
    -type data        \                Type of port
    -associated_from_clocks clk1_i     External driver’s clock domain
```

Table 10 provides a description of the usage of `associated_to_clocks` and `associated_from_clocks` for output ports.

Table 10—Associated clocks usage for output ports

For output port	**-associated_to_clocks defined e.g., -associated_to_clocks clk2**	**-associated_to_clocks Not defined**
associated_from_clocks **defined** e.g., `-associated_from_clocks clk1`	If clk1 and clk2 are asynchronous to each other, an appropriate synchronization scheme is expected in the receiving circuit to avoid a CDC violation.	The receiving clock is found in the actual encompassing design. If it is asynchronous to clk1, an appropriate synchronization scheme is expected in the receiving circuit to avoid a CDC violation.

For ports being associated to a virtual clock refer to 4.3.4.

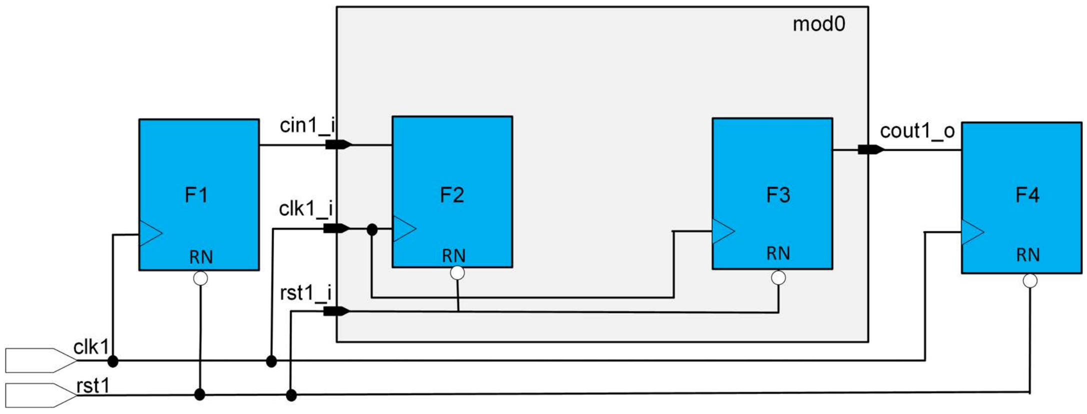


Figure 4—Port attribute: associated_from_clocks

4.3.3 Port attributes: associated_to_clocks

For an input port, the `associated_to_clocks` attribute is part of the mandatory internal model describing the property of the input. For an output port, it is part of the optional external model describing the requirement for the receiver of the output.

Table 11 provides a description of the usage of `associated_to_clocks` and `associated_from_clocks` for input ports.

Table 11—Associated clocks usage for input ports

For input port	-associated_from_clocks defined e.g., -associated_from_clocks clk2	-associated_from_clocks Not defined
`associated_to_clocks` **defined** e.g., `-associated_to_clocks clk1`	If clk1 and clk2 are asynchronous to each other, an appropriate synchronization scheme is expected in the receiving circuit to avoid a CDC violation.	The input port's driving clock is found in the actual encompassing design. If it is asynchronous to `clk1`, an appropriate synchronization scheme is expected in the CDC crossing path to avoid a CDC violation.

4.3.4 Defining a virtual clock

In the CDC LRM 0.3, virtual clocks must be explicitly declared by defining a port with the type `virtual_clock`. This is a virtual port that does not match an actual port in the module. For example, `vclk` used in Figure 5 to define the `-associated_from_clocks` of port `in1_i` must be declared with type `virtual_clock` as shown below.

```
cdc_set_port clk1_i   \                Name of port
    -direction input \                 Direction of port
    -type clock                        Type of port
```

```
cdc_set_port vclk     \                Name of port
    -direction input \                 Direction of port
    -type virtual_clock                Type of port
```

In the following command, `in1_i` applies to all bits when a range is not specified.

```
cdc_set_port in1_i                 \  Name of port
    -direction input               \  Direction of port
    -type data                     \  Type of port
    -associated_from_clocks vclk \  External driver's clock domain
    -associated_to_clocks clk1_i \  Internal receiver clock
    -logic combo                      Port has combo logic on path to
                                        register
```

```
cdc_set_port rst_n_i  \              Name of port
    -direction input  \              Direction of port
    -type async_reset \              Type of port
    -associated_to_clocks clk1_i     Internal receiver clock
```

For a description of the related testcase, refer to Annex B.

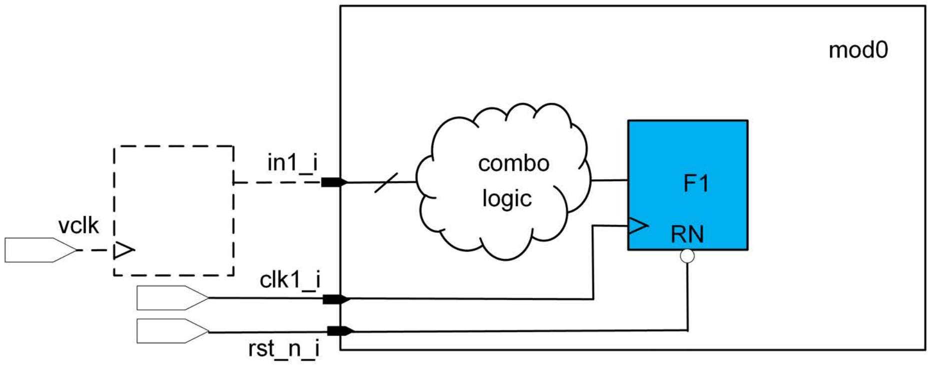


Figure 5—Specifying a virtual clock

4.3.5 Port attribute: associated_from_reset and associated_to_reset

The following case-sensitive TCL and Figure 6 depict the optional `associated_from_reset` and `associated_to_reset` attributes.

```
cdc_set_port cin1_i              \     Name of port
    -direction input             \     Direction of port
    -type data                   \     Type of port
    -associated_to_clocks clk1_i \     Internal receiver clock
    -associated_to_reset rst1_i        Internal receiver reset
```

```
cdc_set_port cout1_o               \   Name of port
    -direction output              \   Direction of port
    -type data                     \   Type of port
    -associated_from_clocks clk1_i \   External driver clock domain
    -associated_from_reset rst1_i      External driver reset
```

For a description of the related testcase, refer to Annex B.

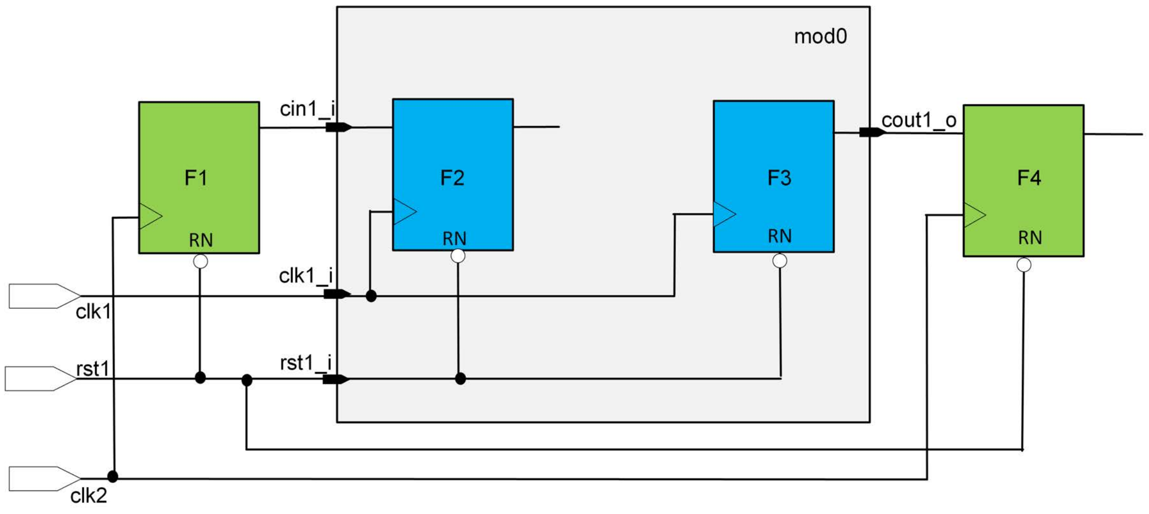


Figure 6—Port attribute: associated_from_reset and associated_to_reset

4.3.6 Port attribute: ignore

The `-ignore` attribute is used to indicate an interface port is not being analyzed for a clock domain crossing that is `hanging` or `blocked`. For example, the `hanging` value is used when an input port is dangling as shown in Figure 7.

```
cdc_set_port in2_i  \               Name of port
    -ignore hanging                 No analysis for CDC that is hanging
```

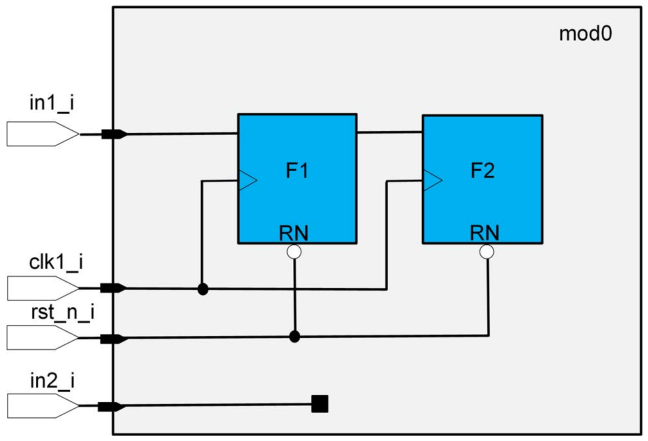


Figure 7—Ignore: hanging

The `blocked` value is used when the propagation of an interface port is blocked as shown in Figure 8. `in1_i` in Figure 8 is blocked due to a constant constraint at `cdc_set_port mode`. RTL tie-off can also cause an interface port to be `blocked`, and therefore, ignored. The `-ignore` attribute is generated by the EDA tool during CDC analysis. Both the RTL and input constraints are used by the EDA tool to determine when an interface port is not being considered for CDC analysis.

```
cdc_set_port mode \               Name of port
    -constant 0                   Constant value is 0
```

```
cdc_set_port in1_i \              Name of port
    -ignore blocked               No analysis for CDC that is blocked
```

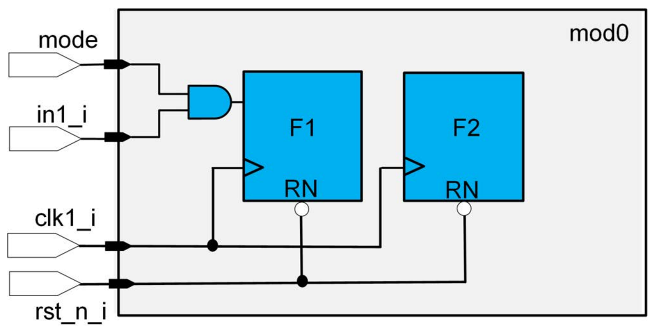


Figure 8—Ignore: blocked

4.3.7 Port attribute: cdc_static

This section discusses the need and usage for the `cdc_static` attribute.

4.3.7.1 Need for the “cdc_static” attribute

In some cases, when the data is crossing the domain boundaries, we may not always want to add synchronization schemes in between to ensure less area overhead. This requires some pseudo static behavior on the data that is crossing the domain boundary to avoid any metastable behavior. For example, in Figure 9, module `mod1` only works on `clk1` and `mod2` only works on `clk2`. `mod0` has a clock gating block on the clock path driving `mod2` and a data gating block on the data path for the incoming port `d1`, `d2`. If it is guaranteed by the design that `mod0`’s `clk2` shall always be gated before the driver for port `d1`, and port `d2` changes (i.e., port `a3` and port `a4` of `mod1`), the `cdc_static` attribute can be applied to port `d1` and port `d2` during `mod0`’s CDC analysis. Hence, CDC analysis shall not be done on these primary interface ports of `mod0`. It is important to have all `cdc_static` attributes validated in the assertion flow to ensure the functionality of the design. `mod0`’s internal nets `DATACOND` and `CLKCOND` illustrated in Figure 9 are not supported in the CDC v0.5 LRM because the current LRM focuses on capturing the interface information. However, `mod0`’s internal nets `DATACOND` and `CLKCOND` might be used in a future LRM, where a regular expression would be added to describe the `cdc_static`’s condition observed on port `d1` and port `d2`.

For a description of the related testcase, refer to Annex B.

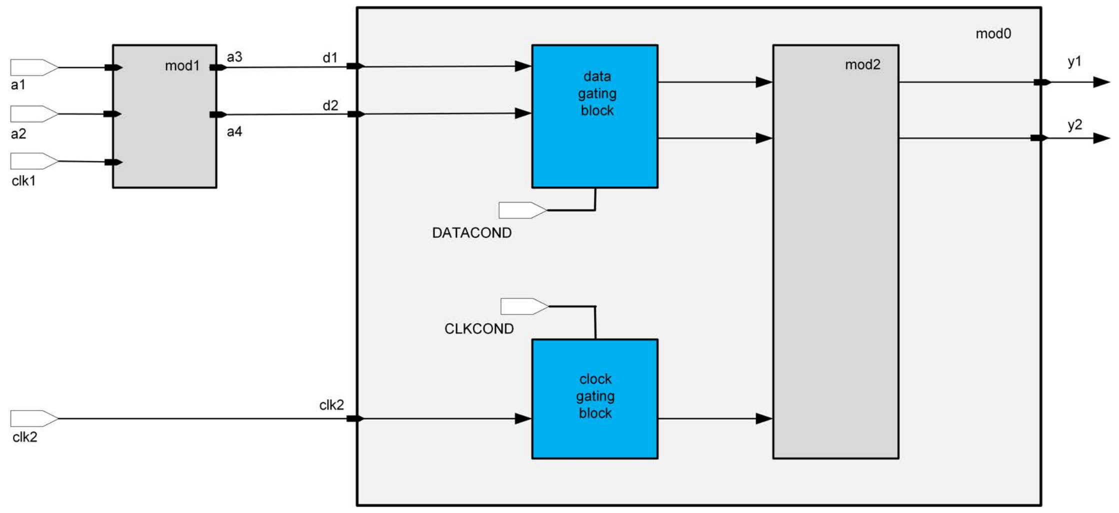


Figure 9—Example showing the need to describe a pseudo static behavior on the data crossing domains

4.3.7.2 Usage for the “cdc_static” attribute

The `cdc_static` attribute associates a data port with a clock or multiple clocks to indicate when the data is changing. It is expected that the clock or clocks will be gated; hence, no CDC verification is required.

Generic Usage:

```
cdc_set_port <name of the port> \
    -direction <input/output>   \
    -cdc_static <clock or list of clocks that will be gated>
```

Usage based on `mod0` of Figure 9:

```
cdc_set_port d1       \           Name of port
    -direction input \            Direction of port
    -cdc_static clk2              Gated clock
```

```
cdc_set_port d2       \           Name of port
    -direction input \            Direction of port
    -cdc_static clk2              Gated clock
```

Note: In contrast to a constant signal (see 4.3.8) the value of a static signal is not known.

4.3.8 Port attribute: constant

This section discusses the need and usage for the `constant` attribute.

4.3.8.1 Need for the “constant” attribute

Figure 10 below shows two different flops in two asynchronous clock domains. For example (Figure 10), when the design for test (DFT) select is 0, then the functional reset path is activated, which clearly indicates a domain crossing violation on one of the reset paths because the same reset signal is consumed in two asynchronous clock domains. While in the case of a DFT select set to 1, the DFT flow shall ensure that functionally, there are no domain crossings on clock or reset paths by making sure that the clock and reset paths are separately controllable. Based on this, we can safely mark the DFT bypass reset path and the DFT select path as constant so that they are ignored in the CDC analysis.

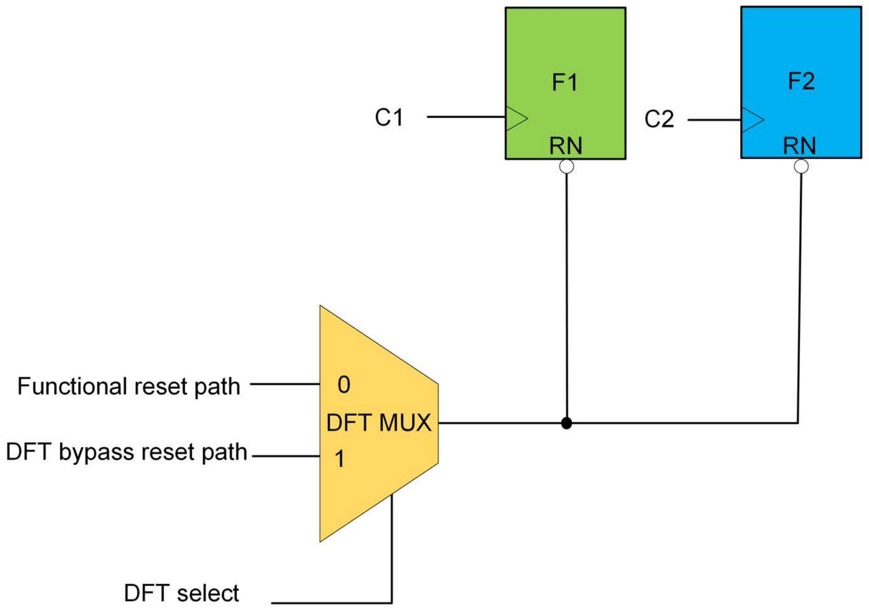


Figure 10—Example of the DFT structure usage in a digital design

4.3.8.2 Usage for the “constant” attribute

Figure 11 shows the design structure with associated ports. `clk1`, `clk2`, and `rstn` are functional ports while `dft_bypass_rstn` and `dft_bypass_rstn_select` are the DFT-specific ports. For the functional CDC analysis, these ports (`dft_bypass_rstn` and `dft_bypass_rstn_select`) can be declared as constants using the `constant` attribute.

Generic usage:

```
cdc_set_port <name of the port> \
    -direction <input/output>   \
    -constant <constant value>
```

Example:

```
cdc_set_port dft_bypass_rstn \           Name of port
    -direction input         \           Direction of port
    -constant 0                          Constant value is 0
```

```
cdc_set_port dft_bypass_rstn_select \    Name of port
    -direction input                \    Direction of port
    -constant 0                          Constant value is 0
```

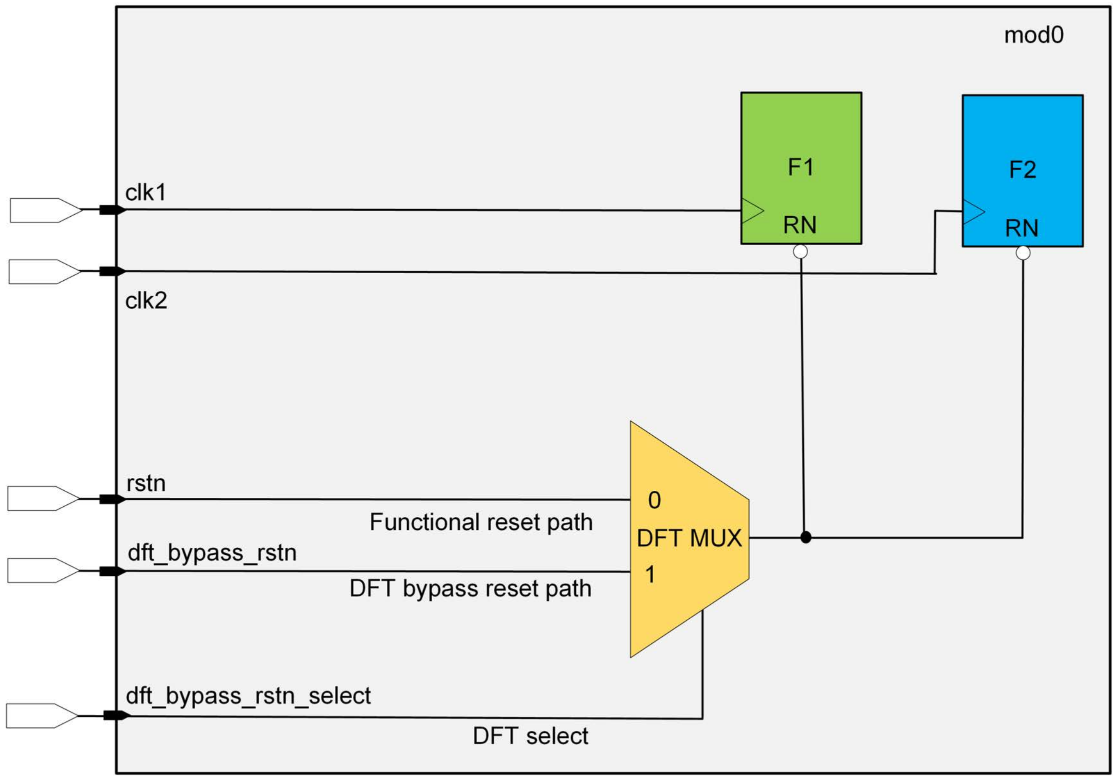


Figure 11—Example of a digital design explaining usage of the “constant” attribute

While using the `constant` attribute, only the name and direction of the ports are mandatory attributes. Other attributes like `type`, `associated_to_clocks`, etc. are not required to be defined.

For a description of the related testcase, refer to Annex B.

4.3.9 Port attribute: associated_outputs

This section discusses the need and usage for the `associated_outputs` attribute. Figure 12 depicts feedthrough logic. The associated case-sensitive TCL includes the optional attribute `associated_outputs` for the feedthrough output.

