Standard for IP Abstraction for Clock and Reset Domain Crossing Integration Version 1.0

March 2026

Abstract: In-house and externally purchased IPs are often combined in SOCs. Verifying these SOCs can be challenging because various verification tools and methodologies that do not integrate are often used by different teams. 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. This standard explains the limits of the attribute set and points out the attributes that may not be supported.

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

Notices

Suggestions for improvements to the Standard for IP Abstraction for Clock and Reset Domain Crossing Integration 1.0 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.

Lee Fueng Yap, Intel Corporation, *Chair*
Sean ODonohue, Synopsys Inc, *Vice-Chair*
Farhad Ahmed, Siemens EDA, *Secretary*

Aiyush Aggarwal	Apoorv Aggarwal	Farhad Ahmed
Inayat Ali	Jeremy Anderson	Shanmuga Raja Arumugapandiyan
hamza asad	Abdelouahab Ayari	Boon Chong Ang
Jerome Avezou	Pradeep B	Mallikarjuna Badam
Serena Badran-Louca	Anupam Bakshi	Lakshmanan Balasubramanian
Luis Humberto Rezende Barbosa	Suresh Barla	Esra Sahin Basaran
Amol Bhinge	Ciro Ceissler	Chandrasekhar Challapalli
Manish Bhati	Kantha Bheemireddygari	Jean-Christophe Brignone
Sam Bueti	Haralds Capkevics	Lauren Carlson
Suman Chalana	Jeffery Chan	Ravi Teja Chatta
Shubhyant Chaturvedi	Gaurav Chhabra	Kaiwen Chin
Saransh Choudhary	David Courtright	Edwin Dankert
Lei Deng	Said Derradji	Sharvil Desai
Aparna Dey	Pawel Duc	Sylvain Duvillard
Ramu Endluri	Eldad Falik	Bill Gascoyne
Larry Geng	Nomita Goswami	Florian Guillochon
Devendra Gupta	Vivek Gupta	Megha Hansaliya
Jan Hayek	Shelly Henry	Stephen Hill
Eric Jackowski	John Selwyn Shimron Jebakumar	Sachin Jain
Vishal Jain	Yann Joanno	Abhay Kejriwal
Diana Kalel	Kenan Kaplan	Siddhant Karak
Devender Khari	Yogita Koli	Pinku Kumar

Raushan Kumar	Sunil Kumar	Sushovan Kunti
Larry Lai	Thien Le	Kay Liu
Chee Yoong Tan Lee	Dinesh M	Shivaji Magadum
Laurent Maillet-Contoz	Abhishek Mashetty	Julian Massicot
Gavin Meil	Greg Milano	Don Mills
William Mok	Sudeep Mondal	Abdul Moyeen
David Murray	Ramya Sri Murugesan	Prasad Naga
Sangeetha Sudha Nakerikanti	Prasad Nandipati	Ravindra Nibandhe
Mayank Nigam	Sean ODonohue	Iredamola Olopade
Pat Overs	Kranthi Pamarthi	Mamta Parab
Abhinav Parashar	Rahul Parashar	Shweta Pujar
Arumugapandiyan S. Raja	Sundara Rajan	Ambuja Rashinkar
ATif Razak	Abhinandan Reddy	Peter Riocreux
Sasa Ristic	Gaurav Saini	Rohit Sinha
Souradip Sarkar	Andrew Seawright	Anshuman Seth
Vineet Sharma	Konrad Sikora	Rajiv Singh
Sahil Singh	Ashish Soni	Chetan Choppali Sudarshan
Vishwanath Sundararaman	Chiara Vercesi	Jebin Vijai
Tai Vo	Joachim Voges	Jing W
David Wallace	Fengzhou Wang	Kuan-I Wu
Joseph Yackzan	Ping Yeung	Yuxin You
Jia Zhu	Xiaodong Zhuang	

At the time of standardization, the CDC WG had the following eligible voting companies:

Agnisys, Inc.	**Microsoft**
ARM Limited	**NVIDIA Corporation**
Blue Pearl Solutions	**QualcommTechnologies, Inc.**
Google LLC	**Renesas Electronics Corp.**
Infineon Technologies AG	**Siemens EDA**
Intel Corporation	**STMicroelectronics N.V.**
International Business Machines Corp	**Synopsys, Inc.**
Marvell Technology, Inc.	

At the time of standardization, the CDC WG included the following observer companies:

AMD	**Huawei Technologies Sweden AB**
Aldec	**Microchip Technologies**
AMS Osram	**NXP Semiconductors**
Apple, Inc.	**Robert Bosch GmbH**
Arteris, Inc.	**Texas Instruments**
Cadence Design Systems	**Verilab**

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 clock domain crossing (CDC) and reset domain crossing (RDC) by vendor tools used across IPs and SOCs.

The CDC/RDC Interface Standard is intended to complement existing hardware description languages (HDLs) and standard IP-XACT -based design methodologies. It addresses aspects of clock domain crossing and reset domain crossing that are outside the expressiveness of typical HDL semantics, providing a consistent and portable abstraction for CDC/RDC-related design and verification concerns.

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) Closed-boxing IP: Assumes the IP is clean and ignores checking for integration issues by closed-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 shall 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 CDC, RDC, and glitch intent enables interoperability of CDC collateral generated by any CDC verification tool.