```
cdc_set_port ftin    \          Name of port
    -type data       \          Type of port
    -direction input \          Direction of port
    -associated_outputs ftout   Output port controlled by ftout
```

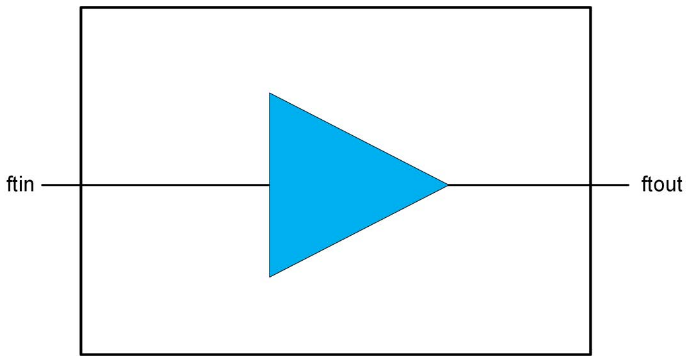


Figure 12—Feedthrough logic

4.3.10 Port attribute: logic

This section discusses the need and usage for the `logic` attribute. The port attribute logic describes the internal elements in the fan-out of an input port or in the fan-in of an output port. The example below includes the values `internal_sync` and `glitch_free_combo`.

The `logic` values include the following:

— `internal_sync` indicates a multi-flop synchronizer on the path through the port.

— `glitch_free_combo` indicates that the path through the port contains combinatorial logic that does not generate glitches in any of the valid operation modes or at reset assertion. Whereas, the attribute value `combo` describes a potentially glitching combinatorial logic.

— `inverter`: indicates the signal is inverted, and accordingly, polarity will be considered inverted.

```
cdc_set_port in1_i                \  Name of port
    -direction input              \  Direction of port
    -type data                    \  Type of port
    -associated_to_clocks clk1_i  \  Internal receiver clock
    -associated_to_reset rstn_i   \  Internal receiver reset
    -logic internal_sync             Internal synchronizer
```

```
cdc_set_port out1_0                 \   Name of port
    -direction output               \   Direction of port
    -type data                      \   Type of port
    -associated_from_clocks clk1_i  \   External driver clock domain
    -associated_to_reset rstn_i     \   The receiver reset of a port
    -logic glitch_free_combo            The path through the port contains
                                          combinatorial logic that does not
                                          generate glitches
```

For a description of the related testcase, refer to Annex B.

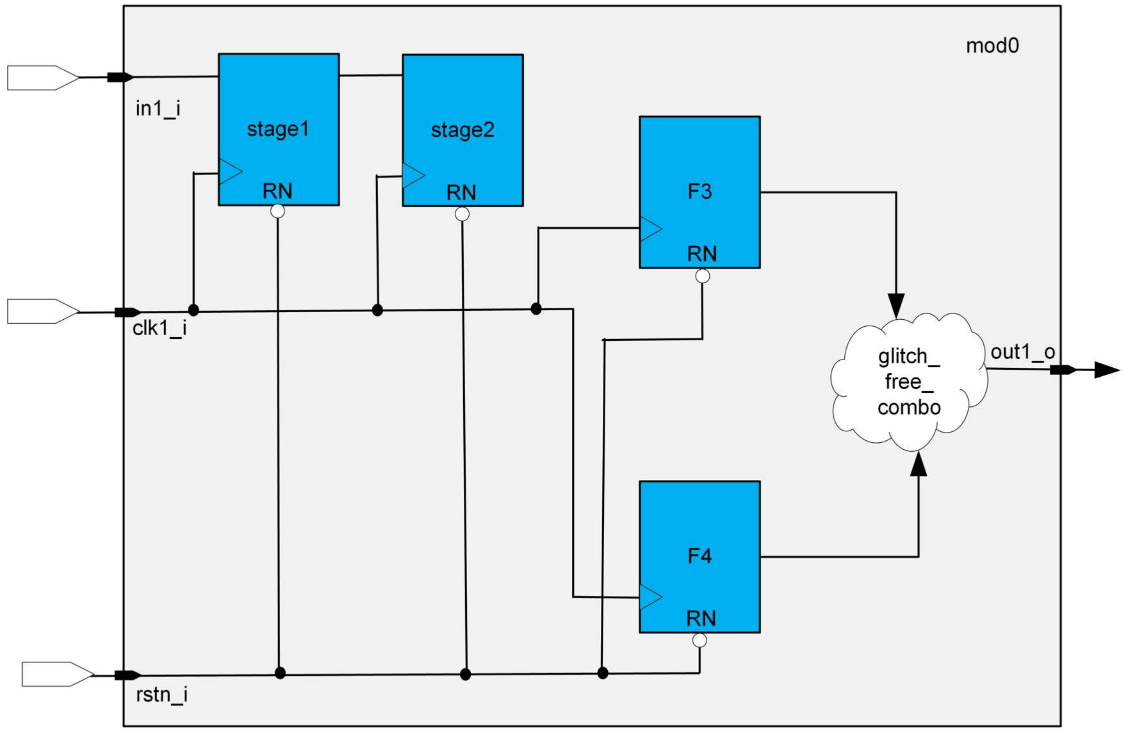


Figure 13—Internal synchronizer and internal combinatorial logic on paths through ports

4.4 async_reset

This section discusses the need and usage for the `async_reset` attribute. To define an asynchronous reset, `port` should include the `-type` attribute with the value `async_reset`. All synchronous resets will be considered data ports.

If the `port` command's `-type` attribute has the value `async_reset`, the attributes in Table 12 will be applicable and the rest of the attributes of the `port` command can be ignored.

Sample command:

```
cdc_set_port <port_name>           \     Name of port
    -direction input               \     Direction of port
    -type async_reset              \     Type of port
    -polarity low                  \     Active low asynchronous reset
    -associated_from_clocks TXCLK \     External driver clock domain
    -associated_to_clocks RX_CLK  \     Internal receiver clock
    -logic inverter                \     The signal is inverted
    -ignore                              No checks required for port
```

Table 12—Supported async_reset attributes

Attribute	**Values**	**Definition**
-name	<port name>	Defines the name of a physical port when used with `-type async_reset`. This attribute is mandatory for `-type async_reset`.
-polarity	low/high/low_high	`-polarity low signifies active_low asynchronous reset` `-polarity high` signifies active_high asynchronous reset `-polarity low_high` signifies the reset is used by some flops as active_low and by some flops as active_high This attribute is mandatory for `async_reset`.
-direction	input/output/inout	`input` signifies that the port is an input port. `output` signifies that the port is an output port. `inout` signifies that the port is an inout port. This attribute is mandatory.

Table 12—Supported async_reset attributes

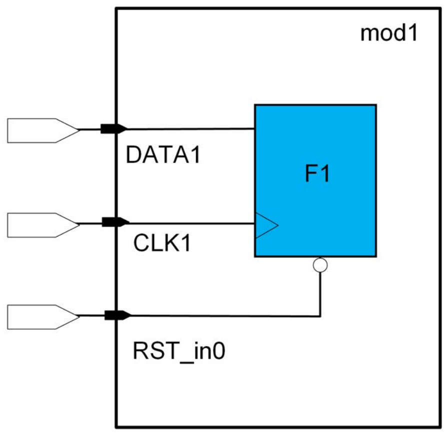
`Attribute`	**Values**	**Definition**
`-associated_from_clocks`		
	`<clock_name>`	Specifies the driver clock of a reset signal. This attribute is conditional. It shall be mandatory for a reset output port unless the port is not generated from any clock domain. The reset output is considered as deasserting with regard to the `associated_from_clock`. If both `associated_from_clocks` and `associated_to_clocks` are specified, the clock domain for both clocks should match.
`-associated_to_clocks`		
	`<clock_name>`	Specifies the receiver clock. This attribute is conditional. It shall be mandatory for a reset input port unless the port does not go to any clock domain in the IP. Without `logic internal_sync` in the reset input port attribute, if both `associated_from_clocks` and `associated_to_clocks` are specified, the clock domain for both clocks should match.
`-logic`	`{combo, inverter, internal_sync}`	`combo`: Signifies that there is combo logic just inside the port. `inverter`: Signifies the signal is inverted, and accordingly, polarity will be considered inverted. `internal_sync`: For an `async_reset` type of input port, this signifies that the port is feeding a reset synchronizer. For an `async_reset` type of output port, `internal_sync` signifies that the signal is generated by a reset synchronizer. This attribute is optional. The attribute shall reflect what is present in the design: — `combo`: the default is *combo not present* — `inverter`: the default is *inverter not present* — `internal_sync`: the default is *internal sync not present*
`-ignore`	N/A	No checks are required for this port This attribute is optional. For high quality of CDC/RDC checks, `-type async_reset port` should not have `-ignore` set.

1

Table 13 provides example cases.

Table 13—Example cases

	Example	Diagram

1	active_low reset on an input port with a virtual `associated_from_clocks`. This means that the signal driving `RST_in0` is asynchronous to `CLK1`. With this modeling, the IP will have an internal reset deassertion violation. If the signal driving `RST_in0` is from `CLK1`, the `associated_from_clocks` should be `CLK1`.
	
	Sample command:
	
	`cdc_set_module mod1       \`
	`    cdc_set_port RST_in0 \`
	`    -direction input     \`
	`    -type async_reset    \`
	`    -polarity low        \`
	`    -associated_from_clocks`
	`  VCLK1                   \`
	`    -associated_to_clocks`
	`  CLK1                    \`
	
	`cdc_set_port RST_in0      \`
	`    -associated_from_clocks`
	`VCLK1`
		

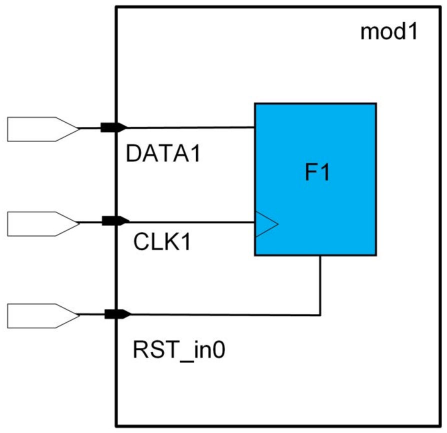
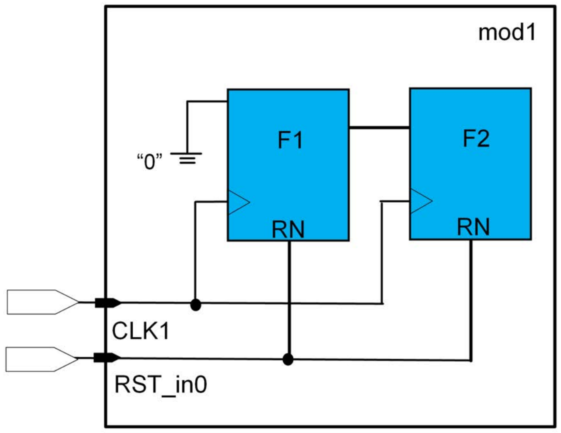
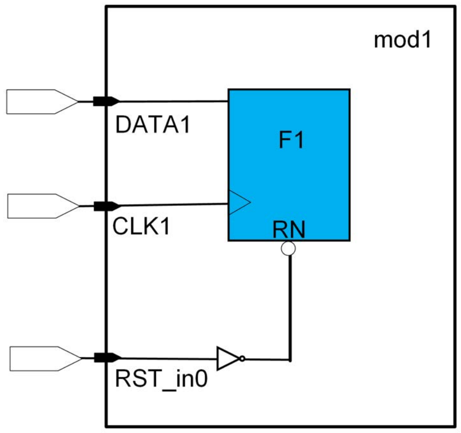
2	active_high reset on an input port with a virtual `associated_from_clocks`. This means that the signal driving `RST_in0` is asynchronous to `CLK1`. With this modeling, the IP will have an internal reset deassertion violation. If the signal driving `RST_in0` is from `CLK1`, the `associated_from_clocks` should be `CLK1`.
	
	Sample command:
	
	`cdc_set_module mod1`
	
	`cdc_set_port RST_in0   \`
	`    -direction input   \`
	`    -type async_reset \`
	`    -polarity high     \`
	`    -associated_to_clocks CLK1`
	
	`cdc_set_port RST_in0   \`
	`    -associated_from_clocks`
	`  VCLK1`
		

Table 13—Example cases

	Example	Diagram

3	active_high reset on an input port feeding a reset synchronizer
	
	Sample command:
	
	`cdc_set_module mod1`
	
	`cdc_set_port RST_in0      \`
	`    -direction input      \`
	`    -type async_reset     \`
	`    -polarity high        \`
	`    -logic internal_sync \`
	`    -associated_to_clocks CLK1`
		

4	active_high reset on an input port and feeding an inverter
	
	Sample command:
	
	`cdc_set_module mod1`
	
	`cdc_set_port RST_in0   \`
	`    -direction input  \`
	`    -type async_reset \`
	`    -polarity high     \`
	`    -logic inverter    \`
	`    -associated_to_clocks CLK1`
		

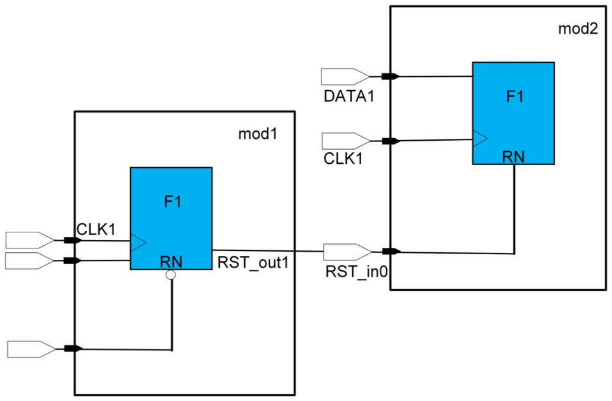
5	active_high reset on an input port with `associated_to_clocks` and known `associated_from_clocks`
	
	Sample command:
	
	`cdc_set_module mod2`
	
	`cdc_set_port RST_in0    \`
	`    -direction input    \`
	`    -type async_reset   \`
	`    -polarity high      \`
	`    -associated_from_clocks`
	`CLK1                    \`
	`    -associated_to_clocks CLK1`
		 mod2
		DATA1
		F1
		CLK1
		RN
		mod1
		F1
		CLK1
		RN
		RST_out1
		RST_in0

Table 14 illustrates failure modes.

Table 14—Failure modes

	Description	Diagram
1	**Reset driven by wrong clock domain**	

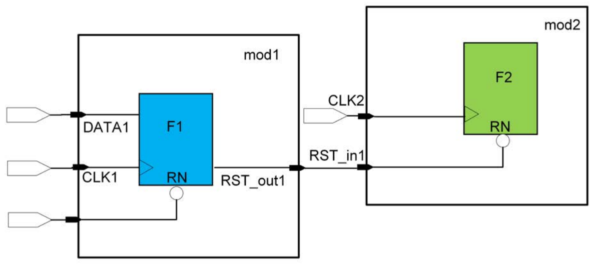
	Example: Clock mismatch. Assuming `CLK1` and `CLK2` to be async
	
	`cdc_set_module mod1`
	
	`cdc_set_port RST_out1 \`
	`    -direction output \`
	`    -type async_reset \`
	`    -polarity low     \`
	`    -associated_from_clocks`
	`  CLK1`
	
	`cdc_set_module mod2`
	
	`cdc_set_port RST_in1  \`
	`    -direction input  \`
	`    -type async_reset \`
	`    -polarity low     \`
	`    -associated_to_clocks CLK2`
	
	Expectation: An unsynchronized crossing on port `RST_in1` should be reported during top CDC analysis for clock domain mismatch.
	
	For a description of the related testcase, refer to Annex B.
		 mod1
		DATA1
		F1
		CLK1
		RN
		RST_out1
		CLK2
		RST_in1
		mod2
		F2
		RN

Table 14—Failure modes

	Description	Diagram
2	**Reset driven by correct clock domain but incorrect polarity**	

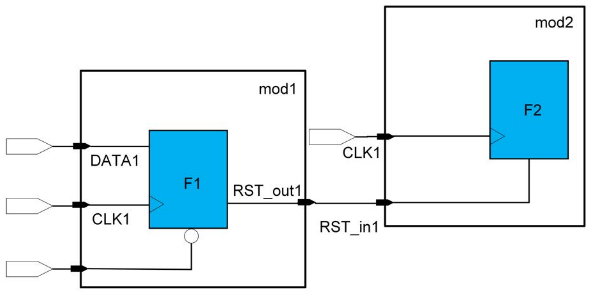
	Example: Simple incorrect polarity
	
	`cdc_set_module mod1`
	
	`cdc_set_port RST_out1 \`
	`-direction output     \`
	`-type async_reset     \`
	`-polarity low         \`
	`-associated_from_clocks CLK1`
	
	`cdc_set_module mod2`
	
	`cdc_set_port RST_in1  \`
	`    -direction input  \`
	`    -type async_reset \`
	`    -polarity high    \`
	`    -associated_from_clocks`
	`  CLK1 \`
	`    -associated_to_clocks clk1`
	
	Expectation: A conflict for polarity should be reported on port `RST_in1` during top CDC analysis.
	
	For a description of the related testcase, refer to Annex B.
		 mod2
		F2
		mod1
		DATA1
		F1
		CLK1
		RST_out1
		CLK1
		RST_in1

Table 14—Failure modes

	Description	Diagram
3	**Multiple ports**	

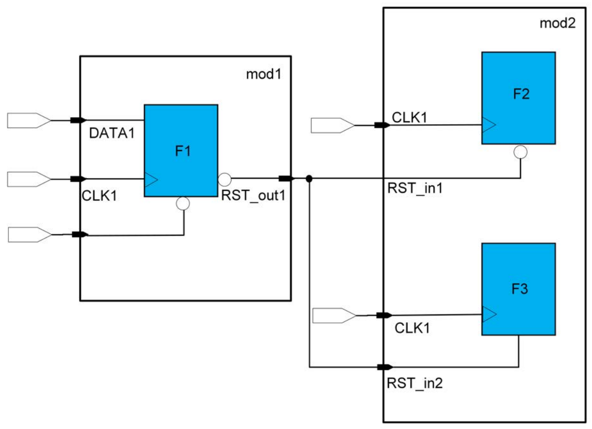
3a	Example 1: Multiple input ports in the same clock domain but with different polarity driven by the same reset
	
	`cdc_set_module mod1`
	
	`cdc_set_port RST_out1 \`
	`-direction output     \`
	`-type async_reset     \`
	`-polarity low        \`
	`-associated_from_clocks CLK1`
	
	`cdc_set_module mod2`
	
	`cdc_set_port RST_in1          \`
	`-direction input             \`
	`-type async_reset            \`
	`-polarity low                \`
	`-associated_from_clocks CLK1 \`
	`-associated_to_clocks CLK1`
	
	`cdc_set_port RST_in2          \`
	`-direction input             \`
	`-type async_reset            \`
	`-polarity high               \`
	`-associated_from_clocks CLK1 \`
	`-associated_to_clocks CLK1`
	
	Expectation: A conflict for polarity should be reported for the second port `RST_in2` and not for `RST_in1`.
	
	For a description of the related testcase, refer to Annex B.
		 mod1
		DATA1
		F1
		CLK1
		RST_out1
		mod2
		CLK1
		F2
		RST_in1
		F3
		CLK1
		RST_in2

Table 14—Failure modes

	Description	Diagram

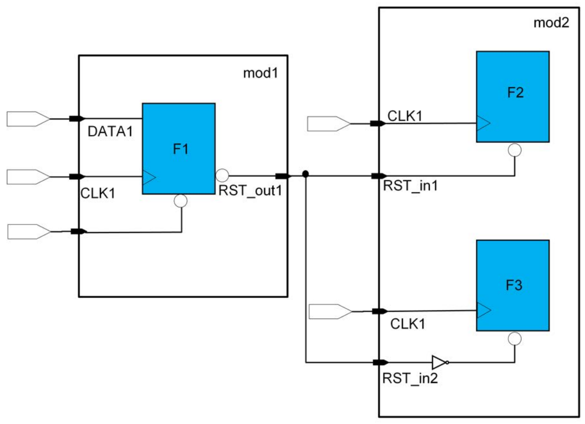
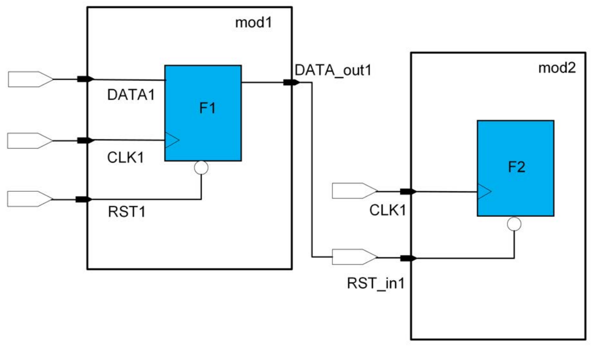
3b	Example 2: Multiple input ports in the same clock domain with the same polarity, but one is inverted and driven by the same reset
	
	`cdc_set_module mod1`
	
	`cdc_set_port RST_out1 \`
	`-direction output     \`
	`-type async_reset     \`
	`-polarity low         \`
	`-associated_from_clocks CLK1`
	
	`cdc_set_module mod2`
	
	`cdc_set_port RST_in1  \`
	`    -direction input  \`
	`    -type async_reset \`
	`    -polarity low     \`
	`    -associated_from_clocks`
	`  CLK1 \`
	`-associated_to_clocks CLK1`
	
	`cdc_set_port RST_in2        \`
	`-direction input            \`
	`-type async_reset           \`
	`-polarity high              \`
	`-associated_to_clocks CLK1 \`
	`-logic inverter`
	
	Expectation: A conflict for polarity should be reported for the second port `RST_in2` and not for `RST_in1`.
	
	For a description of the related testcase, refer to Annex B.
		 mod1
		DATA1
		F1
		CLK1
		RST_out1
		mod2
		CLK1
		F2
		RST_in1
		F3
		CLK1
		RST_in2

Table 14—Failure modes

	Description	Diagram
4	**Non-reset signal feeding into reset signal**	

	Example:
	
	`cdc_set_module mod1`
	
	`cdc_set_port DATA_out1 \`
	`-direction output      \`
	`-type data            \`
	`-polarity low         \`
	`-associated_from_clocks CLK1`
	
	`cdc_set_module mod2`
	
	`cdc_set_port RST_in1         \`
	`-direction input            \`
	`-type async_reset           \`
	`-polarity low               \`
	`-associated_from_clocks CLK1 \`
	`-associated_to_clocks CLK1`
	
	Expectation: A conflict for the signal type should be reported for port `RST_in1`. This might be expected, but it must be reviewed during top CDC analysis.
	
	For a description of the related testcase, refer to Annex B.
		 mod1
		DATA1
		F1
		CLK1
		RST1
		DATA_out1
		mod2
		F2
		CLK1
		RST_in1

4.5 Modeling abstracted blocks

The model of the input interface of an abstracted block can be described

— in terms of requirements to the driving circuit in the fan-in of the block (external model) or
— in terms of the relevant structures inside the block (internal model).

The internal model is a mandatory part of the input interface abstraction. The additional attributes of the external model are optional for input interface abstractions. The following examples describe the circuit in Figure 14.

The port attributes support references between data ports and the CDC control ports in both directions.

For a data port (`-type data`), the option `-cdc_control` specifies a unique CDC control port. If such a relation exists, it is mandatory to describe it in this way.

A CDC control port can synchronize one or many data ports. For a CDC control port (`-type cdc_control`) at the input interface, the option `-associated_inputs` specifies a list of data ports. Because this list may be long, it is an optional part of the abstracted model, which may be specified or omitted depending on the readability. The same rule applies for the option `-associated_outputs` of a CDC control port at the output interface. The option `-cdc_control_setup` specifies the number of destination clock cycles an `-associated_inputs` data port for this CDC control port must be stable before the CDC control port enables the data crossing. The option `-cdc_control_hold` specifies the number of destination clock cycles an `-associated_inputs` data port for this CDC control port must be stable after the CDC control port disables the data crossing.

```
cdc_set_port d1_i                  \  Name of port
    -direction input               \  Direction of port
    -type data                     \  Type of port
    -associated_from_clocks clk2_i \  External driver clock domain
    -cdc_control q1_i              \  Name of control port
    -associated_to_clocks clk1_i   \  Internal receiver’s clock domain
    -logic combo                   \  Port has combo logic on path to reg
```

```
cdc_set_port q1_i                  \  Name of port
    -direction input               \  Direction of port
    -type cdc_control              \  Control port used to qualify qi_i
    -cdc_data_from_clock clk2_i    \  External clock driving this port
    -associated_from_clocks clk1_i \  External driver clock domain
    -associated_inputs d1_i        \  Data ports controlled by this port
    -associated_to_clocks clk1_i   \  Internal receiver’s clock domain
    -logic combo                   \  Port has combo logic on path to reg
    -cdc_control_setup 2           \  d1_i must be stable for 2 clocks
                                   \    before q1_i
    -cdc_control_hold 1                d1_i must be stable for 1 clock after
                                        q1_i
```

The `associated_from_clocks` is the expected driving clock of the port (same semantics for types `data` and `cdc_control`). The `cdc_data_from_clock` is the source clock of the controlled data signal.

In an environment with asynchronous clocks being connected to `clk1_i` and `clk2_i`, the structure in the

```
cdc_set_port c1_i                  \   Name of port
    -direction input               \   Direction of port
    -type data                     \   Type of port
    -associated_from_clocks clk2_i \   External driver clock domain
    -associated_to_clocks {        \   Internal receiver's clock domains
                                   \     for multiple fanouts of port c1_i:
      clk2_i                       \   The first fanout is to a sequential
                                   \     running on clk2_i.
      {clk1_i internal_sync}       \   The second fanout is to a
                                   \     synchronizer running on clk1_i.
      clk1_i                       \   The third fanout is a sequential
      }                                  running on clk1_i.
```

last line should be reported as violated because of missing synchronization. It is acceptable to combine the first and third lines of the last "Internal" example above. The second line is required as a separate line because it describes a type of receiving circuit with a specific attribute (`-logic`) which is not covered by the other types of receivers.

For a description of the related testcase, refer to Annex B.

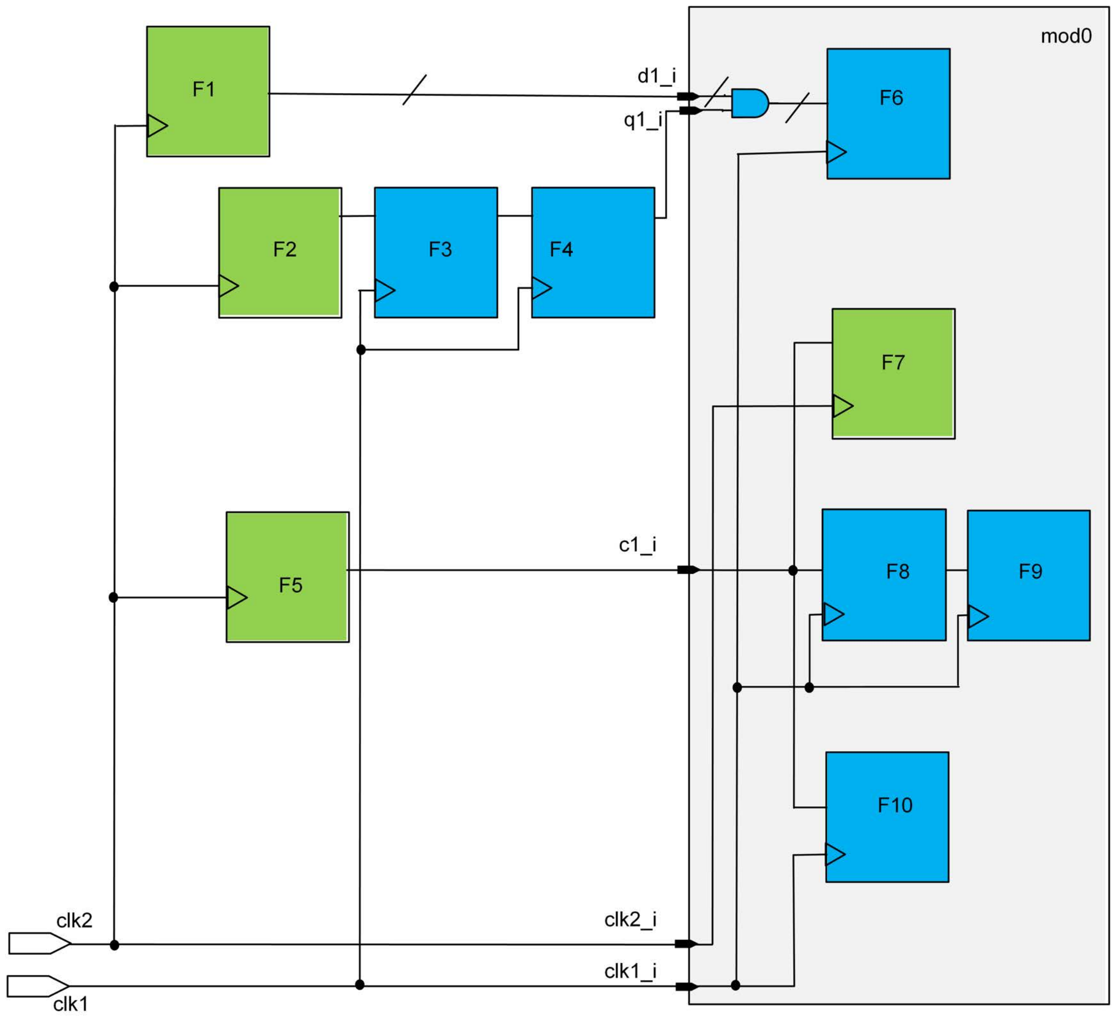


Figure 14—Module mod0 with abstracted input ports

The model of the output interface of an abstracted block is described in terms of the relevant structures inside the block (internal model). The following examples describe the circuit in Figure 15.

```
cdc_set_cdc_set_port d2_o          \  Name of port
    -direction output              \  Direction of port
    -type data                     \  Type of port
    -associated_from_clocks clk2_i \  External driver clock domain
    -cdc_control q2_o                 Output port controlled by d2_o
```

```
cdc_set_port q2_o                  \   Name of port
    -direction output              \   Direction of port
    -type cdc_control              \   Type of port
    -cdc_data_from_clock clk2_i    \   External clock driving this port
    -associated_from_clocks clk1_i \   External driver clock domain
    -associated_outputs d2_o           Output port controlled by q2_o
```

-associated_outputs d2_o is redundant with the information on port d2_o above, but this information can be added optionally:

```
cdc_set_port c2_o      \                Name of port
    -direction output \                 Direction of port
    -type data        \                 Type of port
    -associated_from_clocks clk2_i      External driver clock domain
```

For a description of the related testcase, refer to Annex B.

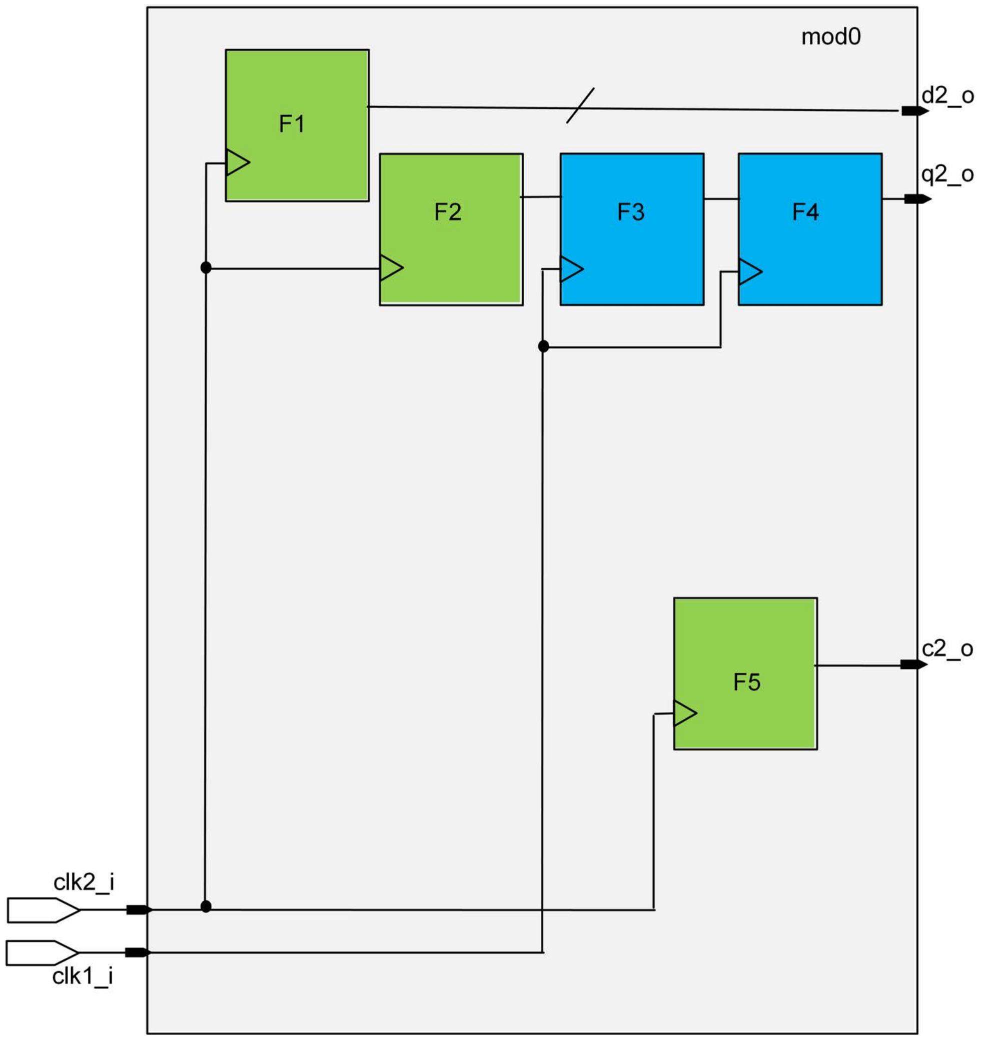


Figure 15—Module mod0 with abstracted output ports

4.6 Clock definitions

This section shows three examples of clock definitions. Figure 16 depicts two asynchronous clocks `txclk` and `rxclk`.

```
cdc_set_port txclk   \    Name of port
    -direction input \    Direction of port
    -type clock           Type of port
```

```
cdc_set_clock_group \
    -name tx        \           Name of clock group
    -clocks {txclk}             Name of clock in the clock
                                  group
```

```
cdc_set_port rxclk   \          Name of port
    -direction input \          Direction of port
    -type clock                 Type of port
```

```
cdc_set_clock_group \
    -name rx        \           Name of clock group
    -clocks {rxclk}             Name of clock in the clock
                                  group
```

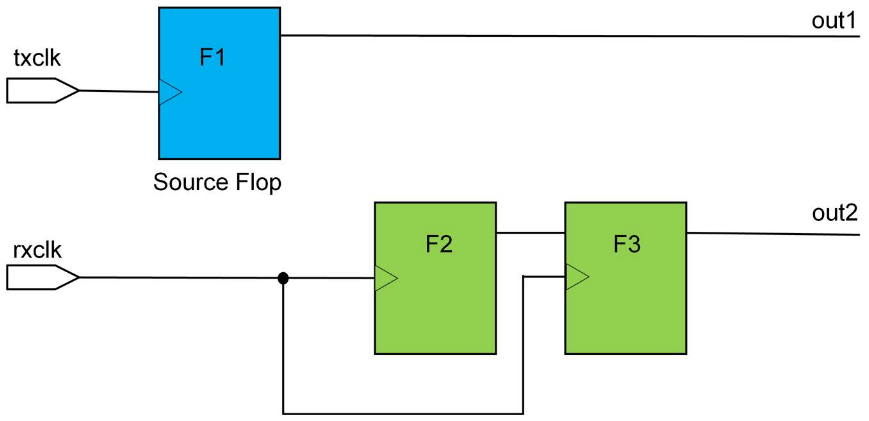


Figure 16—Clock definition A

1 Figure 17 depicts three asynchronous clocks `txclk`, `rxclk`, and `newclk`.

2

```
cdc_set_port txclk   \     Name of port
    -direction input \     Direction of port
    -type clock            Type of port
```

3

```
cdc_set_clock_group \
    -name tx        \      Name of clock group
    -clocks {txclk}        Name of clock in the clock group
```

4

```
cdc_set_port rxclk   \     Name of port
    -direction input \     Direction of port
    -type clock            Type of port
```

```
cdc_set_clock_group \
    -name rx        \      Name of clock group
    -clocks {rxclk}        Name of clock in the clock group
```

5

```
cdc_set_port newclk  \     Name of port
    -direction input \     Direction of port
    -type clock            Type of port
```

6

```
cdc_set_clock_group \
    -name new       \      Name of clock group
    -clocks {newclk}       Name of clock in the clock group
```

7

8 For `cdc_set_clock_group`, refer to 4.7.

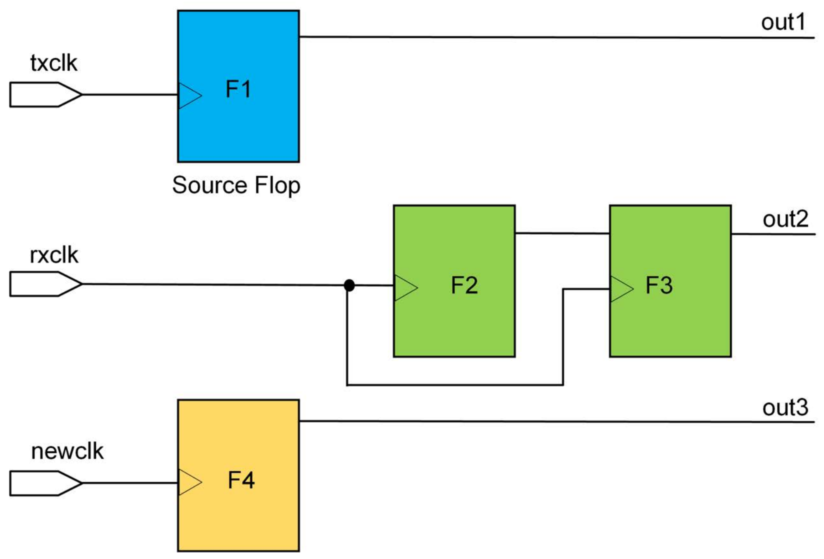


Figure 17—Clock definition B

Figure 18 depicts three clocks `txclk`, `rxclk`, and `newclk`. In this example, `txclk` and `newclk` are asynchronous to `rxclk`, but `txclk` and `newclk` are synchronous to each other.

```
cdc_set_port txclk   \         Name of port
    -direction input \         Direction of port
    -type clock                Type of port
```

```
cdc_set_port newclk \          Name of port
    -direction input           Direction of port
    -type clock     \          Type of port
```

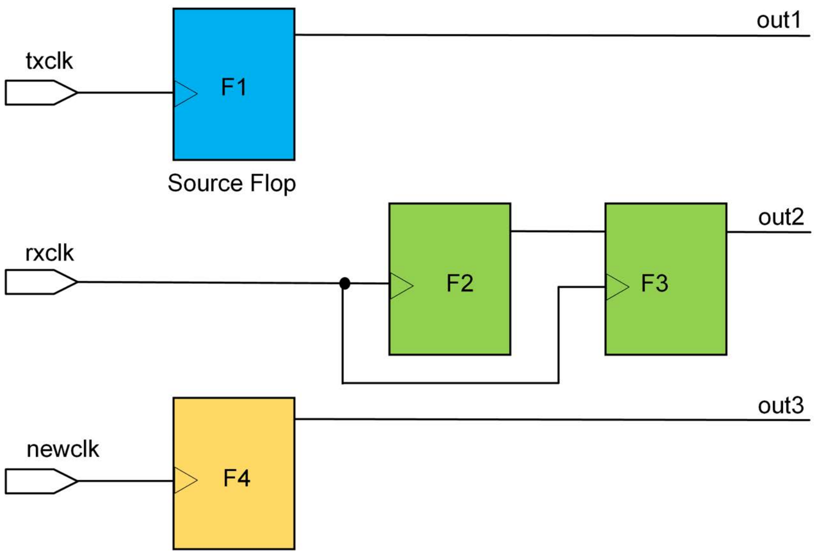
```
cdc_set_clock_group \
    -name tx        \          Name of clock group
    -clocks {txclk newclk}     Names of clocks in the clock group
```

```
cdc_set_clock_group \
    -name tx        \          Name of clock group
    -clocks {txclk newclk}     Names of clocks in the clock group
```

```
cdc_set_port rxclk   \         Name of port
    -direction input \         Direction of port
    -type clock                Type of port
```

```
cdc_set_clock_group \
    -name rx        \          Name of clock group
    -clocks {rxclk}            Name of clock in the clock group
```

1



2 **Figure 18—Clock definition C**

3

4.7 Clock relationships

The command `set_cdc_clock_group` is used to clearly define clock relationships in terms of synchronicity. The default assumption for CDC is that clocks are asynchronous to each other. Any two clocks that are members of a common CDC clock group are considered synchronous.

The command `set_cdc_clock_group` supports an optional attribute `-name`. If names are assigned to CDC clock groups, the names must be unique. Even if a clock group is a transitive continuation of others, it shall have a unique name. This approach avoids synchronicity relations between groups with the same name that may appear distributed over the abstracted model. Instead of introducing several groups with equal names, merge them into one group with a unique name. The `-name`'s space is local to an IP.

In the following example, the same synchronicity relation is described by different grouping and naming of the CDC clock groups. When expanding the groups to the set of all pairs of clocks that you can build from the group members, you receive the same set in both cases. In other words, both the example of one group and the example of three groups listed below are equivalent. Despite having three different group names, all `{clk1;clk2;clk3;clk4}` are synchronous to each other.

In one group:

```
cdc_set_clock_group      \
    -name sys_clk_domain \          Name of clock group
    -clocks {clk1 clk2 clk3 clk4}   Names of clocks in the clock group
```

In three groups:

```
cdc_set_clock_group       \
    -name center_and_left \         Name of clock group
    -clocks {clk1 clk2 clk3}        Names of clocks in the clock group
```

```
cdc_set_clock_group        \
    -name center_and_right \        Name of clock group
    -clocks {clk2 clk3 clk4}        Names of clocks in the clock group
```

```
cdc_set_clock_group
    -name balanced_branches         Name of clock group
    -clocks {clk1 clk4}             Names of clocks in the clock group
```

The usage of the groups in the context of CDC checks does not depend on preprocessing for merging or expansion of the groups but can also keep them as given by the abstracted model. An EDA tool could designate how to merge or expand these groups for top-level consumption.

The synchronicity relation of clocks {A, B, C} is reflexive (A sync to A) and symmetrically (A sync to B means B sync to A). In most applications, it is also transitive:

A sync to B and B sync to C means A sync to C.

In this case, the largest possible sets of synchronous clocks are disjoint; i.e., after merging the CDC clock groups as far as the synchronicity allows, each clock appears in exactly one of the CDC clock groups.

However, there are also applications with a non-transitive synchronicity of clocks, e.g, a root clock driving two generated clocks with “odd” division factors (large value for least common multiple) or two generated clocks that propagate to branches of the clock tree with a large physical distance. In both cases, it may be very expensive to implement synchronous timing arcs between the generated clocks, so instead, they may be considered as asynchronous to each other.

In cases of non-transitive synchronicity, the largest possible sets of synchronous clocks are maximum sets of compatibility that may have common members. However, it is not required to find these largest possible sets. Subsets and even pairs can describe the same synchronicity relation. The above-mentioned groups `center_and_left` and `center_and_right` without the last group `balanced_branches` are an example for a non-transitive synchronicity relation. `clk1` and `clk4` can be understood as non-balanced branches of the clock tree. Refer to Figure 22 for an example of a non-transitive use model.

The following examples show different cases of synchronicity. Blue solid arrows denote synchronous clock relationships. Red dashed arrows denote asynchronous clock relationships.

Figure 19 depicts three clocks. All are pairwise synchronous to each other so that they build a common clock group.