Contents

List of figures 12
List of tables 14
List of examples 15

1. Overview 17

1.1 Scope 17
1.2 Purpose 18
1.3 Conventions used 18
1.3.1 Circuit drawing conventions 18
1.3.2 Word usage 20
1.4 Use of color in this standard 20
1.5 Contents of this standard 20

2. References 22

3. Definitions, acronyms, and abbreviations 23

3.1 Definitions 23
3.2 Acronyms and abbreviations 23

4. CDC/RDC Attributes 25

4.1 module domain attribute usage 41
4.2 parameter domain attributes usage 41
4.3 port domain attributes usage 43
4.3.1 port domain attributes: -name, -direction, and -type 43
4.3.2 port domain attribute: -associated_from_clocks 45
4.3.3 port domain attribute: -associated_to_clocks 46
4.3.4 port domain attribute: -type virtual_clock 46
4.3.5 port domain attributes: -associated_from_reset and -associated_to_reset 47
4.3.6 port domain attribute: -ignore 49
4.3.7 port domain attribute: -cdc_static 51
4.3.8 port domain attribute: -constant 53
4.3.9 port domain attribute: -associated_outputs 55
4.3.10 port domain attribute: -logic 55
4.3.11 port domain attribute: -type async_reset 57
4.4 clock_group domain attributes 63
4.5 port domain attribute: -type clock and clock_group domain attributes usage 68

5. Support for RDC Verification 71

5.1 Output Collateral Requirements for Basic RDC Scenarios 72
5.2 Scenario 1: Input interface drives internal synchronizer 72
5.3 Scenario 2: IP-level reset-sequence for safe internal RDC 73
5.4 Scenario 3: Interface RDC Control for safe internal RDC 74
5.5 Scenario 4: Interface RDC control for safe interface RDC crossing 76
5.6 Scenario 5: Output interface RDC control for safe split RDC 77
5.7 Scenario 6: Output interface prequalified for safe RDC crossing on output 79

6. CDC Tcl format 81

6.1 accellera_cdc Namespace 81
6.2 accellera_cdc::set_module 81
6.3 accellera_cdc::set_param 82
6.4 accellera_cdc::set_port 82
6.5 accellera_cdc::set_clock_group 83

7. CDC IP-XACT format 84

7.1 Top-level elements 84
7.2 Top element—accellera-cdc:wireCDCDef 84
7.3 Data port type—accellera-cdc:data 85
7.4 Clock port type—accellera-cdc:clock 87
7.5 Reset port type—accellera-cdc:asyncReset 88
7.6 Control port type—accellera-cdc:cdcControl and accellera-cdc:rdcControl 89
7.7 Control port type—accellera-cdc:componentCDCDef 91
7.8 Accellera CDC Vendor Extensions SCRs 92

8. SVA Requirements for closed box CDC integrity verification 94

8.1 Overview 94
8.2 Sampling edge requirement 95
8.3 Implementation headroom for a crossing 96
8.4 Verification clock for a crossing 97

Annex A
(informative)
CDC and RDC use case examples 99

A.1 IP packaging 99
A.2 IP Integration 99
A.3 Modeling abstracted blocks 99
A.4 Failure modes 103
A.4.1 Reset driven by the wrong clock domain 103
A.4.2 Reset driven by correct clock domain but incorrect reset polarity 104
A.4.3 Multiple ports conflicts 105
A.5 Synchronizer failure modes 109
A.5.1 Failure mode with type data due to a missing synchronizer 109
A.5.2 Failure mode due to a missing synchronized control 110
A.5.3 Failure mode due to a missing reset synchronizer 111
A.6 Failure mode due to combo logic driving clock/reset 112

Annex B
(informative)
Bibliography 114

Annex C
(informative)
CDC and RDC testcases 115

List of figures

Figure 1—Circuit drawing template ... 19
Figure 2—module domain attribute ... 41
Figure 3—port domain attributes: -name, -direction, -type ... 44
Figure 4—port domain attribute: -associated_from_clocks ... 45
Figure 5—Specifying a virtual_clock ... 47
Figure 6—port domain attributes: -associated_from_reset, -associated_to_reset ... 48
Figure 7—port domain attribute -ignore hanging for input port data1_i ... 49
Figure 8—port domain attribute -ignore blocked for input port data0_i ... 50
Figure 9—Example showing the need to describe a pseudo static behavior on the data crossing domains . 52
Figure 10—Example for -constant attribute ... 54
Figure 11—Feedthrough logic ... 55
Figure 12—Internal synchronizer and internal combinatorial logic on paths through ports ... 56
Figure 13—active_low reset on an input port with virtual -associated_from_clocks ... 59
Figure 14—active_high reset on an input port with a virtual -associated_from_clocks ... 60
Figure 15—active_high reset on an input port feeding a reset synchronizer ... 60
Figure 16—active_high reset on an input port feeding an inverter ... 61
Figure 17—active_high reset on input with -associated_to_clocks ... 62
Figure 18—One clock group ... 64
Figure 19—Two clock groups ... 65
Figure 20—Three clock groups ... 66
Figure 21—Two clock groups ... 67
Figure 22—Clock definition A ... 68
Figure 23—Clock definition B ... 69
Figure 24—Clock definition C ... 70
Figure 25—Internal synchronization logic ... 72
Figure 26—Reset assertion sequence ... 74
Figure 27—Input port modeling: Internal data/clock gating with internal data source ... 75
Figure 28— Input port modeling: Internal data/clock gating with external data source ... 77
Figure 29—Output port modeling: External data/clock gating for output port ... 78
Figure 30—Output port modeling: Internal data gating ... 80
Figure 31—Schema for top-level elements ... 84
Figure 32—Schema for accellera-cdc:wireCDCDef ... 85
Figure 33—Schema for accellera-cdc:data ... 86
Figure 34—Schema for accellera-cdc:clock ... 87
Figure 35—Schema for accellera-cdc:asyncReset ... 89
Figure 36—Schema for accellera-cdc:cdcControl and accellera-cdc:rdcControl ... 90
Figure 37—Schema for accellera-cdc:componentCDCDef ... 91
Figure 38—Sampling edge requirement example ... 96
Figure 39—Example of an untimed path at the clock domain boundary ... 97
Figure 40—Implementation headroom ... 97
Figure 41—Necessity of a faster clock ... 98
Figure 42—Module mod0 with abstracted input ports ... 101
Figure 43—Module mod0 with abstracted output ports ... 103
Figure 44—Unsynchronized reset crossing clock domains ... 104
Figure 45—Reset driven by correct clock domain but incorrect polarity ... 105
Figure 46—Failure due to reset polarity conflict ... 106
Figure 47—Failure due to inverter driving one of multiple resets ... 108
Figure 48—Failure due to non-reset signal feeding into reset signal ... 109
Figure 49—Failure mode due to a missing synchronizer ... 110
Figure 50—Failure mode due to a missing synchronized control ... 111
Figure 51—Failure mode due to a missing reset synchronizer ... 112