```
cdc_set_clock_group      \
    -name common_domain \       Name of clock group
    -clocks {clk gclk0 gclk1}   Names of clocks in the clock group
```

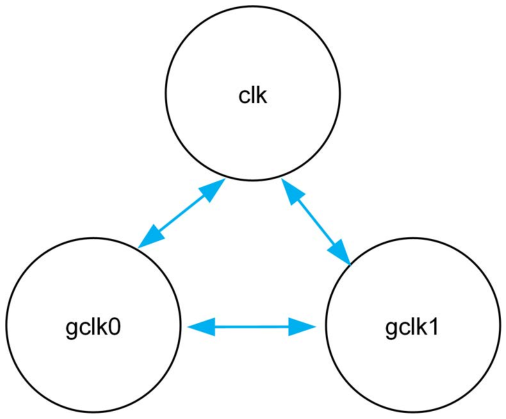


Figure 19—One clock domain

Figure 20 depicts two clock groups. The clock `clk` builds a group on its own because it is asynchronous to the other two clocks.

```
cdc_set_clock_group    \
    -name small_domain \        Name of clock group
    -clocks {clk}               Name of clock in the clock group


cdc_set_clock_group    \
    -name large_domain \        Name of clock group
    -clocks {gclk0 gclk1}       Name of clock in the clock group
```

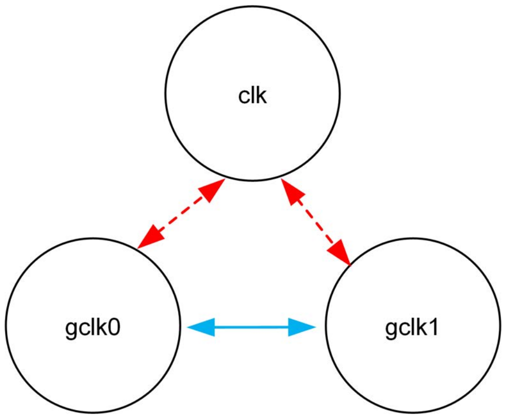


Figure 20—Two clock domains

Figure 21 depicts three clocks that are all pairwise asynchronous to each other so that every clock builds an individual clock group.

```
cdc_set_clock_group \
    -name domain_c  \           Name of clock group
    -clocks {clk}               Name of clock in the clock group
```

```
cdc_set_clock_group \
    -name domain_0  \           Name of clock group
    -clocks {gclk0}             Name of clock in the clock group
```

```
cdc_set_clock_group \
    -name domain_1  \           Name of clock group
    -clocks {gclk1}             Name of clock in the clock group
```

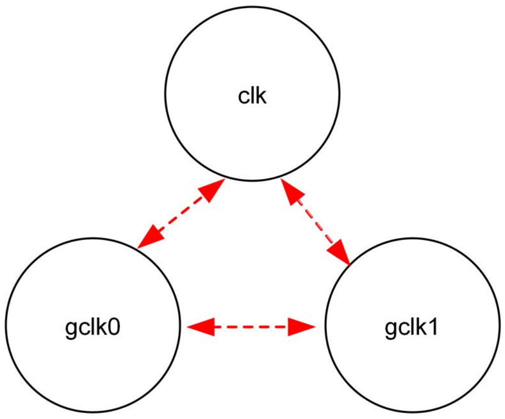


Figure 21—Three clock domains

Figure 22 depicts a non-transitive synchronicity between three clocks. The root clock `clk` is synchronous to both generated clocks `gclk0` and `gclk1`, but these are asynchronous to each other. The root clock `clk` appears as a member in both clock groups, whereas `gclk0` and `gclk1` do not appear in a common group. Both clock groups are described as maximum compatibility sets of clocks.

```
cdc_set_clock_group   \
    -name clk_branch0 \         Name of clock group
    -clocks {clk gclk0}         Names of clocks in the clock group
```

```
cdc_set_clock_group   \
    -name clk_branch1 \         Name of clock group
    -clocks {clk gclk1}         Names of clocks in the clock group
```

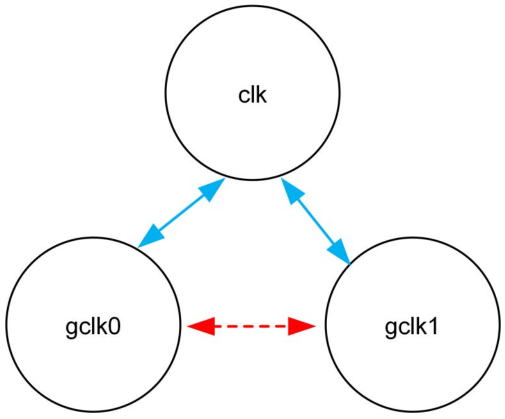


Figure 22—Two clock domains

4.8 Failure modes

This section shows attributes that may help a tool detect failure modes and their reasons in cases where an abstracted block is integrated into an encompassing design.

Figure 23 depicts the propagation of metastability with a mean time before failure (MTBF) that is too low. The failure is due to a missing synchronizer. This scenario assumes that `cin1_i` has an asynchronous driver. This failure mode can occur at ports of type `data` or of type `cdc_control`, which are described in the following models.

```
cdc_set_port cin1_i  \              Name of port
    -direction input \              Direction of port
    -type data       \              Type of port
    -associated_to_clocks clk1_i    Internal receiver's clock domains for
                                      multiple fanouts of port cin1_i
```

Independent from the example above, the port `cin1_i` in Figure 23 could be described as a `cdc_control` input (the synchronized data is not represented in the diagram):

```
cdc_set_port cin1_i  \              Name of port
    -direction input \              Direction of port
    -type cdc_control               Type of port
    -cdc_data_from_clock VCLK       External clock driving this port
    -associated_to_clocks clk1_i    Internal receiver's clock domains for
                                      multiple fanouts of port cin1_i
```

In both examples above, the encompassing design in Figure 23 introduces a CDC violation. The driver is related to `clk2` but should be related to `clk1`.

For a description of the related testcase, refer to Annex B.

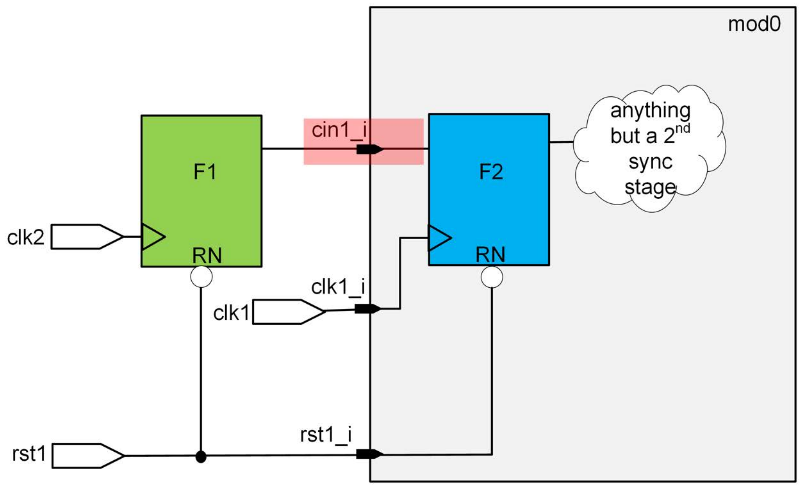


Figure 23—Failure mode due to missing synchronizer

Figure 24 depicts the propagation of an intermediate (random or metastable) data value. The failure is due to a missing cdc_control.

Model of the port `din1_i`:

```
cdc_set_port din1_i \               Name of port
-direction input    \               Direction of port
-type data          \               Type of port
-associated_to_clocks clk1_i        Internal receiver's clock domains for
                                      multiple fanouts of port din1_i
```

The encompassing design in Figure 24 introduces a CDC violation. The driver is related to `clk2`. Instead, the input signal should toggle related to `clk1`.

For a description of the related testcase, refer to Annex B.

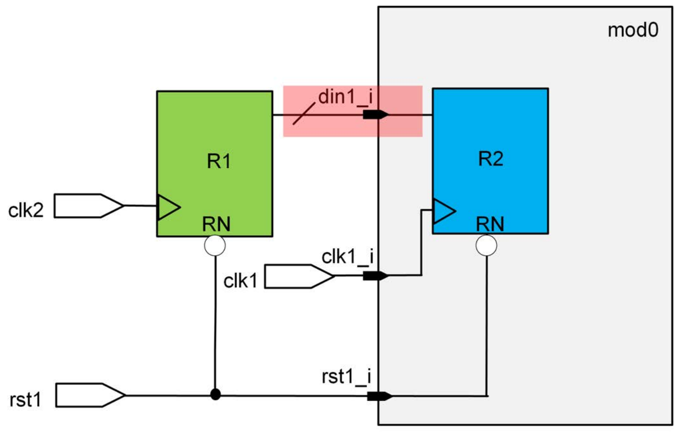


Figure 24—Failure mode due to missing synchronized control

Figure 25 depicts an asynchronous deassertion of an asynchronous reset (CDC on reset tree). The failure is due to a missing reset synchronizer.

```
cdc_set_module mod0                  Name of module
```

```
cdc_set_port rst_i                 \   Name of port
    -direction input               \   Direction of port
    -type async_reset              \   Type of port
    -associated_to_clocks clk1_i \     Internal receiver’s clock domains for
                                   \     multiple fanouts of port rst_i
    -polarity low                      Active low asynchronous reset
```

```
cdc_set_module driver                Name of module
```

```
cdc_set_port out                \      Name of port
-direction output               \      Direction of port
-type async_reset               \      Type of port
-associated_from_clocks clk2 \         External driver clock domain
-polarity low                          Active low asynchronous reset
```

For a description of the related testcase, refer to Annex B.

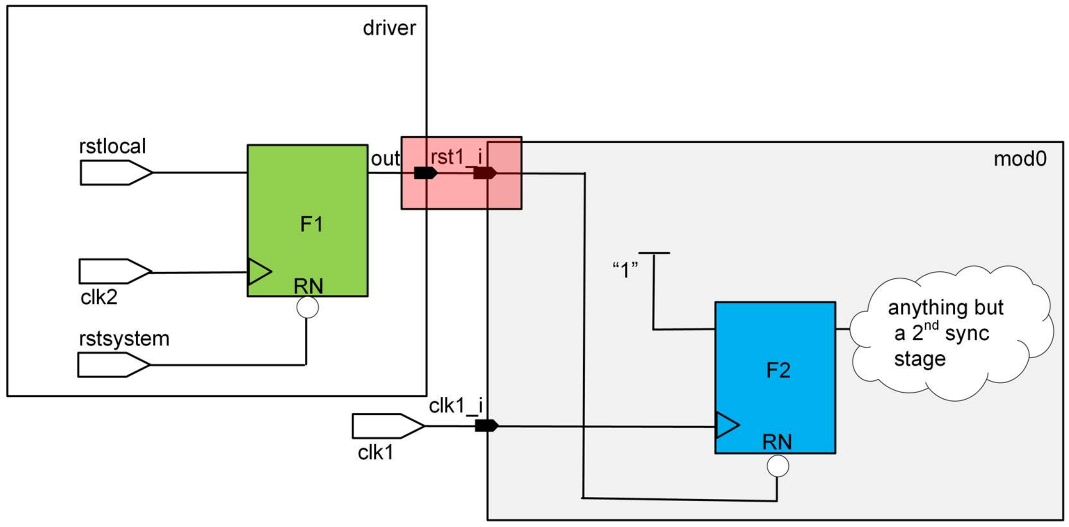


Figure 25—Failure mode due to missing reset synchronizer

We cannot have combo logic that can glitch, for instance, on a clock or a reset tree. Figure 26 shows that if the combo logic can glitch on the clock or reset network, there will be a violation.

```
cdc_set_module mod0                 Name of module
```

```
cdc_set_port out1_o           \     Name of port
-direction output             \     Direction of port
-type data                    \     Type of port
-logic combo                  \     Port has combo logic on path to reg
-associated_from_clocks clk1 \      External driver clock domain
-associated_inputs {in1_i in2_i}    Data ports controlled by this port
```

```
cdc_set_module mod0                 Name of module
```

```
cdc_set_port out2_o           \     Name of port
-direction output             \     Direction of port
-type data                    \     Type of port
-logic combo                  \     Port has combo logic on path to reg
-associated_from_clocks clk2 \      External driver clock domain
-associated_inputs {in3_i in4_i}    Data ports controlled by this port
```

NOTE—The attribute `-logic combo` describes a potentially glitching combinatorial logic. The attribute `-logic glitch_free_combo` describes the opposite.

For a description of the related testcase, refer to Annex B.

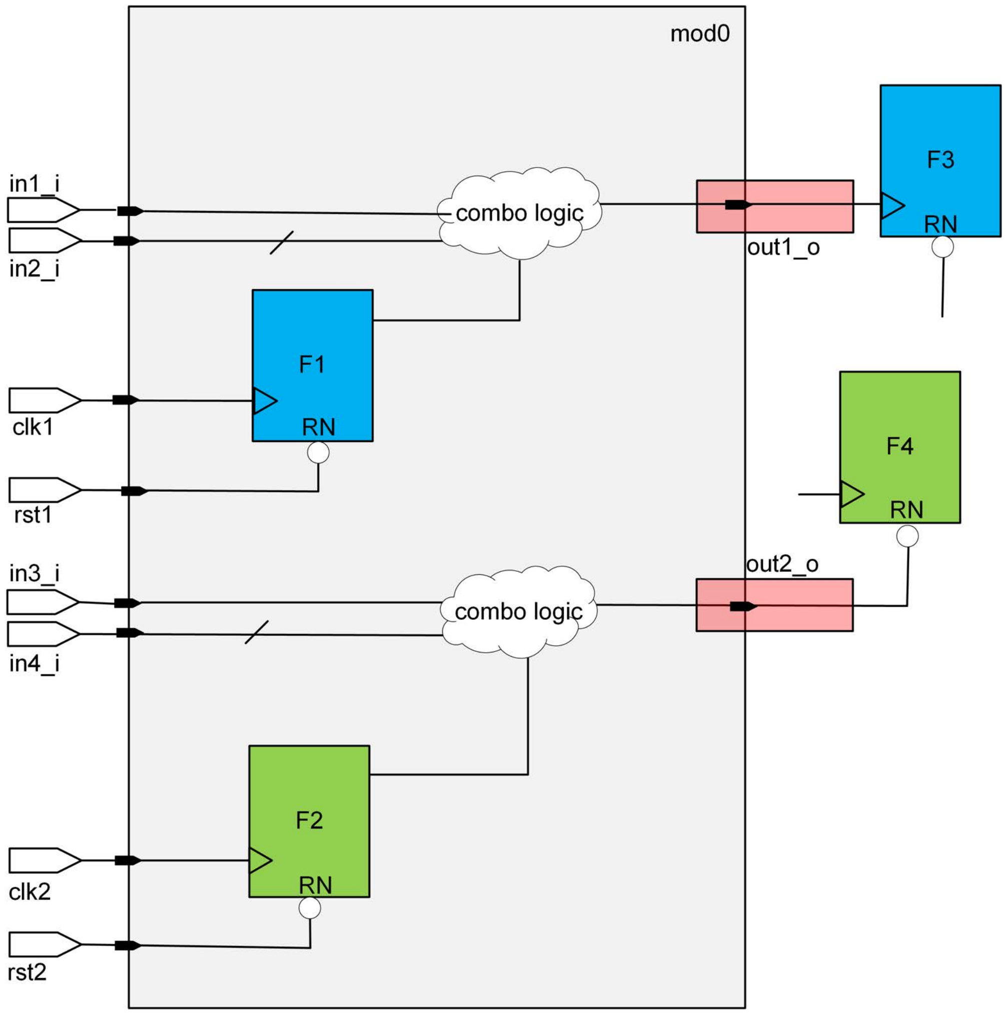


Figure 26—Failure mode due to combo logic driving clock/reset

5. Support for RDC verification

This clause details output requirements for IP with multiple resets from the top level and RDC control within and outside the IP. See Table 1 for the supported RDC attributes along with their domain, type, accepted values, whether they are mandatory, and clarifying comments.

To support RDC interface verification, the CDC attribute tables (See Clause 4) have been updated with the following content beginning with the CDC LRM version 0.3.

a) New Domain
   1) `set_reset_group`: Defines the reset relation. There are no RDC violations between resets that appear together in the same reset group.

b) New attributes
   1) `-rdc_control`: Defines the RDC control or net that is associated with the data port to avoid a timing violation on RDC.
   2) `-rdc_data_from_reset`: Defines the source reset of RDC, i.e., start point of the sequential's reset. This attribute shall be used together with `-rdc_control`. This is the source reset being blocked by `-rdc_control`.
   3) `-rdc_data_to_reset`: Defines the destination reset of RDC, i.e., the end point of the sequential's reset. This attribute shall be used together with `-rdc_control`.
   4) `-rdc_data_to_clock`: Defines the destination clock of the RDC end point of the sequential's clock. This attribute shall be used together with `-rdc_control`. This is the destination clock being gated by `-rdc_control`.
   5) `-rdc_clock_gate_location`: Defines the location of the clock gate, i.e., whether it is internal or external. This attribute shall be used together with `-rdc_data_to_clock`.
   6) `-associated_from_reset`: Defines the source reset of a port.
   7) `-associated_to_reset`: Defines the destination reset of a port.

c) New Values
   1) `rdc_control` for the `-type` attribute: Defines the RDC control. The usage for this attribute is the same as `cdc_control`.
   2) `virtual_reset` for the `-type` attribute: Defines a virtual reset. The usage for this attribute is the same as `virtual_clock`.

The usage of the new additions will be described with scenarios in the following sections.

5.1 Requirements for IP with control outside the IP

This section describes support for RDC verification at the top level without re-abstracting output collateral of the IP. It is assumed that when the output collateral for the IP is generated, the IP does not have any knowledge of the number of resets driving the IP's interface reset ports.

5.1.1 Scenario 1

This section describes support for an RDC interface at the top level without the need to read and process the RTL of the IP as shown in Figure 27. The IP shall capture the following information in its output collateral for the top-level interface RDC verification. The code in the example below describes `mod1`.

```
cdc_set_port a                   \   Name of port
    -direction input             \   Direction of port
    -type data                   \   Type of port
    -logic internal_sync         \   Internal synchronizer
    -associated_from_clocks clk \   External driver's clock domain
    -associated_to_clocks clk        Internal receiver clock
```

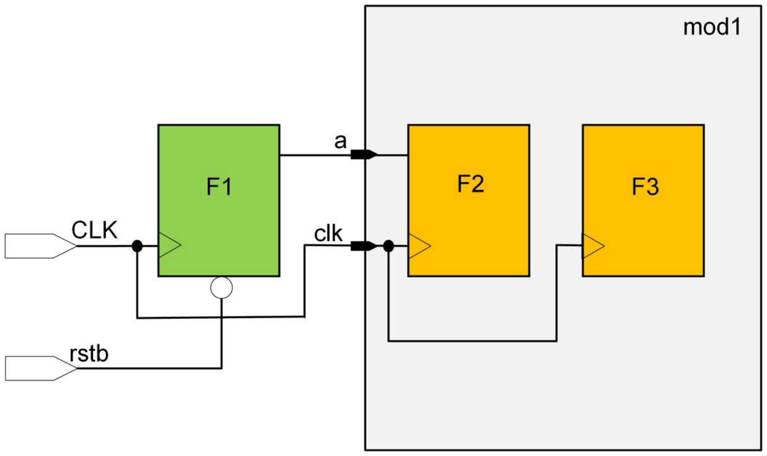


Figure 27—External rstb with internal synchronization logic

When the IP is integrated by the top-level logic as shown in Figure 27, the connectivity, clock domain, and reset domain of the IP at the top shall match the IP's modeling. As such, the following shall be checked by the tool:

When a receiving port has a synchronizer property, any RDC can be ignored because the synchronizer is present within the IP. In Figure 27, port `a` of the IP is a receiving port connected to the input of the synchronizer.

If the IP and top have mismatches, the tool can report them to the user for investigation during the top-level verification.

For a description of the related testcase, refer to Annex B.

5.1.2 Scenario 2

This section describes support for an RDC interface at the top level without the need to read and process the RTL of the IP as shown in Figure 28. The IP shall capture the following information in its output collateral for the top-level interface RDC verification.

```
cdc_set_port b                              \  Name of port
    -direction input                        \  Direction of port
    -type data                              \  Type of port
    -associated_from_clocks clk             \  External driver clock domain
    -associated_from_reset virtual_reset_b  \  External driver reset,
                                            \    virtual to the block
    -associated_to_reset rstb2b             \  Internal receiver reset
    -associated_to_clocks clk                  Internal receiver clock
```

```
cdc_set_port rstb2b              \             Name of port
    -direction input             \             Direction of port
    -type async_reset            \             Type of port
    -polarity low                \             Active low asynchronous reset
    -associated_from_clocks clk  \             External driver clock domain
    -associated_to_clocks clk                  Internal receiver clock
```

```
cdc_set_port virtual_reset_b \                 Name of port
    -direction input         \                 Direction of port
    -type virtual_reset                        Type of port
```

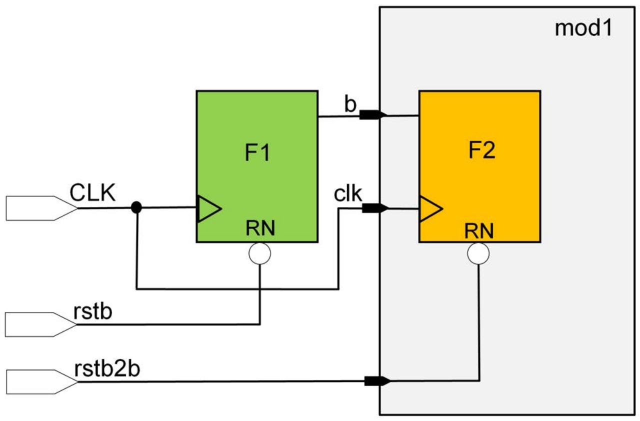


Figure 28—rstb2b of IP with associated reset

When the IP is integrated by the top-level logic as shown in Figure 28, the connectivity, clock domain, and reset domain of the IP at the top shall match the IP's modeling. As such, the following shall be checked by the tool:

a) The top level's reset and clock signals for port `b` of `IP` shall align with the IP's assumptions to ensure there are no CDC or RDC issues. Specifically, the tool shall compare the reset domain of `rstb` with that of the receiver reset `rstb2b` of `IP`. It is essential to ensure that `rstb2b` occurs before `rstb` to maintain the correct reset sequence, which is critical for the stable operation of sequential elements within the design.

b) If the design constraints guarantee that `rst2b` precedes `rstb`, or if `rst2b` of `IP` is properly connected at the interface level to `rstb`, there will be no RDC violations. Thus, if these conditions are met, no synchronization is necessary.

If the IP and top have mismatches, the tool can report them to the user for investigation during the top-level verification.

5.1.3 Scenario 3

Figure 29 captures how the RDC interface is managed when the control is handled outside the IP with port `C` being driven by logic from reset qualifier `rdcq`. This section describes support for an RDC interface at the top level without the need to read and process the RTL of the IP. The IP shall capture the following information in its output collateral for the top-level interface RDC verification.

```
cdc_set_port c                  \  Name of port
    -direction input            \  Direction of port
    -type data                  \  Type of port
    -associated_from_clocks ck  \  External driver clock domain
    -associated_to_clocks ck       Internal receiver clock
```

```
cdc_set_port ck      \             Name of port
    -direction input \             Direction of port
    -type clock                    Type of port
```

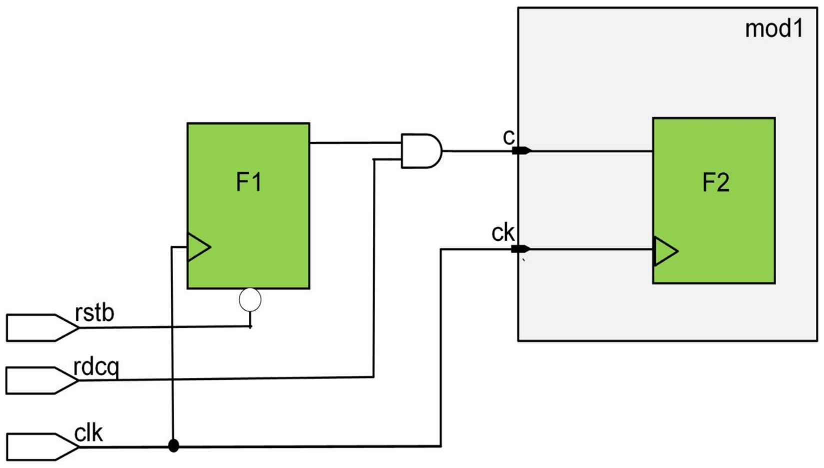


Figure 29—External rdcq on the data signal of the IP

When the IP is integrated by the top-level logic as shown in Figure 29, the connectivity, clock domain, and reset domain of the IP at the top shall match the IP's modeling. As such, the following shall be checked by the tool:

a) The external RDC qualifier `rdcq` shall ensure there is no RDC from the receiving port `c` of `IP` to the `rx` flop of `IP` at the top level due to a reset qualifier at the data pin. When `rstb` is asserted at the top level, it is blocked by `rdcq`, eliminating the need for an additional RDC qualifier at the IP level.

b) Upon integrating this IP at the top level, it is crucial to have an RDC qualifier to manage the RDC from `rstb` to the receiving port `c` of `IP`. The `rdcq` signal effectively blocks the RDC, and this mechanism is modeled as an RDC qualifier/control at the top level.

If the IP and top have mismatches, the tool can report them to the user for investigation during the top-level verification.

5.1.4 Scenario 4

Figure 30 captures using a reset qualifier on a clock signal. It is crucial to ensure that the data port receives the clock signal that has been properly qualified by the reset. This section describes support for an RDC interface at the top level without the need to read and process the RTL of the IP. The IP shall capture the following information in its output collateral for the top-level interface RDC verification.

```
cdc_set_port d                  \  Name of port
    -direction input            \  Direction of port
    -type data                  \  Type of port
    -associated_from_clocks ck \  External driver clock domain
    -associated_to_clocks ck       Internal receiver clock
```

```
cdc_set_port ck      \             Name of port
    -direction input \             Direction of port
    -type clock                    Type of port
```

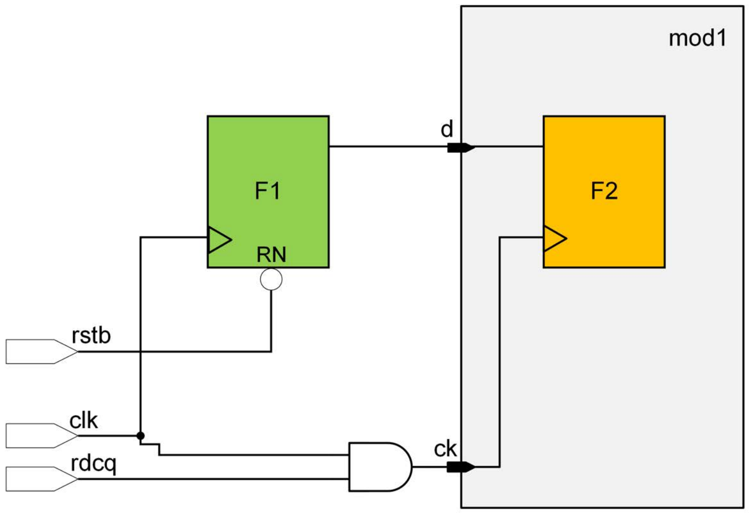


Figure 30—External rdcq on the clock signal of the IP

When the IP is integrated by the top-level logic as shown in Figure 30, the connectivity, clock domain, and reset domain of the IP at the top shall match the IP's modeling. As such, the tool shall verify that the data port is correctly receiving the reset-qualified clock signal. Since the clock `ck` is present in the IP and is the driver of receiving port `d` of `IP`, it is essential to use this information for top-level analysis. Knowing that `ck` of `IP` is gated before `rstb` is asserted ensures no internal RDC violations occur.

If the IP and top have mismatches, the tool can report them to the user for investigation during the top-level verification.

5.2 Requirements for IP with control within the IP

This section describes the necessary interface information for an IP design that has RDC control within the IP and multiple resets driven from the top level. It is assumed that when the output collateral for the IP is generated, the IP has no knowledge of the number of resets that drive the IP's interface reset ports. Five scenarios will be presented in this section with usage for the attributes introduced in the CDC LRM version 0.3.

5.2.1 Scenario 1

This section describes support for an RDC interface at the top level without the need to read and process the RTL of the IP as shown in Figure 31. It is recommended for the IP to capture the following information in its output collateral for the top-level RDC interface verification.

```
cdc_set_port rstb                 \  Name of port
    -direction input              \  Direction of port
    -type async_reset             \  Type of port
    -polarity low                 \  Active low asynchronous reset
    -associated_from_clocks clkx  \  External driver clock domain
    -associated_to_clocks clkx       Internal receiver clock
```

```
cdc_set_port rstx                 \  Name of port
    -direction input              \  Direction of port
    -type async_reset             \  Type of port
    -polarity low                 \  Active low asynchronous reset
    -associated_from_clocks clkx  \  External driver clock domain
    -associated_to_clocks clkx       Internal receiver clock
```

```
cdc_set_port rdcq                 \  Name of port
    -direction input              \  Direction of port
    -type rdc_control             \  Type of port
    -rdc_data_from_reset rstb     \  Source reset port of RDC (start point
                                  \    of sequential's reset)
    -rdc_data_to_reset rstx       \  Destination reset port of RDC (the end
                                  \    point of sequential's reset)
    -associated_to_clocks clkx       External driver clock domain
```

```
cdc_set_port clkx    \          Name of port
    -direction input \          Direction of port
    -type clock                 Type of port
```

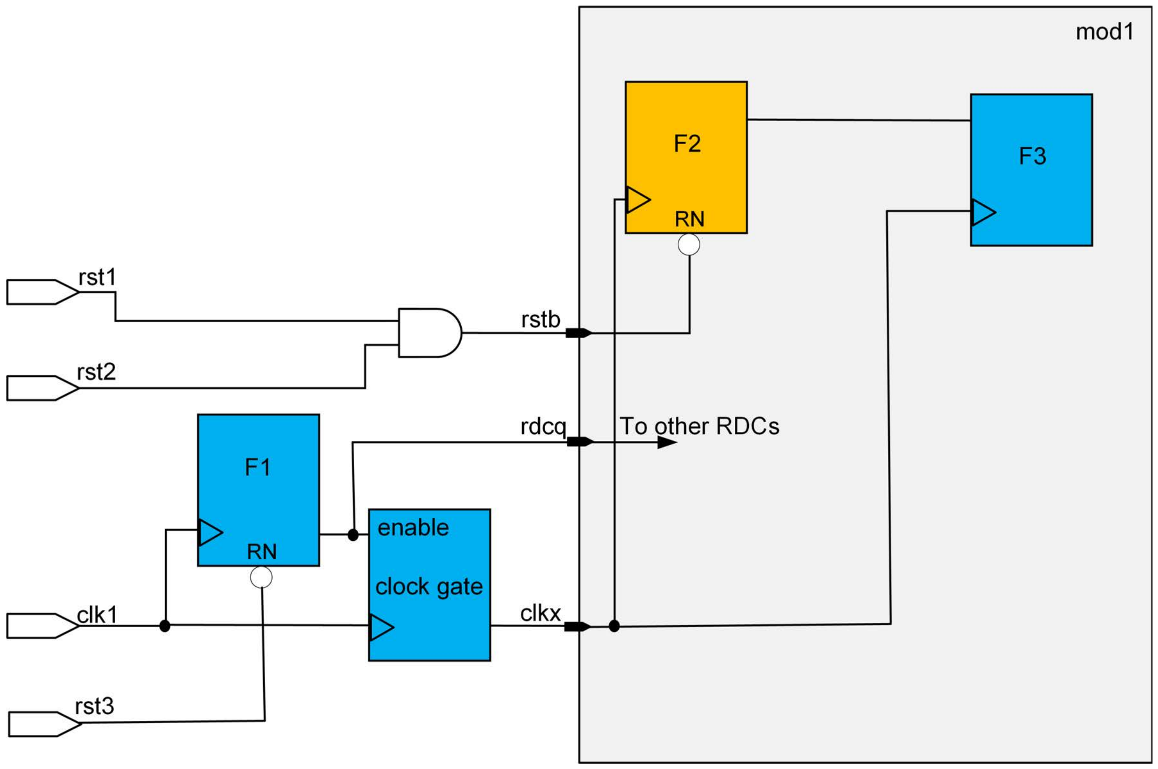


Figure 31—rstb of IP driven by multiple resets

`-associated_to_clocks` for `cdc_set_port rstb` and `cdc_set_port rstx` conveys information used by the IP to ensure there is no reset deassertion issue within the IP. Although this information is not needed for RDC interface modeling, it is needed for CDC interface verification. It is illustrated in this example because the output collateral covers interface information needed for both CDC and RDC verification.

With `-associated_from_clocks` and `-associated_to_clocks` information for `cdc_set_port rstb` and `cdc_set_port rstx` respectively, this indicates that the signal driving the async reset pin is synchronous to the receiver flop's clock. EDA tools may determine how much information will be generated for the IP's output collateral.

`-type rdc_control` was added to the CDC LRM version 0.3 to support RDC modeling. The usage is similar to that of `cdc_control`, which was introduce in the CDC LRM version 0.1. The only difference is that `rdc_control` is use for blocking RDC instead of CDC.

Along with the introduction of the `-type rdc_control` attribute, the following four new attributes are also added: `-rdc_data_from_reset`, `-rdc_data_to_reset`, `-rdc_data_to_clock`, and `-rdc_clock_gate_location`. These attributes are used with `-type rdc_control` to describe the blocking conditions. For example,

```
cdc_set_port rdcq                   \   Name of port
    -direction input                \   Direction of port
    -type rdc_control               \   Type of port
    -rdc_data_from_reset rstb       \   Source reset port of RDC (start point
                                    \     of sequential’s reset)
    -rdc_data_to_reset rstx         \   Destination reset port of RDC (the end
                                    \     point of sequential’s reset)
    -associated_to_clocks clkx          Internal receiver clock
```

The command above specifies that `rdcq` is an RDC control signal that is used to block any RDC from `rstb` to `rstx`. The signal `rdcq` goes into the `clkx` domain in `mod1`.

The usage of `-rdc_data_to_clock` and `-rdc_clock_gate_location` is shown in Scenario 2 (see 5.2.2).

When the IP is integrated by the top-level logic as shown in Figure 31, the connectivity, clock domain, and reset domain of the IP at the top shall match the IP’s modeling. As such, the following shall be checked by the tool.

a) The top level’s reset for `rdcq` is aligned with the IP’s assumption such that `rdcq` is driven by a reset that is in the same reset domain as the `IP.rstx`.

b) The top level’s clock for `rdcq` is aligned with the IP’s assumption such that `rdcq` is driven by a clock that is in the same clock domain as `IP.clkx`.

c) Based on the reset relationship at the top level, the following assumption shall be true at the top level:

```
cdc_set_port rdcq                             \   Name of port
    -direction input                          \   Direction of port
    -type rdc_control                         \   Type of port
    -rdc_data_from_reset top.rst1 top.rst2    \   Source reset port of RDC
                                              \     (start point of
                                              \     sequential’s reset)
    -rdc_data_to_reset top.rst3               \   Destination reset port of
                                              \     RDC (the end point of
                                              \     sequential’s reset)
    -associated_to_clocks top.clkx                Internal receiver clock
    -associated_from_clocks top.clkx              External driver clock domain
```

If the IP and top have mismatches, the tool can report them to the user for investigation during the top-level verification.

For a description of the related testcase, refer to Annex B.

5.2.2 Scenario 2

This section describes support for an RDC interface at the top level without the need to read and process the RTL of the IP as shown in Figure 32. It is recommended for the IP to capture the following information in its output collateral for the top-level RDC interface verification.

```
cdc_set_port rstb                   \          Name of port
    -direction input                \          Direction of port
    -type async_reset               \          Type of port
    -polarity low                   \          Active low asynchronous reset
    -associated_from_clocks clkx    \          External driver clock domain
    -associated_to_clocks clkx                 Internal receiver clock
```

```
cdc_set_port rdcq                               \   Name of port
    -direction input                            \   Direction of port
    -type rdc_control                           \   Type of port
    -rdc_data_from_reset rstb                   \   Source reset port of RDC
                                                \     (start point of
                                                \     sequential’s reset)
    -rdc_data_to_clock clkx                     \   Destination clock port of
                                                \     RDC (end point of
                                                \     sequential’s clock)
    -rdc_clock_gate_location external           \   Location of the clock gate
                                                \     cell
    -associated_to_clocks clkx                  \   Internal receiver clock
    -associated_from_reset virtual_reset_rdcq       External driver reset,
                                                      virtual to the block
```

```
cdc_set_port clkx    \                              Name of port
    -direction input \                              Direction of port
    -type clock                                     Type of port
```

```
cdc_set_port virtual_reset_rdcq     \               Name of port
    -direction input                \               Direction of port
    -type virtual_reset                             Type of port
```

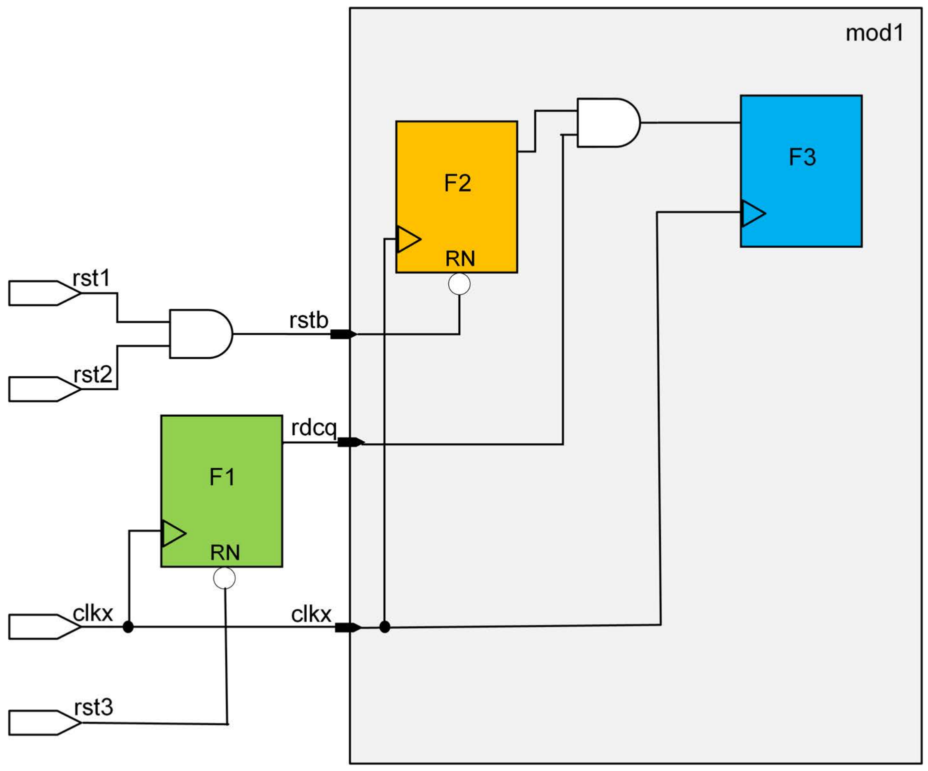


Figure 32—rstb of IP driven by multiple resets. Destination flop has no async reset pin

When `-rdc_data_from_reset` and `-rdc_data_to_clock` are used together with `-type rdc_control`, they specify that when the source reset (the reset specified with `-rdc_data_from_reset`) goes into reset, it is expected that the RDC endpoint's flop's clock (the clock specified with `-rdc_data_to_clock`) shall be gated to prevent metastability. `-rdc_clock_gate_location` defines the location of the clock gate cell. In this Scenario 2 example, the IP expects `clkx` to be gated by the SoC. The attribute `-associated_from_reset` is intended for use by the top level to ensure there is no RDC from the reset specified by `-associated_from_reset` to the RDC endpoint's flop in the IP after integration.

When the IP is integrated by the top-level logic as shown in Figure 32, the connectivity, clock domain, reset domain, clock, and reset relationship of the IP at the top level shall match the IP's modeling. As such, the following shall be checked by the tool.

a) The top level's reset and clock for `rdcq` is aligned with the IP's assumption such that when the reset from `rdcq` is asserted, the clock for the IP, i.e., `IP.clkx` shall be gated.

   The following interface information from the IP indicates that `IP.rdcq` is expecting `IP.clkx` to gate when the reset that drives `IP.rdcq` is asserted.