Figure 52—Failure mode due to combo logic driving clock/reset 113

List of tables

Table 1—CDC and RDC attributes: module domain 25
Table 2—CDC and RDC attributes: parameter domain 26
Table 3—CDC and RDC attributes: port domain 27
Table 4—CDC and RDC attributes: tool domain 36
Table 5—CDC and RDC attributes: design domain 37
Table 6—CDC and RDC attributes: clock_group domain 38
Table 7—CDC and RDC attributes: reset_group domain 39
Table 8—CDC and RDC attribute: reset_assertion_sequence domain 40
Table 9—parameter domain attributes usage 42
Table 10—port domain attributes used with async_reset 57
Table 11—CDC semantic consistency rules 92
Table 12—CDC and RDC testcases 115

List of examples

Example 1—module domain attribute usage 41
Example 2—parameter domain attributes usage 41
Example 3—port domain attributes: name, direction, type 44
Example 4—port domain attribute: -associated_from_clocks, with -associated_to_clocks 45
Example 5—port domain attribute: -associated_from_clocks, without -associated_to_clocks 45
Example 6—port domain attribute: -type virtual_clock 46
Example 7—port domain attributes: -associated_from_reset, -associated_to_reset 47
Example 8—port domain attribute: -ignore hanging 49
Example 9—Port blocked due to a constant constraint 49
Example 10—port domain attribute: -ignore blocked 49
Example 11—Describing a pseudo static behavior on the data crossing domains for data0_i 51
Example 12—Describing a pseudo static behavior on the data crossing domains for data1_i 51
Example 13—port domain attribute -constant for cksel_i 53
Example 14—port domain attribute -constant for clk2_i 53
Example 15—port domain attribute: -associated outputs 55
Example 16—Internal synchronizer on paths through ports 56
Example 17—Internal combinatorial logic on paths through ports 56
Example 18—active_low reset on an input port with virtual associated_from_clocks 59
Example 19—active_high reset on an input port with a virtual -associated_from_clocks 59
Example 20—active_high reset on an input port feeding a reset synchronizer 60
Example 21—active_high reset on an input port feeding an inverter 61
Example 22—active_high reset on input with -associated_to_clocks 61
Example 23—clock_group domain: defined in one group 63
Example 24—clock_group domain: defined in three groups 63
Example 25—clock_group domain: three clocks building a common clock group 64
Example 26—clock_group domain: defining two clock groups 65
Example 27—clock_group domain: defined in individual clock groups 66
Example 28—clock_group domain: defining non-transitive synchronicity between three clocks 67
Example 29—Clock definition A: two asynchronous clocks 68
Example 30—Clock definition B: three asynchronous clocks 69
Example 31—Clock definition C: mixture of synchronous and asynchronous clocks 70
Example 32—Internal synchronization logic 72
Example 33—Reset assertion sequence 73
Example 34— Input port modeling: Internal data/clock gating with internal data source 75
Example 35— Input port modeling: Internal data/clock gating with external data source 76
Example 36—Output port modeling: External data/clock gating for output port 78
Example 37—Output port modeling: Internal data gating 79
Example 38—Defining an accellera_cdc namespace 81
Example 39—accellera_cdc::set_module 82
Example 40—accellera_cdc::set_module 82
Example 41—accellera_cdc::set_port 83
Example 42—accellera_cdc::set_clock_group 83
Example 43—IP-XACT code for accellera-cdc:data 87
Example 44—IP-XACT code for accellera-cdc:clock 88
Example 45—IP-XACT code for virtual accellera-cdc:clock 88
Example 46—IP-XACT code for accellera-cdc:asyncReset 89
Example 47—IP-XACT code for accellera-cdc:cdcControl 91
Example 48—IP-XACT code for accellera-cdc:componentCDCDef 92
Example 49—Modeling abstracted blocks, port data0_i 100
Example 50—Modeling abstracted blocks, port cdcq0_i 100
Example 51—Modeling abstracted blocks, port data1_i 100

Example 52—Abstracted output port data1_o 101
Example 53—Abstracted output port cdcq0_o 102
Example 54—Abstracted output port data1_o 102
Example 55—Unsynchronized reset crossing clock domains 104
Example 56—Reset driven by correct clock domain but incorrect polarity 105
Example 57—Failure due to reset polarity conflict 106
Example 58—Failure due to inverter driving one of multiple resets 107
Example 59—Failure due to non-reset signal feeding into reset signal 108
Example 60—Failure mode with type data due to a missing synchronizer 109
Example 61—Failure mode with type cdc_control due to a missing synchronizer 110
Example 62—Failure mode due to a missing synchronized control 110
Example 63—Failure mode due to a missing reset synchronizer 111
Example 64—Failure mode due to combo logic driving clock/reset 112

Standard for IP Abstraction for Clock and Reset Domain Crossing Integration Version 1.0

1. Overview

Electronic Design Automation (EDA) tools claiming conformance with this standard shall support both the CDC/RDC Tcl API Tcl format and the IP-XACT-based CDC/RDC Vendor Extensions (VEs). These tools shall also ensure compliance with internal consistency constraints as specified in this standard.

This Standard for IP Abstraction for Clock and Reset Domain Crossing Integration provides the following:

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

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.

NOTE—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 CDC 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

[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.

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
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 CDC 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. Refer to the tied high and tied low legend below the figure for additional information.

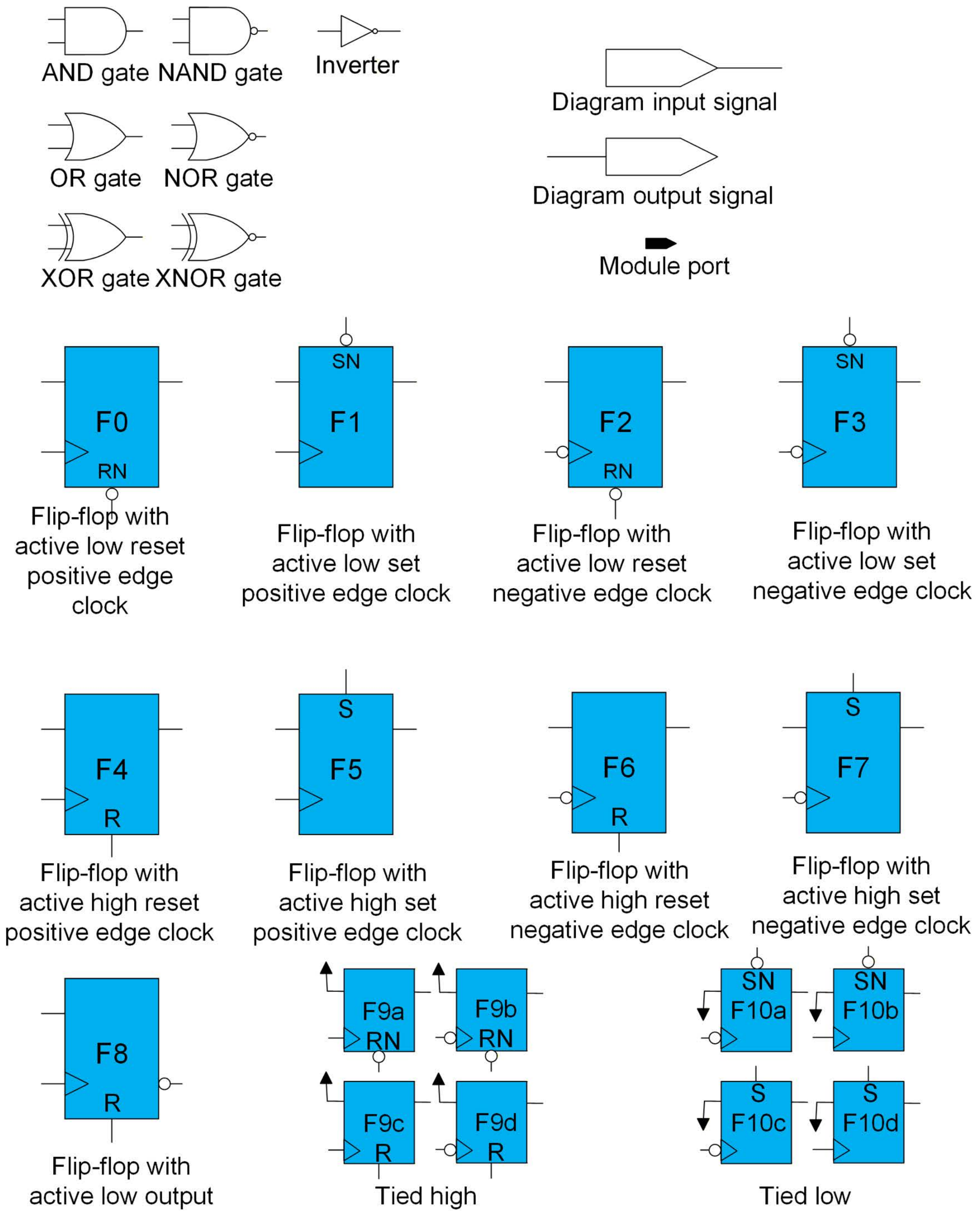
AND gate
NAND gate
Inverter
OR gate
NOR gate
XOR gate
XNOR gate
Diagram input signal
Diagram output signal
Module port
F0
RN
Flip-flop with active low reset positive edge clock
SN
F1
Flip-flop with active low set positive edge clock
F2
RN
Flip-flop with active low reset negative edge clock
SN
F3
Flip-flop with active low set negative edge clock
F4
R
Flip-flop with active high reset positive edge clock
S
F5
Flip-flop with active high set positive edge clock
F6
R
Flip-flop with active high reset negative edge clock
S
F7
Flip-flop with active high set negative edge clock
F8
R
Flip-flop with active low output
F9a
RN
F9b
RN
F9c
R
F9d
R
Tied high
SN
F10a
SN
F10b
S
F10c
S
F10d
Tied low

Figure 1—Circuit drawing template

Tied high and tied low legend:

- F9a: Tied high, flop with active low reset, positive edge clock
- F9b: Tied high, flop with active low reset, negative edge clock
- F9c: Tied high, flop with active high reset, positive clock

— F9d: Tied high, flop with active high reset, negative edge clock
— F10a: Tied low, flop with active low reset, negative edge clock
— F10b: Tied low, flop with active low reset, positive edge clock
— F10c: Tied low, flop with active high reset, negative edge clock
— F10d: Tied low, flop with active high reset, positive edge clock

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. For example, with regard to CDC, flops with a common clock domain have a common color. And with regard to RDC, flops with a common reset domain have a common color. However, in both cases, it is possible to gain an understanding of the illustration by looking at the connections. The color coding potentially facilitates a faster way to consume the meaning.