```
cdc_set_port rdcq                            \  Name of port
    -direction input                         \  Direction of port
    -type rdc_control                        \  Type of port
    -rdc_data_from_reset rstb                \  Source reset port of RDC
                                             \    (start point of
                                             \    sequential’s reset)
    -rdc_data_to_clock clkx                  \  Destination clock port of
                                             \    RDC (end point of
                                             \    sequential’s clock)
    -rdc_clock_gate_location external        \  Location of the clock gate
                                             \    cell
    -associated_to_clock clkx                \  Internal receiver clock
    -associated_from_reset virtual_reset_rdcq  External driver reset of the
                                                 port
```

b) Based on the reset relationship at the top level, the following assumption shall be true at the top level.

```
cdc_set_port rdcq                           \   Name of port
    -direction input                        \   Direction of port
    -type rdc_control                       \   Type of port
    -rdc_data_from_reset top.rst1 top.rst2  \   Source reset of RDC (start
                                            \     point of sequential’s
                                            \     reset)
    -rdc_data_to_clock top.clkx             \   Destination clock port of
                                            \     RDC (end point of
                                            \     sequential’s clock)
    -rdc_clock_gate_location external       \   Location of the clock gate
                                            \     cell
    -associated_from_clocks top.clkx        \   External driver clock domain
    -associated_to_clocks top.clkx          \   Internal receiver clock
    -associated_from_reset top.rst3             External driver reset of the
                                                  port
```

`cdc_set_port rdcq` shall block the RDC before `top.rst1` and `top.rst2` assert.

If the IP and top have mismatches, the tool can report them to the user for investigation during the top-level verification.

For a description of the related testcase, refer to Annex B.

5.2.3 Scenario 3

This section describes support for an RDC interface at the top level without the need to read and process the RTL of the IP shown in Figure 33. It is recommended for the IP to capture the following information in its output collateral for the top-level RDC interface verification.

```
cdc_set_port a                             \  Name of port
    -direction input                       \  Direction of port
    -type data                             \  Type of port
    -associated_from_clocks clkx           \  External driver clock domain
    -associated_from_reset virtual_reset_a \  The driver reset of the port
    -associated_to_reset rstb              \  The receiver reset of a port
    -associated_to_clocks clkx                Internal receiver clock
```

```
cdc_set_port rstb                \            Name of port
    -direction input             \            Direction of port
    -type async_reset            \            Type of port
    -polarity low                \            Active low asynchronous reset
    -associated_from_clocks clkx \            External driver clock domain
    -associated_to_clocks clkx                Internal receiver clock
```

```
cdc_set_port rdcq                           \  Name of port
    -direction input                        \  Direction of port
    -type rdc_control                       \  Type of port
    -rdc_data_from_reset rstb               \  Source reset of RDC (start
                                            \    point of sequential’s
                                            \    reset)
    -rdc_data_to_clock clkx                 \  Destination clock port of
                                            \    RDC (end point of
                                            \    sequential’s clock)
    -rdc_clock_gate_location external       \  Location of the clock gate
                                            \    cell
    -associated_to_clock clkx               \  Internal receiver clock
    -associated_from_reset virtual_reset_rdcq  The driver reset of the port
```

```
cdc_set_port clkx    \                         Name of port
    -direction input \                         Direction of port
    -type clock                                Type of port
```

```
cdc_set_port virtual_reset_rdcq \              Name of port
    -direction input            \              Direction of port
    -type virtual_reset                        Type of port
```

```
cdc_set_port virtual_reset_a \              Name of port
    -direction input         \              Direction of port
    -type virtual_reset                     Type of port
```

```
set_reset_group                  \          Definition of the reset relation
    -name reset_domain_1         \          Name of the reset domain
    -reset {virtual_reset_a; rstb}          Names of the resets
```

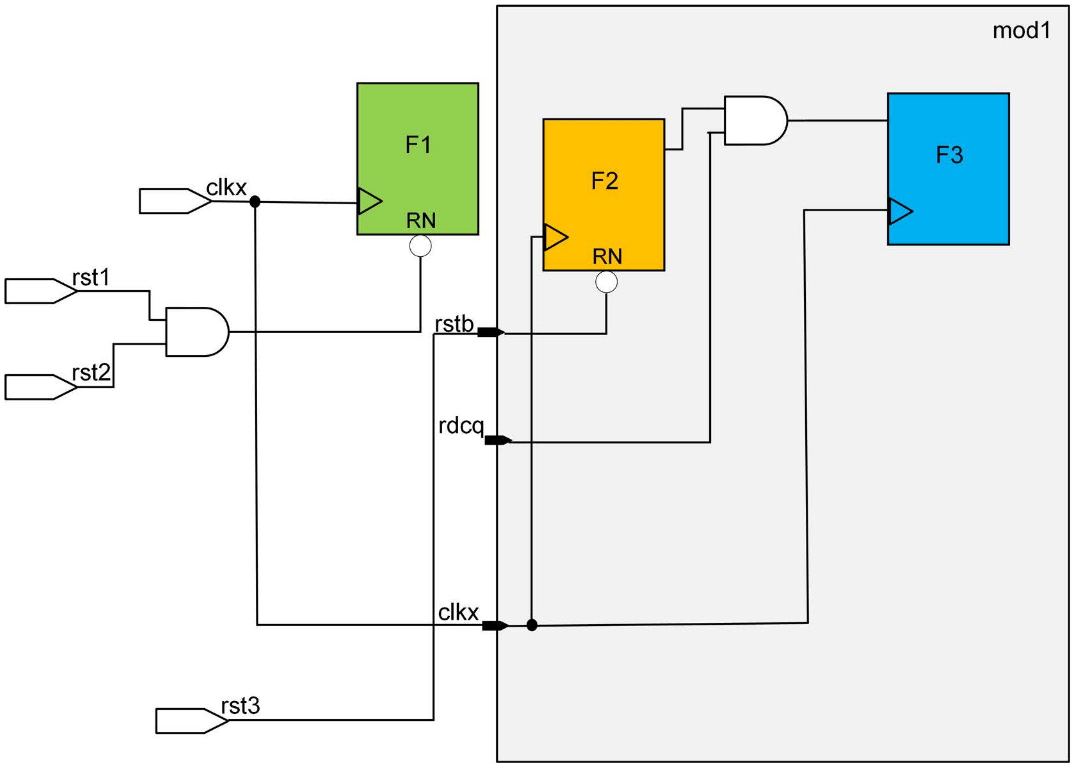


Figure 33—rstb of IP is controlling a flop that is driven by multiple resets

Scenario 3 shows the modeling of a data path in IP to support RDC interface verification. Within the IP, `cdc_set_port a` is a pure data path. To enable the RDC interface, both the driver and the receiver's reset information shall be modeled at the interface; hence, `-associated_from_reset` and `-associated_to_reset` are used. For example,

```
cdc_set_port a                                \  Name of port
    -direction input                          \  Direction of port
    -type data                                \  Type of port
    -associated_to_clocks clkx                \  Internal receiver clock
    -associated_from_reset virtual_reset_a \  The driver reset of the port
    -associated_to_reset rstb                    The receiver reset of a port
```

```
set_reset_group                    \         Definition of the reset relation
    -name reset_domain_1           \         The name of the reset domain
    -reset {virtual_reset_a; rstb}           The names of the resets
```

The `set_reset_group` attribute above indicates that RDC from `virtual_reset_a` to `rstb` is safe from metastablity issues. When the IP is integrated with the top-level logic as shown in Figure 33, the connectivity, clock domain, reset domain, clock, and reset relationship of the IP at the top level shall match the IP's modeling. As such, the following shall be checked by the tool.

a) The top level's reset and clock for `IP.a` aligns with the IP's assumption, so there will be no CDC or RDC. There shall be no RDC violation from `top.rst1` and `top.rst2` to `top.rst3` such that the following assumption shall be true at the top level.

```
cdc_set_port top.F1.Q                            \  Name of port
    -type data                                   \  Type of port
    -associated_to_clocks clkx                   \  Internal receiver clock
    -associated_from_clocks clkx                 \  External driver clock
                                                 \    domain
    -associated_from_reset top.rst1 top.rst2 \  The driver reset of the
                                                 \    port
    -associated_to_reset top.rst3                   The receiver reset of a
                                                      port
```

`top.rst3` shall belong to the same reset domain as `top.rst1` and `top.rst2`.

b) When mapped to the top level, when the driver of `virtual_reset_a` asserts, it is expected that `top.clkx` will be gated to fulfill the requirement of the IP's assumption.

```
cdc_set_port rdcq                        \        Name of port
    -direction input                     \        Direction of port
    -type rdc_control                    \        Type of port
    -rdc_data_from_reset rstb            \        Source reset of RDC (start
                                         \          point of sequential's
                                         \          reset)
    -rdc_data_to_clock clkx              \        Destination clock port of
                                         \          RDC (end point of
                                         \          sequential's clock)
    -rdc_clock_gate_location external \        Location of the clock gate
    -associated_to_clocks clkx           \          cell
    -associated_from_clock clkx          \        External driver clock domain
    -associated_from_reset virtual_reset_rdcq     The driver reset of the port
```

If the IP and top have mismatches, the tool can report them to the user for investigation during the top-level verification.

For a description of the related testcase, refer to Annex B.

5.2.4 Scenario 4

This section describes support for an RDC interface at the top level without the need to read and process the RTL of the IP shown in Figure 34. It is recommended for the IP to capture the following information in its output collateral for the top-level RDC interface verification.

```
cdc_set_port a                             \  Name of port
    -direction input                       \  Direction of port
    -type data                             \  Type of port
    -associated_from_clocks clkx           \  External driver clock domain
    -associated_from_reset virtual_reset_a \  The driver reset of the port
    -associated_to_reset rstb              \  The receiver reset of a port
    -associated_to_clocks clkx             \  Internal receiver clock
    -logic combo                              Port has combo logic on path
                                                to register
```

```
cdc_set_port rstb                \  Name of port
    -direction input             \  Direction of port
    -type async_reset            \  Type of port
    -polarity low                \  Active low asynchronous reset
    -associated_from_clocks clkx \  External driver clock domain
    -associated_to_clocks clkx      Internal receiver clock
```

```
cdc_set_port rdcq                             \  Name of port
    -direction input                          \  Direction of port
    -type rdc_control                         \  Type of port
    -rdc_data_from_reset virtual_reset_a      \  Source reset of RDC (start
                                              \    point of sequential’s
                                              \    reset)
    -rdc_data_to_reset rstb                   \  Destination reset of RDC
                                              \    (the end point of
                                              \    sequential’s reset)
    -rdc_data_to_clock clkx                   \  Destination clock port of
                                              \    RDC (end point of
                                              \    sequential’s clock)
    -rdc_clock_gate_location internal         \  Location of clock gate
                                              \    cell
    -associated_to_clocks clkx                \  Internal receiver clock
                                              \  External driver reset of
    -associated_from_reset virtual_reset_rdcq \    the port
                                              \  Ports controlled by rdcq
    -associated_inputs a
```

```
cdc_set_port clkx    \                Name of port
    -direction input \                Direction of port
    -type clock                       Type of port
```

```
cdc_set_port virtual_reset_a \       Name of port
    -direction input         \       Direction of port
    -type virtual_reset              Type of port
```

```
cdc_set_port virtual_reset_rd  \     Name of port
    -type virtual_reset              Type of port
```

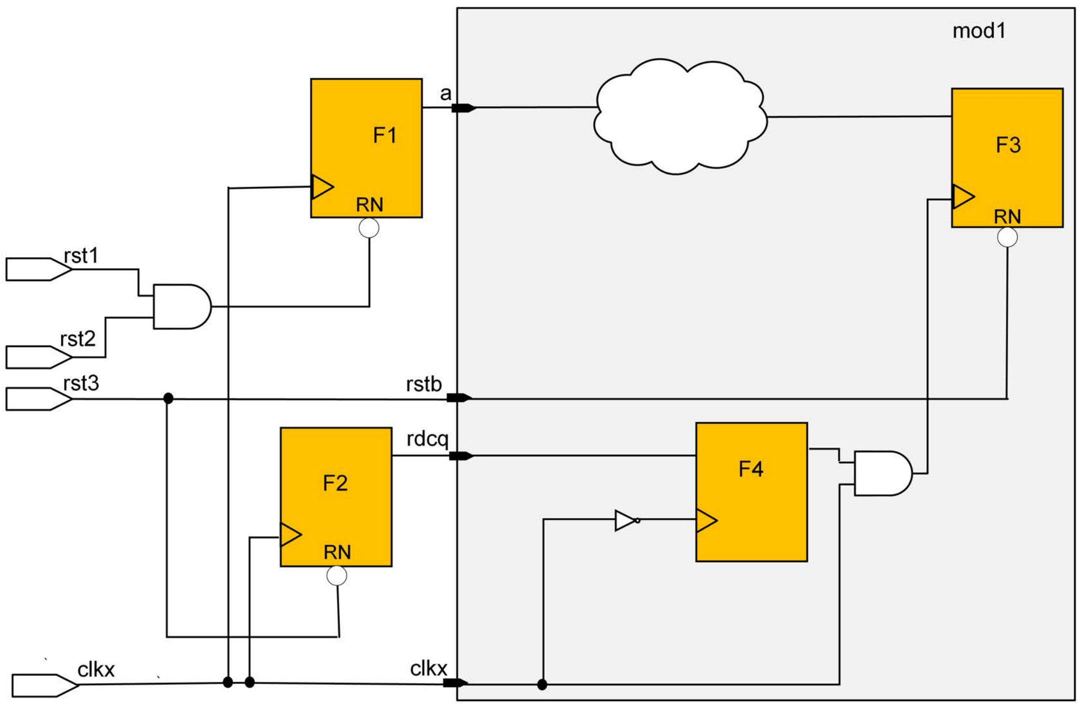


Figure 34—rdcq is gating the interface clock

Without the `rdcq` modeling shown in Figure 34, RDC from `virtual_reset_a` to `rstb` would have been a violation within the IP because there is no definition of a relationship between these two resets. The `rdcq` modeling indicates that `rdcq` will be used to block RDC for `cdc_set_port a` if the source reset is from `virtual_reset_a` and the destination reset is `rstb` where the RDC's endpoint clock will be gated with internal clock gating present in the IP.

If the same RDC control has been used by multiple ports, `-associated_input` can have a list of ports. Figure 34 shows an example where only port `a` has used the `rdc_control`.

When the IP is integrated by the top-level logic as shown in Figure 34, the connectivity, and clock domain of the IP at the top level shall match the IP's modeling. As such, the followings shall be checked by the tool.

a) The clock domain and reset domain for `IP.rstb` and `IP.rdcq` shall retain the same domains at the top level.

b) `IP.clkx` shall be gated by `rdcq` before the assertion of the top-level `rst1` or `rst2`.

For a description of the related testcase, refer to Annex B.

5.2.5 Scenario 5

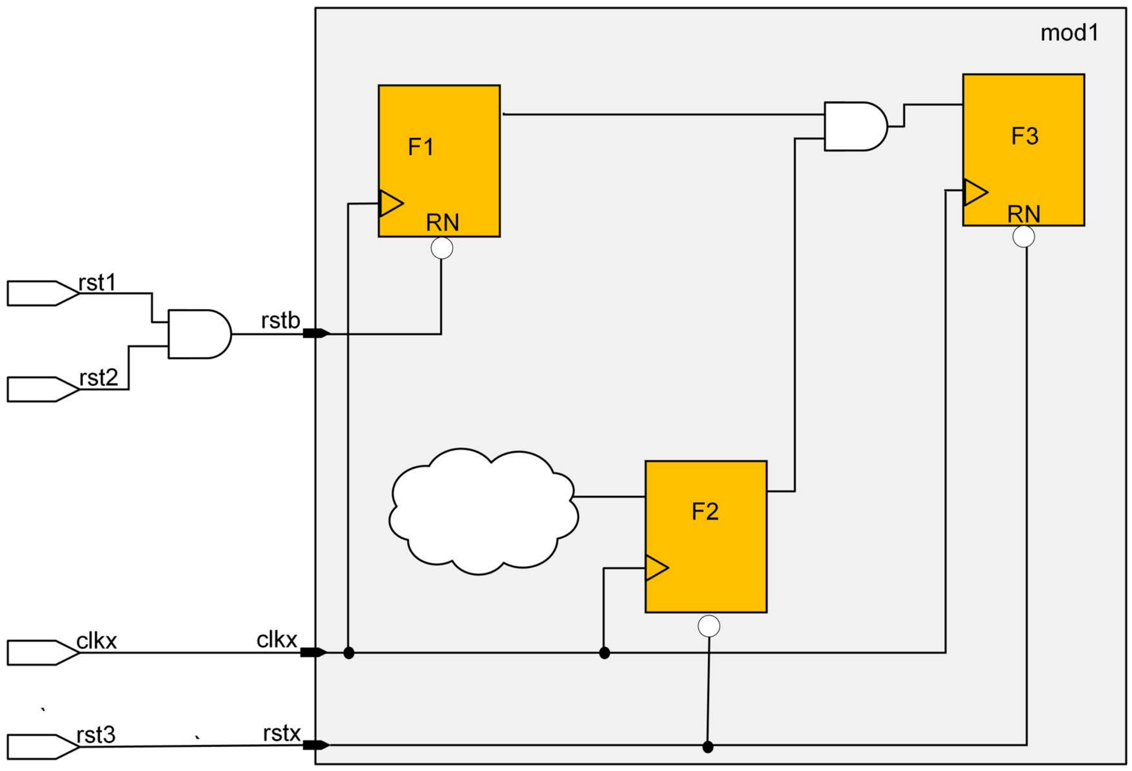


Figure 35—Internal rdcq

This scenario is not supported in the CDC v0.5 LRM. Because `F2`, which is the RDC control from `IP.rstb` to `IP.rstx` resides within the IP, there is no interface port available for the RDC interface for the top-level RDC verification. To verify this kind of design, the RDC control information shall be made available at the interface.

This will be a future focus for the CDC WG.

For a description of the related testcase, refer to Annex B.

6. CDC TCL format

This clause describes the CDC format that is based on TCL language for CDC specification from an output collateral perspective. It is assumed that users have one file per block/module in which users can specify the CDC attributes for ports of that module and users can also specify clock groups. We expect to add further commands for input collateral going forward. This clause describes the syntax and semantics for each CDC command.

NOTE—The TCL described here is case sensitive.

It is expected that EDA tools should process the CDC specification in case-sensitive TCL format and generate corresponding IP-XACT collateral. Similarly, there could be a requirement for generating TCL specification of CDC from IP-XACT.

6.1 cdc_set_module

Purpose	Indicates the module/block for which CDC specification is provided in further commands. It is specified in the beginning of a file.	
Syntax	**`cdc_set_module`** *module-name*	
Arguments	*module-name*	The name of the module for which CDC specification is being created.
Return value	Returns an empty string if successful or raises a TCL_ERROR if not.	

This command allows users to indicate the module/block name for which CDC specification is being provided in a file. The subsequent commands are applicable to the module that is specified as *module-name* in the `cdc_set_module` command.

There will be an error if *module-name* is not specified or does not exist in the design.

6.1.1 Syntax example

```
# Set a module for CDC specification
cdc_set_module ALU
# other CDC commands
...
```

6.2 cdc_set_port

Purpose	Sets the attributes of a port of a module/block.
Syntax	**`cdc_set_port`** *port-name* **`-type`** *port-type* [**`-direction`** *port-direction*] [**`-polarity`** *polarity*] …

Arguments	*port-name*	The name of the port for which CDC is specified.
	-type *port-type*	The type of a port, which can be `data`, `clock`, `async_reset`, or `cdc_control`.
	-direction *port-direction*	The direction of a port, which can be `input`, `output`, or `inout`. RTL already has direction, so it can be optional here, but it will be useful when the containing module is a black box or in the case of an inout port.
	-polarity *polarity*	Polarity is applicable for the `async_reset` port type only.
Return value	Returns an empty string if successful or raises a TCL_ERROR if not.	

This command allows users to set attributes of a port on a single line. The CDC attributes for a port are listed in the Table 15 below.

There will be an error if:

a) *port-name* is not specified or does not exist in the module.

b) a module has not been set using the `cdc_set_module` command.

c) an attribute is specified for a particular type of port that is not applicable to that type of port.

6.2.1 Syntax example