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 and RDC attributes along with their type, accepted values, conditions of usage, and clarifying comments.
— Clause 5 details output collateral requirements for IP with multiple resets from the top level and RDC control within and outside the IP.

[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.

— 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 and RDC format that is based on the IP-XACT standard.
— Clause 8 describes SVA Requirements for closed box CDC integrity verification.
— Annex A provides CDC/RDC use case examples for modeling abstracted blocks and for failure modes.
— Annex B provides an informative bibliography.
— Annex C provides a list of unit testcases for early testing by EDA vendors.

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 this Standard.

closed box: CDC and RDC verification ensures signals safely move between different clock domains without metastability or data loss. Two common approaches are closed box and glass box verification. Closed box denotes a sealed device; that is, it is only possible to observe the inputs and outputs, not the internals. In CDC and RDC, the tool does not look inside synchronizers, FIFOs, or other CDC /RDC structures. Instead, the verification tool assumes these blocks are correct if they are marked as such. In summary, only the block boundary view is needed.

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

glass box: CDC and RDC verification ensures signals safely move between different clock domains without metastability or data loss. Two common approaches are closed box and glass box verification. Glass box denotes "opening the lid" of the device; that is, inspecting every gear and spring. In CDC and RDC, this equates to the tool examining the internal RTL of synchronizers, FIFOs, and CDC logic to validate correctness. In summary, the full RTL is used for verifying both the block's internals and its connections.

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) 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

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

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 between failures
PWG	proposed working group
QoR	quality of results
RDC	reset domain crossing
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 domain crossings).

4. CDC/RDC Attributes

This standard may be applied as a standalone specification independent of HDL source code. It supports key design concepts such as instantiation and parameter-driven configurability, enabling applicability across multiple design flows and abstraction levels.

All CDC/RDC-related declarations (such as modules, ports, and parameters) must be explicitly defined and referentially consistent. Enforcement of these consistency rules is required to guarantee predictable behavior and interoperability. (Ref: Consistency Rule Section X).

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 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.

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 takes precedence.

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

Table 1—CDC and RDC 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 and RDC attributes: parameter domain

Domain	Attribute	Type	Values	Conditions of usage	Comments
parameter	**name**	string	{parameter name}	Mandatory	Defines the name of the parameter. See also: 4.2
parameter	**value**	range-list	{values}	Optional	Specifies the value of the parameter. See also: 4.2
parameter	**type**	defined set	{integer, string, Boolean*, number (hex, decimal, oct, binary)} * Boolean values 0/1 and true/false (case independent) are accepted.	Optional	Specifies type of the value stored by the parameter See also: 4.2
parameter	**ignore**	Boolean	{0 or 1, true or false} (case independent)	Optional	Tells that the parameter can be ignored. Any parameter that is ignored but still referred to in the model in following commands is a failure. See also: 4.2

Table 3 lists details of the port domain attributes.

Table 3—CDC and RDC 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	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. Type virtual_reset defines a virtual reset port that does not match an actual port in the module. See also: 4.3.1, 4.3.4, 4.3.11, and Clause 5
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

Table 3—CDC and RDC attributes: port domain *(continued)*

Domain	Attribute	Type	Values	Conditions of usage	Comments
port	**cdc_data_from_clock**	string	{clock name}	Conditional Mandatory for output ports of type cdc_control. Optional for input ports of type cdc_control. Prohibited for other port types.	Associates a cdc_control to a clock domain where the data is coming from. Supports only one clock. See also: Annex A, Figure 43
port	**associated_from_clocks**	space-separated list	{clock-names}	Conditional This attribute is mandatory for an output port unless the input port is not received by any clock domain.	See also: 4.3.2 and Annex A
port	**associated_to_clocks**	space-separated list	{clock-names}	Conditional This attribute is mandatory for an input port unless the output port is not generated from any clock domain.	See also Annex A and 5.5

Table 3—CDC and RDC attributes: port domain *(continued)*

Domain	Attribute	Type	Values	Conditions of usage	Comments
port	**associated_inputs**	space-separated list	{ports}	Optional	Used for feed-through 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. See also: Annex A
port	**associated_outputs**	space-separated list	{ports}	Optional	Used for a feed-through configuration where one or more outputs trace from one or more inputs; applicable to inputs. The cdc_control signal configuration might be useful. See also: 4.3.9
port	**cdc_control**	space-separated list	{associated-ports}	Conditional This attribute is mandatory for a data port if it is paired with a port of type cdc_control.	See also: Annex A

Table 3—CDC and RDC attributes: port domain *(continued)*

Domain	Attribute	Type	Values	Conditions of usage	Comments
port	**cdc_control_setup**	integer	value	Conditional	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 shall be stable before the cdc_control port enables the data crossing. If the value is 3, associated_inputs data port shall 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. See also: Annex A

Table 3—CDC and RDC attributes: port domain *(continued)*

Domain	Attribute	Type	Values	Conditions of usage	Comments
port	**cdc_control_hold**	integer	value	Conditional	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 shall be stable after the cdc_control port disables the data crossing. If the value is 3, associated_inputs data port shall 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. See also: Annex A

Table 3—CDC and RDC 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 verification 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 shall be described by the attribute "-logic internal_sync".	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. See also: Annex A
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. See also: Table 10
port	**blocking_polarity**	defined set	{0, 1}	Conditional This attribute is mandatory whenever an rdc_control signal is associated with a port for RDC handling.	Specifies the active value of the rdc_control signal that places the associated port into a blocking state for RDC handling. A value of 0 indicates that the port is in the blocking state when the rdc_control signal is logic 0. A value of 1 indicates that the port is in the blocking state when the rdc_control signal is logic 1.

Table 3—CDC and RDC attributes: port domain *(continued)*

Domain	Attribute	Type	Values	Conditions of usage	Comments
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	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.). See also: 4.3.8
port	**hamming1**	Boolean	{0 or 1, true or false} Default: 0 or false (case independent)	Optional	Hamming1 implies mutually exclusive signals that are part of a bus, such as, Gray coding (only 1 bit changing sequentially). Hamming1 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 a transition with a Hamming distance of 1.
port	**clock_period**	string	{clock period}	Conditional This attribute is required for generating assertions, including fast to slow assertions.	See also 8.4.

Table 3—CDC and RDC attributes: port domain *(continued)*

Domain	Attribute	Type	Values	Conditions of usage	Comments
port	**associated_from_reset**	space-separated list	{reset-names}	Conditional This attribute is required when RDC is required and for an output port.	Defines the driver reset of a port. See also: 4.3.5
port	**associated_to_reset**	space-separated list	{reset-names}	Conditional This attribute is required when RDC is required and for an input port.	Defines the receiver reset of a port. See also: 4.3.5
port	**rdc_control**	space-separated list	{associated-port}	Conditional This attribute is required for a data port that is paired with a control port of type rdc_control.	See also: 5.7
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: 5.4, 5.6
port	**rdc_data_to_reset**	space-separated list	{reset-names}	Optional	Used with rdc_control. Defines the destination reset of the RDC; i.e., the end point's reset. See also: Clause 5

Table 3—CDC and RDC 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 and RDC attributes: tool domain

Domain	Attribute	Type	Values	Conditions of usage	Comments
tool	**name**	string	{tool name}	Conditional Mandatory if it is EDA-tool-gener-ated. 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-gener-ated. N/A if user generated.	

Table 5 lists details of the design domain attributes.

Table 5—CDC and RDC 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 `clock_group` domain attributes.

Table 6—CDC and RDC attributes: clock_group domain

Domain	Attribute	Type	Values	Conditions of usage	Comments
clock_ group					A set of clocks that are synchro-nous to each other in a design. See also 4.4
	clocks	space-sep-arated list	{clock-names}	Mandatory	
	name	string	{group-name}	Optional	

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

Table 7—CDC and RDC attributes: reset_group domain

Domain	Attribute	Type	Values	Conditions of usage	Comments
reset_ group					Defines the reset relation. There are no RDC violations between resets that appear together in the same reset group.

Table 8 lists details of the `reset_assertion_sequence` domain attribute.

Table 8—CDC and RDC attribute: reset_assertion_sequence domain

Domain	Attribute	Type	Values	Conditions of usage	Comments
reset_ assertion_ sequence	**assert_sequence**	space-separated list	{reset-names}	Optional	Specifies the required order in which multiple reset signals shall be asserted.

4.1 module domain attribute usage

To define an HDL module, the syntax shown in Example 1 is used.Refer to the case-sensitive Tcl in Example 1 and Figure 2, which depicts the `mod0` module.

```
accellera_cdc::set_module -name mod0        Name of the module
```

Example 1—module domain attribute usage

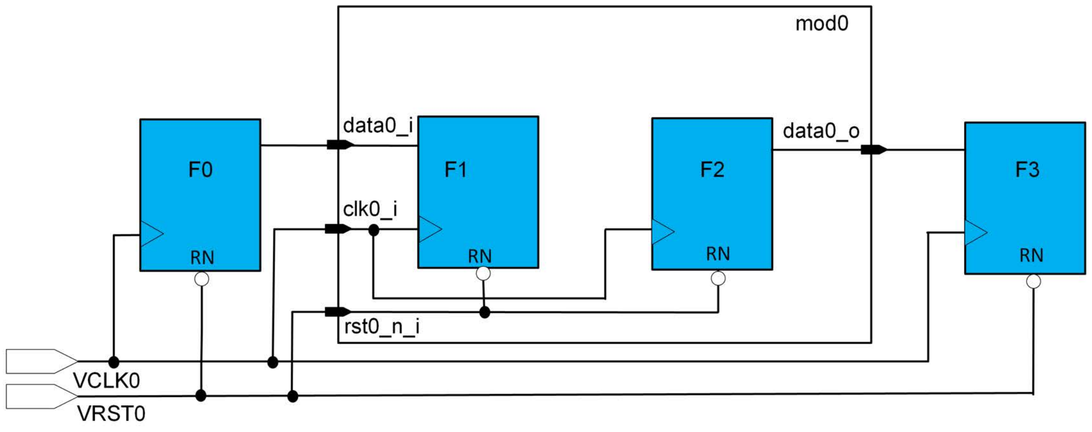

Figure 2—module domain attribute

4.2 parameter domain attributes usage

The `parameter` command shown in Example 2 is used to define parameters within the scope of the module.

```
accellera_cdc::set_param \
    -name PARAM1         \       Name of parameter
    -type int            \       Type of parameter
    -value 0;                    Value of parameter
```

Example 2—parameter domain attributes usage

Table 9 contains usage examples for the `parameter` command with the `accellera_cdc::set_param` command.