```
# Set a module
cdc_set_module ALU
# Set attributes of a port
cdc_set_port CLK -type clock -virtual_port p2 -frequency value ...

cdc_set_port RegA -type data ...

cdc_set_port RESET -type reset -polarity high ...
...
```

Table 15—CDC Port Attributes

Attribute	Type	Values	Mandatory?
Name	string	{port name}	Mandatory
Direction	defined set	{input, output, inout}	Mandatory
Type	defined set	{data, clock, async_reset, cdc_control}	Mandatory
Logic	defined set	{combo, inverter, glitch_free_combo, internal_sync}	Optional
cdc_data_from_clock	string	{clock name}	Optional

Table 15—CDC Port Attributes *(continued)*

Attribute	Type	Values	Mandatory?
associated_from_clocks	; separated list	{clock-names}	Conditional
associated_to_clocks	; separated list	{clock-names}	Conditional
associated_from_reset	; separated list	{reset-names}	Optional
associated_to_reset	; separated list	{reset-names}	Optional
associated_inputs	; separated list	{ports}	Optional
associated_outputs	; separated list	{ports}	Optional
cdc_control	; separated list	{associated-ports}	Optional
cdc_control_setup	integer	value	Conditional
cdc_control_hold	integer	value	Conditional
sampling_edge	defined set	{pos, neg}	Conditional
Polarity	defined set	{high, low, low_high}	Conditional
Ignore	defined set	{blocked, hanging}	Optional
cdc_static	; separated list	{can be any, <clocks to which it is cdc_static>}	Optional
constant	; separated list	{binary, hex, and of any length}	Optional
gray_coded	Boolean	{true, false:default}	Optional
clock_period	string	{clock period}	Optional

1 ## 6.3 cdc_set_clock_group

2

Purpose	Creates a clock group for a module/block.	
Syntax	**`cdc_set_clock_group`** [-name *clock-group-name*] -clocks {*clock-name+*}	
Arguments	**`-name`** *clock_group_name*	The name of the clock group. It is optional.
	`-clocks` *clock-name+*	TCL list consisting of one or more clock names that are synchronous to each other.
Return value	Returns an empty string if successful or raises a TCL_ERROR if not.	

3 This command allows users to set clock groups of synchronous clocks. A clock group can be specified as a
4 TCL list having one or more clock names that are synchronous.

There will be an error if one or more specified clock names do not exist in the design.

6.3.1 Syntax example

```
# Set a module
cdc_set_module ALU
# Set attributes of a port
cdc_set_port CLK -type clock -virtual_port p2 -frequency value ...

cdc_set_port RegA -type data ...

cdc_set_port RESET -type reset -polarity high ...
...
# set cdc clock group
cdc_set_clock_group -clocks {clk1 clk2}
```

6.4 cdc_set_param

Purpose	Defines a parameter within the scope of a module.	
Syntax	**`cdc_set_param`** **`-name`** *parameter-name* **`-type`** *type* [**`-value`** *value*] [**`-ignore`** *ignore*]	
Arguments	**`-name`** *parameter-name*	The name of the parameter. It is mandatory.
	`-type` *type*	Can be any of int/string/Boolean. It is optional. Default: int
	`-value` *value*	Value can be a string, number, or Boolean based on the type of the parameter.
	`-ignore` *ignore*	If true, the parameter is ignored. It flags an error if it is used in subsequent commands. By default, it is true if no value is specified.
Return value	Returns an empty string if successful or raises a TCL_ERROR if not.	

This command allows users to define parameters within the scope of a module.

There will be an error if:

a) *parameter-name* is not specified.

b) If parameter is ignored and still referred to in subsequent commands.

6.4.1 Syntax example

```
# Set a module
cdc_set_module ALU
# Set a parameter for the module
cdc_set_param -name MSB -type int -value 15
cdc_set_param -name LSB -type int -value 0
```

```
cdc_set_port RegA[MSB:LSB] -type data ...
```

7. CDC IP-XACT format

This section describes the IP-XACT format for CDC specification. It is an extension to the IP-XACT standard. The IP-XACT standard allows users to capture some attributes related to components, ports, clocks, and resets. Other attributes that are currently not part of the IP-XACT standard are defined as Vendor Extensions. IP-XACT allows extending the standard using these vendor extensions. Vendor extensions for CDC are defined as Accellera Vendor extensions at different elements in the design hierarchy. They are described in detail in the following sections.

7.1 Top-level elements

The Accellera vendor extensions for CDC allows specifying the following sets of information:

— `accellera:component`
— `accellera:wire`
— `accellera:componentInstance`

This is demonstrated in the schema diagram in Figure 36.

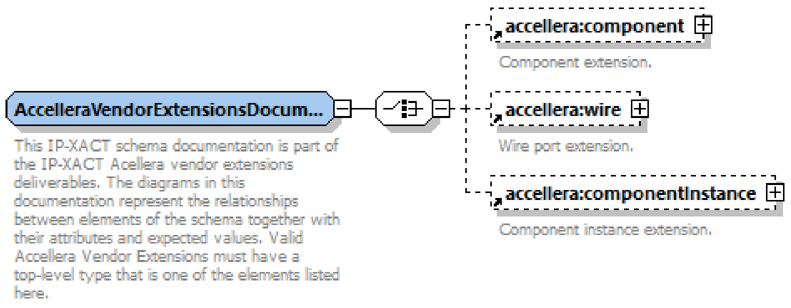

AccelleraVendorExtensionsDocum...
This IP-XACT schema documentation is part of the IP-XACT Acellera vendor extensions deliverables. The diagrams in this documentation represent the relationships between elements of the schema together with their attributes and expected values. Valid Accellera Vendor Extensions must have a top-level type that is one of the elements listed here.
accellera:component
Component extension.
accellera:wire
Wire port extension.
accellera:componentInstance
Component instance extension.

Figure 36—Schema for top-level elements

7.2 Top element—accellera:wire

The top-level element `accellera:wire` is used to define an Accellera-specific wire port extension, which contains the wire port CDC definition, `accellera-cdc:wireCDCDef`. The `wireCDCDef` element allows defining one of the CDC port types, that is, data, clock, reset, or control as shown in the schema diagram in Figure 37.

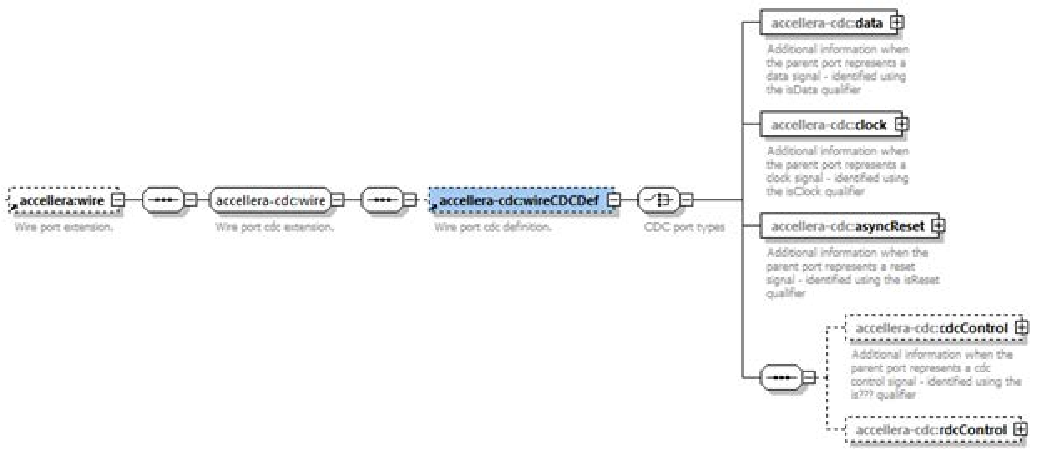

accellera:wire
Wire port extension.
accellera-cdc:wire
Wire port cdc extension.
accellera-cdc:wireCDCDef
Wire port cdc definition.
CDC port types
accellera-cdc:data
Additional information when the parent port represents a data signal - identified using the isData qualifier
accellera-cdc:clock
Additional information when the parent port represents a clock signal - identified using the isClock qualifier
accellera-cdc:asyncReset
Additional information when the parent port represents a reset signal - identified using the isReset qualifier
accellera-cdc:cdcControl
Additional information when the parent port represents a cdc control signal - identified using the is??? qualifier
accellera-cdc:rdcControl

Figure 37—Schema for accellera:wire

Each element of a `wireCDCDef` is defined as follows:

a) `accellera-cdc:data` describes the attributes of a data port type
b) `accellera-cdc:clock` describes the attributes of a clock port type
c) `accellera-cdc:asyncReset` describes the attributes of a reset port type
d) `accellera-cdc:cdcControl` describes the attributes of a CDC control port type
e) `accellera-cdc:rdcControl` describes the attributes of a CDC reset port type

7.3 Data port type—accellera-cdc:data

When a port represents a data signal, all attributes required for CDC are captured in the extension, `accellera-cdc:data`, as shown in the schema diagram in Figure 38. Each element of the `accellera-cdc:data` extension map to an attribute for the data port as defined in Clause 4, Table 1.

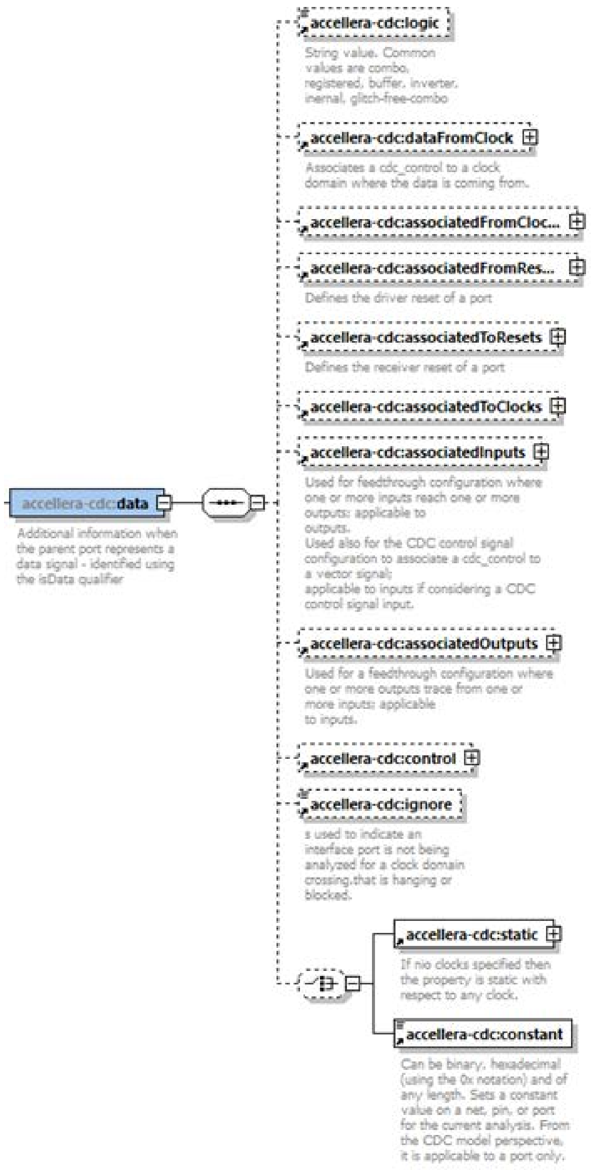

accellera-cdc:logic
String value. Common values are combo, registered, buffer, inverter, inernal, glitch-free-combo
accellera-cdc:dataFromClock
Associates a cdc_control to a clock domain where the data is coming from.
accellera-cdc:associatedFromCloc...
accellera-cdc:associatedFromRes...
Defines the driver reset of a port
accellera-cdc:associatedToResets
Defines the receiver reset of a port
accellera-cdc:associatedToClocks
accellera-cdc:associatedInputs
Used for feedthrough configuration where one or more inputs reach one or more outputs: applicable to outputs. Used also for the CDC control signal configuration to associate a cdc_control to a vector signal; applicable to inputs if considering a CDC control signal input.
accellera-cdc:data
Additional information when the parent port represents a data signal - identified using the isData qualifier
accellera-cdc:associatedOutputs
Used for a feedthrough configuration where one or more outputs trace from one or more inputs: applicable to inputs.
accellera-cdc:control
accellera-cdc:ignore
s used to indicate an interface port is not being analyzed for a clock domain crossing.that is hanging or blocked.
accellera-cdc:static
If nio clocks specified then the property is static with respect to any clock.
accellera-cdc:constant
Can be binary, hexadecimal (using the 0x notation) and of any length. Sets a constant value on a net, pin, or port for the current analysis. From the CDC model perspective, it is applicable to a port only.

Figure 38—Schema for accellera-cdc:data

7.3.1 IP-XACT code for accellera-cdc:data

```
<ipxact:port>
   <ipxact:name>i_data</ipxact:name>
   <ipxact:wire>
      <ipxact:direction>in</ipxact:direction>
      <ipxact:vector>
         <ipxact:left>7</ipxact:left>
         <ipxact:right>0</ipxact:right>
      </ipxact:vector>
   </ipxact:wire>
   <ipxact:vendorExtensions>
      <accellera-cdc:wire>
         <accellera-cdc:wireCDCDef>
```

```
            <accellera-cdc:data>
               <accellera-cdc:associatedFromClocks>
                  <accellera-cdc:clockPortReference>i_clk</accellera-
      cdc:clockPortReference>
               </accellera-cdc:associatedFromClocks>
            </accellera-cdc:data>
            </accellera-cdc:wireCDCDef>
         </accellera-cdc:wire>
      </ipxact:vendorExtensions>
   </ipxact:port>
```

7.4 Clock port type-accellera-cdc:clock

When a port represents a clock signal, the clock attributes for CDC are captured in the extension `accellera-cdc:clock` as shown in the schema diagram in Figure 39. It allows specifying the `accellera-cdc:logic` attribute in the `accellera-cdc:clock` extension.

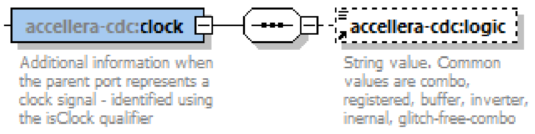

accellera-cdc:clock
Additional information when the parent port represents a clock signal - identified using the isClock qualifier
accellera-cdc:logic
String value. Common values are combo, registered, buffer, inverter, inernal, glitch-free-combo

Figure 39—Schema for accellera-cdc:clock

7.4.1 IP-XACT code for accellera-cdc:clock

```
<ipxact:port>
    <ipxact:name>i_clk</ipxact:name>
    <ipxact:wire>
        <ipxact:direction>in</ipxact:direction>
        <ipxact:qualifier>
            <ipxact:isClock>true</ipxact:isClock>
        </ipxact:qualifier>
    </ipxact:wire>
    <ipxact:vendorExtensions>
        <accellera-cdc:wire>
            <accellera-cdc:wireCDCDef>
                <accellera-cdc:clock>
                    <accellera-cdc:logic>combo</accellera-cdc:logic>
                        <accellera-cdc:clock>
            </accellera-cdc:wireCDCDef>
        </accellera-cdc:wire>
    </ipxact:vendorExtensions>
</ipxact:port>
```

7.4.2 IP-XACT code for virtual accellera-cdc:clock

```
<ipxact:port>
    <ipxact:name>i_vclk</ipxact:name>
    <ipxact:wire>
        <ipxact:direction>phantom</ipxact:direction>
        <ipxact:qualifier>
            <ipxact:isClock>true</ipxact:isClock>
        </ipxact:qualifier>
    </ipxact:wire>
    <ipxact:vendorExtensions>
        <accellera-cdc:wire>
            <accellera-cdc:wireCDCDef>
                <accellera-cdc:clock>
                    <accellera-cdc:logic>internal</accellera-cdc:logic>
                <accellera-cdc:clock>
            </accellera-cdc:wireCDCDef>
        </accellera-cdc:wire>
    </ipxact:vendorExtensions>
</ipxact:port>
```

7.5 Reset port type—accellera-cdc:asyncReset

When a port represents a reset signal, the reset attributes for CDC are captured in the extension, `accellera-cdc:asyncReset`, as shown in the schema diagram in Figure 40. In addition to the `accellera-cdc:logic` attribute in the `accellera-cdc:asyncReset` extension, there are additional available attributes; e.g., `polarity`, `associatedFromClocks`, and `associatedToClocks` as defined in Clause 4, Table 1.

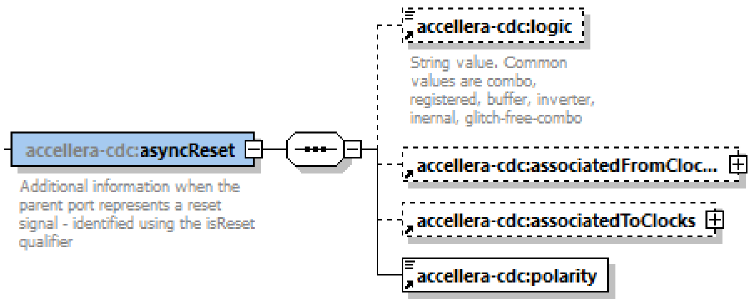

accellera-cdc:logic
String value. Common values are combo, registered, buffer, inverter, inernal, glitch-free-combo
accellera-cdc:asyncReset
Additional information when the parent port represents a reset signal - identified using the isReset qualifier
accellera-cdc:associatedFromCloc...
accellera-cdc:associatedToClocks
accellera-cdc:polarity

Figure 40—Schema for accellera-cdc:asyncReset

7.5.1 IP-XACT code for accellera-cdc:asyncReset

```
    <ipxact:port>
       <ipxact:name>i_rst</ipxact:name>
       <ipxact:wire>
          <ipxact:direction>in</ipxact:direction>
          <ipxact:qualifier>
             <ipxact:isReset>true</ipxact:isReset>
          </ipxact:qualifier>
       </ipxact:wire>
       <ipxact:vendorExtensions>
          <accellera-cdc:wire>
             <accellera-cdc:wireCDCDef>
                <accellera-cdc:asyncReset>
                   <accellera-cdc:associatedFromClocks>
                      <accellera-cdc:clockPortReference>i_clk
                      </accellera-cdc:clockPortReference>
                   </accellera-cdc:associatedFromClocks>
                 <accellera-cdc:asyncReset>
              </accellera-cdc:wireCDCDef>
           </accellera-cdc:wire>
       </ipxact:vendorExtensions>
```