Table 9—parameter domain attributes usage

Number	Scenario	Example command

1	Sample integer type parameter with value 32	`accellera_cdc::set_param \`
		`    -name PARAM1          \`
		`    -type int             \`
		`    -value 32`

2	Sample string type parameter with value DEFAULT_CASE	`accellera_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.
		`accellera_cdc::set_param \`
		`    -name SELECT          \`
		`    -type boolean         \`
		`    -value false`
		
		or
		
		`accellera_cdc::set_param \`
		`    -name SELECT          \`
		`    -type boolean         \`
		`    -value 0`

4	A bus named DATA 7 down to 0 defined using the parameter for MSB and LSB.	`accellera_cdc::set_param \`
		`    -name MSB             \`
		`    -type int             \`
		`    -value 7`
		
		`accellera_cdc::set_param \`
		`    -name LSB             \`
		`    -type int             \`
		`    -value 0`
		
		`accellera_cdc::set_port \`
		`    -name {DATA[7:0]}    \`
		`    -type data`
		
		or
		
		`accellera_cdc::set_port \`
		`    -name {DATA[MSB:0]} \`
		`    -type data`
		
		or
		
		`accellera_cdc::set_port \`
		`    -name {DATA[7:LSB]} \`
		`    -type data`
		
		or
		
		`accellera_cdc::set_port   \`
		`    -name {DATA[MSB:LSB]} \`
		`    -type data`

Table 9—parameter domain attributes usage

Number	Scenario	Example command

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.
		`accellera_cdc::set_param \`
		`    -name MSB           \`
		`    -type int           \`
		`    -value 7`
		
		`accellera_cdc::set_param \`
		`    -name LSB           \`
		`    -type int           \`
		`    -value 0`
		
		Only the 8th bit
		`accellera_cdc::set_port \`
		`    -name {DATA[MSB]}   \`
		`    -type data`
		Only the first four MSB bits
		`accellera_cdc::set_port     \`
		`    -name {DATA[MSB:MSB-3]} \`
		`    -type data`
		
		Only the last 2 LSB bits
		`accellera_cdc::set_port     \`
		`    -name {DATA[LSB+1:LSB]} \`
		`    -type data`
		
		3-bit slice from the middle of the bus
		`accellera_cdc::set_port       \`
		`    -name {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	`accellera_cdc::set_param \`
		`    -name SEL_VAL       \`
		`    -type boolean       \`
		`    -value true`
		
		`accellera_cdc::set_port \`
		`    -name P1            \`
		`    -constant SEL_VAL`

4.3 port domain attributes usage

Several examples of the port domain's attributes are detailed in this section.

4.3.1 port domain attributes: -name, -direction, and -type

Refer to the case-sensitive Tcl shown in Example 3 and the diagram in Figure 3, which depict the `name`, `direction`, and `type` attributes. Also see 4.3.11 for additional information about the `type` attribute. If the name does not include a range, it applies to all bits of the port.

```
accellera_cdc::set_port -name data0_i    \      Name of port
    -direction input                     \      Direction of port
    -type data                           \      Type of port
    -associated_from_clocks virtual_clk \       External driver’s clock
    -associated_to_clocks clk0_i                Internal receiver clock
```

Example 3—port domain attributes: name, direction, type

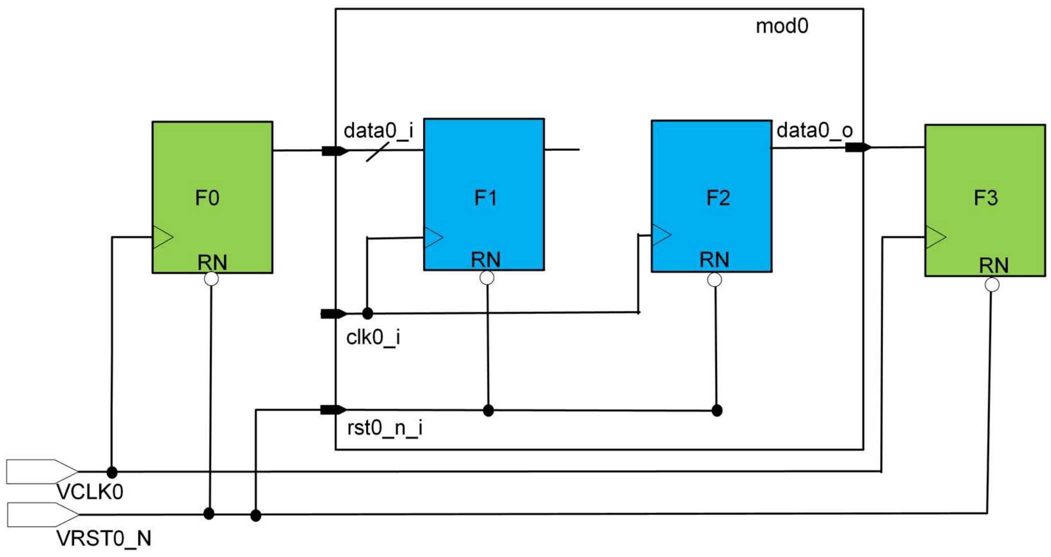

Figure 3—port domain attributes: -name, -direction, -type

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

4.3.2 port domain 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. For example, if `associated_from_clocks` is `clk0` and `associated_to_clocks` is `clk1`, and if `clk0` and `clk1` are asynchronous to each other, an appropriate synchronization scheme is expected in the receiving circuit to avoid a CDC violation (see Example 4). However, if `associated_to_clocks` is not defined, as shown in Example 5, the receiving clock is found in the actual encompassing design. If it is asynchronous to `clk0`, an appropriate synchronization scheme is expected in the receiving circuit to avoid a CDC violation.

```
accellera_cdc::set_port -name data0_i \      Name of port
    -direction input                  \      Direction of port
    -type data                        \      Type of port
    -associated_from_clocks clk0_i    \      External driver clock
    -associated_to_clocks clk0_i             Internal receiver clock
```

Example 4—port domain attribute: -associated_from_clocks, with -associated_to_clocks

```
accellera_cdc::set_port -name data0_o \   Name of port
    -direction output                 \   Direction of port
    -type data                        \   Type of port
    -associated_from_clocks clk0_i        Internal driver clock
```

Example 5—port domain attribute: -associated_from_clocks, without -associated_to_clocks