```
</ipxact:port>
```

7.6 Control port type—accellera-cdc:cdcControl and accellera-cdc:rdcControl

When a port represents a CDC or RDC control signal, the corresponding attributes are captured in the extension `accellera-cdc:cdcControl` or `accellera-cdc:rdcControl` as shown in the schema diagram in Figure 41. While attributes for CDC control signals are currently defined, the RDC control attributes will be defined in a future version of the CDC LRM.

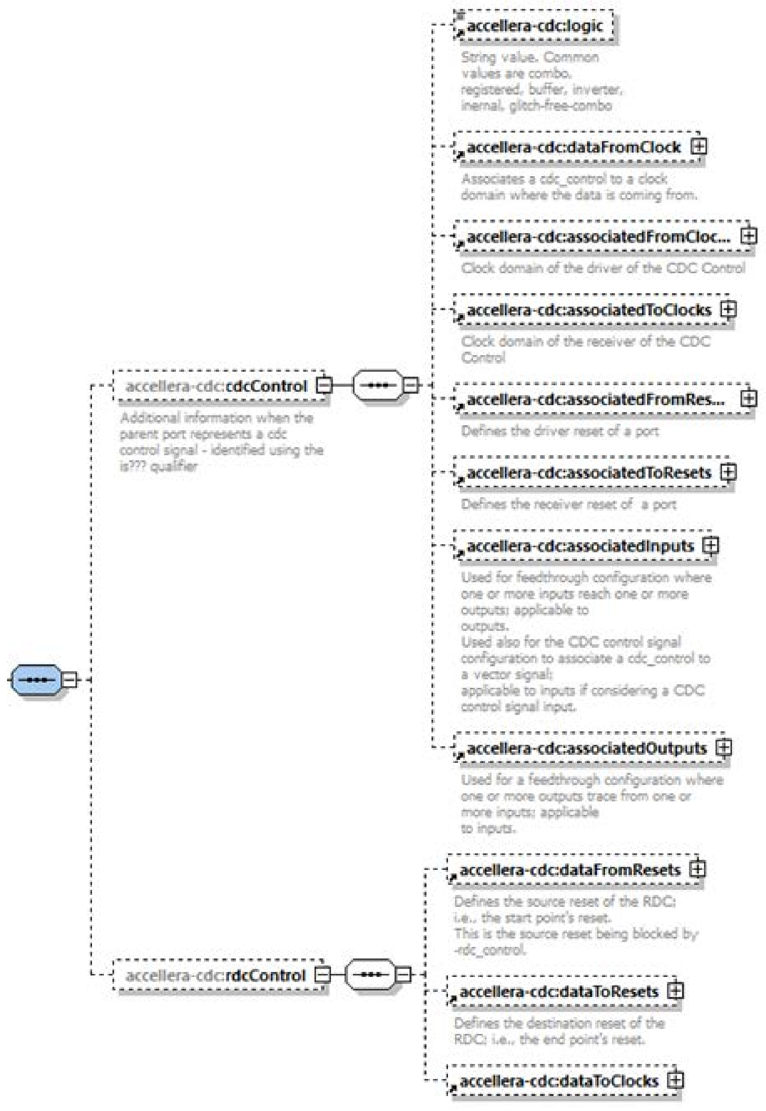

accellera-cdc:cdcControl
Additional information when the parent port represents a cdc control signal - identified using the is??? qualifier
accellera-cdc:logic
String value. Common values are combo, registered, buffer, inverter, inernal, glitch-free-combo
accellera-cdc:dataFromClock
Associates a cdc_control to a clock domain where the data is coming from.
accellera-cdc:associatedFromCloc...
Clock domain of the driver of the CDC Control
accellera-cdc:associatedToClocks
Clock domain of the receiver of the CDC Control
accellera-cdc:associatedFromRes...
Defines the driver reset of a port
accellera-cdc:associatedToResets
Defines the receiver reset of a port
accellera-cdc:associatedInputs
Used for feedthrough configuration where one or more inputs reach one or more outputs: applicable to outputs. Used also for the CDC control signal configuration to associate a cdc_control to a vector signal; applicable to inputs if considering a CDC control signal input.
accellera-cdc:associatedOutputs
Used for a feedthrough configuration where one or more outputs trace from one or more inputs: applicable to inputs.
accellera-cdc:rdcControl
accellera-cdc:dataFromResets
Defines the source reset of the RDC: i.e., the start point's reset. This is the source reset being blocked by -rdc_control.
accellera-cdc:dataToResets
Defines the destination reset of the RDC: i.e., the end point's reset.
accellera-cdc:dataToClocks

Figure 41—Schema for accellera-cdc:cdcControl and accellera-cdc:rdcControl

7.6.1 IP-XACT code for accellera-cdc:cdcControl

```
<ipxact:port>
   <ipxact:name>r_valid</ipxact:name>
   <ipxact:wire>
```

```
        <ipxact:direction>in</ipxact:direction>
     </ipxact:wire>
     <ipxact:vendorExtensions>
        <accellera-cdc:wire>
           <accellera-cdc:wireCDCDef>
              <accellera-cdc:cdcControl>
                 <accellera-cdc:controlFromClock>
                    <accellera-cdc:clockPortReference>i_clk
                    </accellera-cdc:clockPortReference>
                 </accellera-cdc:controlFromClock >
              <accellera-cdc:cdcControl>
           </accellera-cdc:wireCDCDef>
        </accellera-cdc:wire>
     </ipxact:vendorExtensions>
  </ipxact:port>
```

7.7 Control port type—accellera-cdc:componentCDCDef

A component is a top element in IP-XACT encompassing bus interfaces, memory maps, models (views and ports), power domains, parameters, etc. In CDC, there are clock groups that are associated with components. The extension `accellera-cdc:componentCDCDef` is defined to capture the clock groups of a component using the `accellera-cdc:clockGroup` extensions shown in the schema diagram in Figure 42. It contains references to clock ports or phantom ports (virtual clock) in addition to `name`, `displayName`, and `description`.

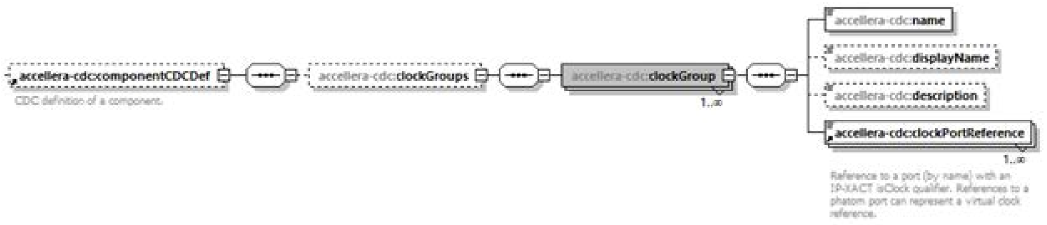

accellera-cdc:componentCDCDef
CDC definition of a component.
accellera-cdc:clockGroups
accellera-cdc:clockGroup
1..∞
accellera-cdc:name
accellera-cdc:displayName
accellera-cdc:description
accellera-cdc:clockPortReference
1..∞
Reference to a port (by name) with an IP-XACT isClock qualifier. References to a phatom port can represent a virtual clock reference.

Figure 42—Schema for accellera-cdc:componentCDCDef

7.7.1 IP-XACT code for accellera-cdc:componentCDCDef

```
  <ipxact:component>
     <!-- Specify VLNV -->
     ...
     ...
     <ipxact:vendorExtensions>
        <accellera-cdc:clockGroups>
           <accellera-cdc:clockGroup>
              <accellera-cdc:name>grp_clk</accellera-cdc:name>
                 <accellera-cdc:clockPortReference>i_clk</accellera-
     cdc:clockPortReference>
                 <accellera-cdc:clockPortReference>j_clk</accellera-
     cdc:clockPortReference>
                 <accellera-cdc:clockPortReference>k_clk</accellera-
     cdc:clockPortReference>
           </accellera-cdc:clockGroup>
        </accellera-cdc:clockGroups>
```

```
    </ipxact:vendorExtensions>
</ipxact:component>
```

7.8 Accellera CDC Vendor Extensions SCRs

Accellera IP-XACT CDC vendor extensions shall obey the semantic consistency rules listed in Table 16 to be valid.

Table 16—CDC semantic consistency rules

Name	Rule	Single doc check	Notes
`CDCPortReferenceExists`	A port reference identified by `clockPortReference`, `resetPortReference`, `inputPortReference`, or `outputPortReference` shall match the name of an existing wire port.	Yes	
`CDCClockPortReference`	A `clockPortReference` shall reference an existing port that specifies the clock qualifier.	Yes	
`CDCResetPortReference`	A `resetPortReference` shall reference an existing port that specifies the reset qualifier.	Yes	
`CDCInputPortReference`	An `inputPortReference` shall reference an existing port that specifies the data qualifier with a direction of “in”.	Yes	
`CDCOutputPortReference`	An `outputPortReference` shall reference an existing port that specifies the data qualifier with the direction of “out”.	Yes	

8. SVA Requirements for black box CDC integrity verification

8.1 Overview

Black box CDC integrity verification refers to verifying the quality of CDC integration at the SoC level where one or multiple IPs are integrated with a possibility of collateral from one or multiple EDA tools. The CDC integrity within an IP must be performed using white box methods where the IP vendor is privy to information related to the internal working of the IP and is not covered here. This clause details the requirements to perform successful CDC integration at the SoC level using IP without the knowledge of its internal working, based on the collateral provided by the IP provider. A few basic assumptions can be formulated for an IP to perform black box CDC integrity verification:

— The leaf-level IP is a black box.

— The internal working of the IP is unknown except for provided CDC collateral.

— The verification occurs at the IP port level.

A multi-bit crossing is related to an n-bit port enabled for crossing by single-bit of data synchronized either internally inside the destination IP or externally at the SoC level. SVA must be written based on data change at both of the ports. A single-bit crossing is related to a 1-bit port synchronized either internally (inside the destination IP) or externally (at the SoC level). SVA may be written for black box-to-black box single-bit crossings with external synchronization between the output port of the source IP and the input port of the destination IP. The following categories can be covered under the current version of the LRM based on available attributes.

The black box IP can be categorized as:

— Black box with external synchronization.

— Black box with internal synchronization.

The crossing type can be categorized as:

— SoC glue logic to black box, single-bit crossing.

— SoC glue logic to black box, multi-bit crossing.

— Black box to black box, single-bit crossing.

— Black box to black box, multi-bit crossing.

The crossing can further be categorized as:

— Open-loop crossing.
  1) Single-bit data crossing.
  2) Multi-bit data crossing with synchronization on the control signal.

— Closed-loop handshake data crossing.
  1) Single-bit data crossing from source domain synchronized into destination domain and one associated feedback signal from destination domain synchronized back into source domain.
  2) Multibit data crossing with control signal from source domain into destination domain and one associated feedback signal from destination domain synchronized into source domain.

— Gray-coded data crossing.
  1) Open-loop multibit data crossing.
  2) Closed-loop multibit data crossing with synchronization on gray-coded signals from both domains.

The examples based on SoC glue logic to black box crossings can also be used to signify SoC with only one IP. Black box level CDC verification cannot verify the integrity of crossings using a destination domain control signal that enables the crossing due to its invisibility outside the black box. It can be grouped under a black box with external synchronization only if the destination domain control signal is available at the interface.

8.2 Sampling edge requirement

Any data crossing from one clock domain to another clock domain requires synchronization to reduce propagation of possible metastability in the receiving clock domain. This is generally performed using an N-flop synchronizer where N signifies the number of flip flops constituting the synchronizer in the destination domain. A 2-flop synchronizer is the most-used type, with the number increasing based on design specifications or requirements. When one signal is being transmitted between clock domains through a 2-flop synchronizer, the signal should conservatively be wider than two destination clock cycles. This requirement is the same for N-flop synchronizers where N > 2.

A more relaxed requirement can be created considering the edge triggering nature of the flop. If the destination clock period is large enough, the crossing signal is required to stable for at least one and a half cycles of the destination clock period plus enough time to satisfy the setup and hold requirements of the flop. This can be generalized for an N-flop synchronizer as the crossing signal should be stable for four clock edges of the destination clock for an N-flop synchronizer. The type of activating edge of the first flop is important to determine the minimum pulse width of the crossing signal.

In black box verification with internal synchronization, the edge requirement will directly help in determining the amount of time the source data needs to be stable after the source control signal is toggled to be disabling, to guarantee sampling of correct source data in the destination domain. It will also help with data stability after the source control signal is toggled to be enabling but with more nuance depending on the speeds of clocks participating in the crossing.

The port domain attribute `-sampling_edge` is used to define the active sampling edge of the first flop of a synchronizer.

Figure 43 shows an internal synchronizer for an internal clock `dest_clk`, with an active positive edge sampling.

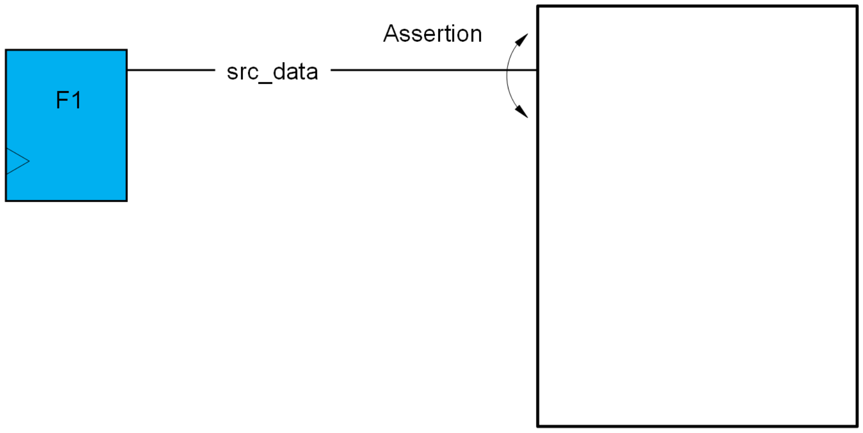


Figure 43—Sampling edge requirement example

```
cdc_set_port src_data              \
    -direction input               \
    -logic internal_sync           \
    -associated_to_clocks dest_clk \
    -sampling_edge pos
```

If the negative edge of the clock is used as an active sampling edge of the first flop of a synchronizer, then the value of `-sampling_edge` is `neg`.

8.3 Implementation headroom for a crossing

There exist timing differences between RTL and implemented gate netlists due to net and the transition delay of physical standard cells and wires as well as signal buffering in the final implementation. The timing of a signal at the domain boundary is not timed by digital design tools. During static timing analysis (STA), this path is ignored. Implementation headroom on either side of the toggling edge on paths crossing clock domain boundaries is proposed to successfully meet timing on these paths as well as verify the integration quality of a crossing.

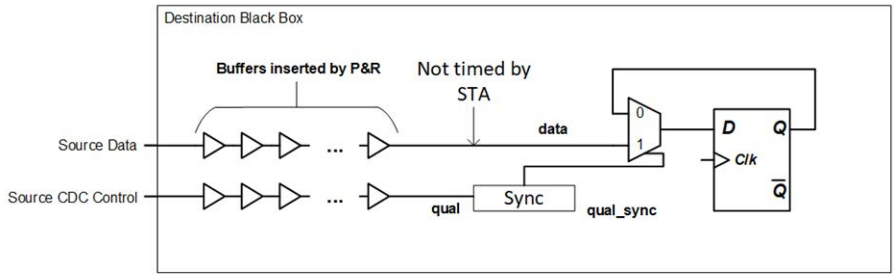


Figure 44—Example of an untimed path at the clock domain boundary

The path delay between the interface port and the sink within the destination black box will include some shift. For a multibit bus, a certain amount of skew exists between different individual signals in the bus. To accommodate this implementation headroom, the port domain provides two attributes called `cdc_control_setup` and `cdc_control_hold`.

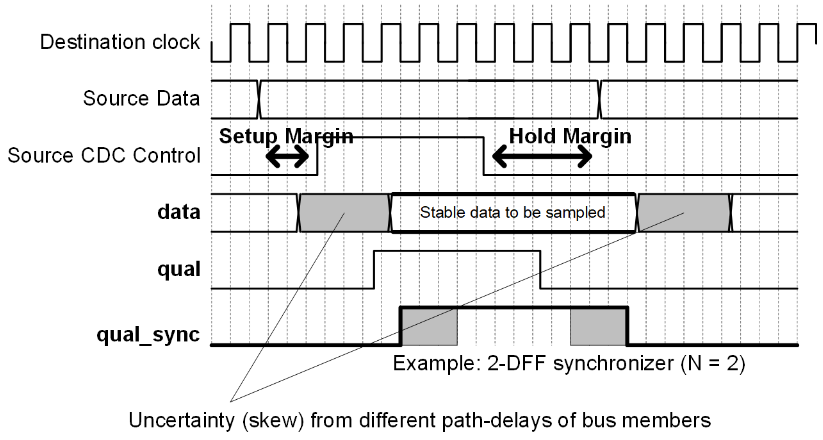

Destination clock
Source Data
Source CDC Control
Setup Margin
Hold Margin
data
Stable data to be sampled
qual
qual_sync
Example: 2-DFF synchronizer (N = 2)
Uncertainty (skew) from different path-delays of bus members

Figure 45—Implementation headroom

The setup margin of implementation headroom is defined as the amount of duration a crossing signal must be stable before its synchronization control signal enables the crossing. It is provided in the collateral using the attribute `-cdc_control_setup`. The hold margin of implementation headroom is defined as the amount of duration a crossing signal must be stable after its synchronization control signal disables the crossing. It is provided in the collateral using the attribute `-cdc_control_hold`. Refer to the example available in 4.5.

8.4 Verification clock for a crossing

Data crossings can be broadly classified into three types:

— A fast to slow crossing where the frequency of the source clock is faster than the frequency of the destination clock.

— A slow to fast crossing where the frequency of the source clock is slower than the frequency of the destination clock.

— Crossings at the same speed with possibly different phase relationships.

The requirement for any successful data crossing is correct data being sampled by the destination domain.

Synchronizing signals from a faster source clock into a slower destination clock is more problematic compared to synchronizing signals from a slower source clock into a faster destination clock. In all cases, tracking the crossing signal width is imperative for successful integration.

To verify the data stability, for dynamic verification, the most obvious choice of clock is the destination domain clock. However, it may not be the most prudent choice for verification. To make it easier for both dynamic and formal verification, it is better to use either the fastest clock available between the two clocks or create a verification-only virtual clock that can be multiple orders of magnitude faster than the fastest of two clocks participating in the crossing. Having this fastest clock will help capture glitches in source data that are smaller than the sampling clock frequency, which may occur when the crossing is enabled.

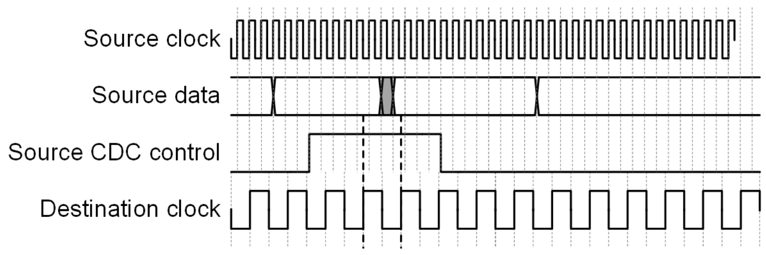


Figure 46—Necessity of a faster clock

Annex A

(informative)

Bibliography

[B1] IEEE 100, *The Authoritative Dictionary of IEEE Standards Terms,* Seventh Edition. New York: Institute of Electrical and Electronics Engineers, Inc.

Annex B

(informative)

CDC and RDC testcases

This annex lists unit testcases for early testing by EDA vendors. The list was created by the Testing SWG members. Testcases will be provided in both Verilog and VHDL formats.

.

Table 17—CDC and RDC testcases

Figure	Description	Summary
Figure 3	Port attributes: name, direction type	This testcase is for affirming that the tool in question applies the name, direction, and type attributes to the intended point of the model.
Figure 5	Specifying a virtual clock	This testcase is for affirming that the tool in question correctly applies a virtual clock.
Figure 6	Port attributes: associated to and from reset	This testcase is for affirming that the tool in question applies the `associated_from_reset` and `associated_to_reset` attributes to the intended point of the model.
Figure 9	Pseudo-static behavior at domain crossing	This testcase is for affirming that the tool in question correctly uses the `cdc_static` attribute to avoid specifying unnecessary synchronization circuitry.
Figure 11	Constant usage	This testcase is for affirming that the tool in question correctly applies the `constant` attribute.
Figure 13	Internal synch and combo logic at HDM boundary	This testcase is for affirming that the tool in question correctly uses the `logic` attribute and its various values on the associated port.
Figure 14 Figure 15	Abstract input and output ports	This testcase combines Figure 14 and Figure 15 to affirm that the tool in question properly represents a module with abstracted input and output ports as described in Figure 4.5.
Figure 23	Failure due to missing synchronizer	This testcase is for affirming that the tool in question properly flags a data port that is registered but has no synchronizer.

Table 17—CDC and RDC testcases *(continued)*

Figure	Description	Summary
Figure 24	Failure due to missing synchronized control	This testcase is for affirming that the tool in question properly flags a wide or vectored data port that is registered but has no associated synchronizer control signal. Double-registering a bus is not a proper synchronization technique.
Figure 25	Failure due to missing reset synchronizer	This testcase is for affirming that the tool in question properly flags a reset port that has no synchronizer.
Figure 26	Failure due to combo logic on clock and reset paths	This testcase is for affirming that the tool in question properly flags the issue of having a reset or clock port driven by a signal that comes from combinational logic.
Figure 27	RDC verification: internal synchronizer	This testcase is for affirming that the tool in question supports IP that requires proper external RDC-compliant interfaces.
Figure 31	CDC and RDC verification: multiple resets	This testcase is for affirming that the tool in question supports IP with resets that may be driven by multiple reset signals.
Figure 32	RDC verification: multiple resets and no reset on receiving flop	This testcase is for affirming that the tool in question supports IP with resets that may be driven by multiple reset signals and with a destination flop that has no asynchronous reset.
Figure 33	RDC verification: Receiving flop driven by multiple resets inside IP	
Figure 34	RDC verification: gating the interface clock	
Figure 35	RDC verification: internal	
Table 14, Example 1	Failure due to reset from wrong clock domain	This testcase is for affirming that the tool in question properly flags the issue of having a reset port driven by a signal that is synchronized to a different clock domain.
Table 14, Example 2	Failure due to reset wrong polarity	This testcase is for affirming that the tool in question properly flags the issue of having a reset port driven by a signal that is from the same clock domain but of the wrong polarity.
Table 14, Example 3a	Failure due to multiple clock domains but same reset domain	This testcase is for affirming that the tool in question properly flags the issue of having multiple reset ports in the same domain but of different polarities driven by the same signal.

Table 17—CDC and RDC testcases *(continued)*

Figure	Description	Summary
Table 14, Example 3b	Failure due to inverter driving one of multiple resets	This testcase is for affirming that the tool in question properly flags the issue of having multiple reset ports in the same domain driven by the same signal but with one port having inverting logic in the path.
Table 14, Example 4	Failure due to non-reset signal driving reset port	This testcase is for affirming that the tool in question properly flags the issue of having a reset port driven by a non-reset signal.