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

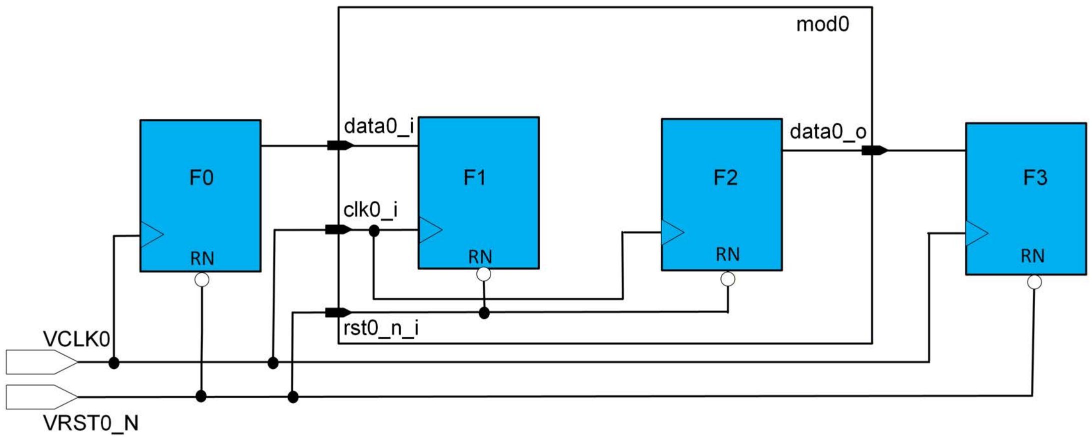

Figure 4—port domain attribute: -associated_from_clocks

4.3.3 port domain attribute: -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. For example, if `-associated_to_clocks` is `clk0_i` and `-associated_from_clocks` is `clk1_i`, and if `clk0_i` and `clk1_i` are asynchronous to each other, an appropriate synchronization scheme is expected in the receiving circuit to avoid a CDC violation. However, if `-associated_from_clocks` is not defined, the input port's driving clock is found in the actual encompassing design. If it is asynchronous to `clk0_i`, an appropriate synchronization scheme is expected in the CDC path to avoid a CDC violation.

4.3.4 port domain attribute: -type virtual_clock

Virtual clocks shall be explicitly declared by defining a port with the type `virtual_clock`. This is a virtual port that does not match an actual port at the module interface. For example, `vclk0_o` used in Figure 5 to define the receiving clock of port `data0_i` shall be declared with type `virtual_clock` as shown in Example 6. If the virtual clock is internal to the block, define its direction as `output`. If it is external to the block, define its direction as `input`.

```
accellera_cdc::set_port -name vclk0_o \     Name of port
    -direction output                 \     The virtual clock is in the IP. Define it as “output”.
    -type virtual_clock                     Type of port

accellera_cdc::set_port -name data0_i \     Name of port
    -direction input                  \     Direction of port
    -type data                        \     Type of port
    -associated_to_clocks vclk0_o     \     Internal receiver clock
    -logic internal_sync                    Internal synchronizer

accellera_cdc::set_port -name rst0_n_i \    Name of port
    -direction input                   \    Direction of port
    -type async_reset                  \    Type of port
    -associated_to_clocks vclk0_o      \    Internal receiver clock
    -logic internal_sync                    Internal synchronizer
```

Example 6—port domain attribute: -type virtual_clock

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

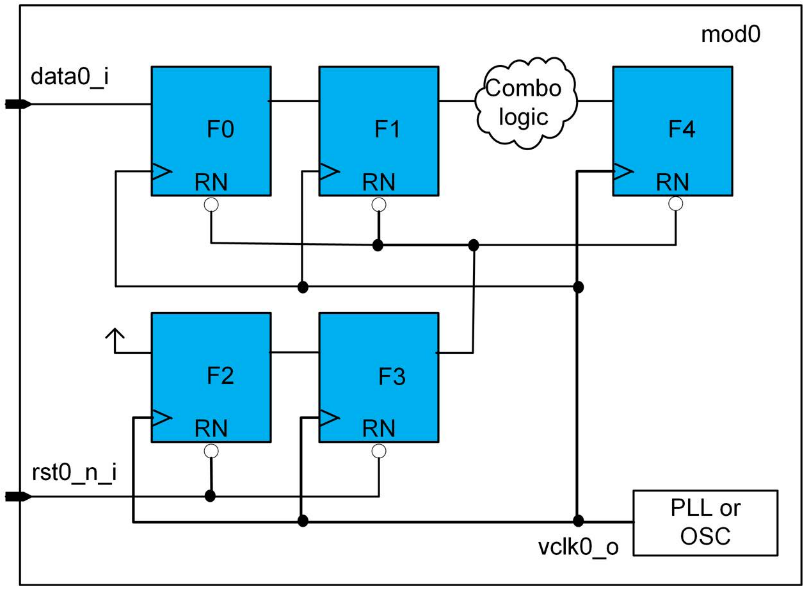

Figure 5—Specifying a virtual_clock

4.3.5 port domain attributes: -associated_from_reset and -associated_to_reset

The case-sensitive Tcl shown in Example 7 and the accompanying Figure 6 depict the optional `-associated_from_reset` and `-associated_to_reset` attributes.

```
accellera_cdc::set_port -name data0_i \    Name of port
    -direction input                  \    Direction of port
    -type data                        \    Type of port
    -associated_to_clocks clk0_i      \    Internal receiver clock
    -associated_to_reset rst0_n_i          Internal receiver reset

accellera_cdc::set_port -name data0_o \    Name of port
    -direction output                 \    Direction of port
    -type data                        \    Type of port
    -associated_from_clocks clk0_i    \    External driver clock
    -associated_from_reset rst0_n_i        The internal reset of a port
```

Example 7—port domain attributes: -associated_from_reset, -associated_to_reset

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

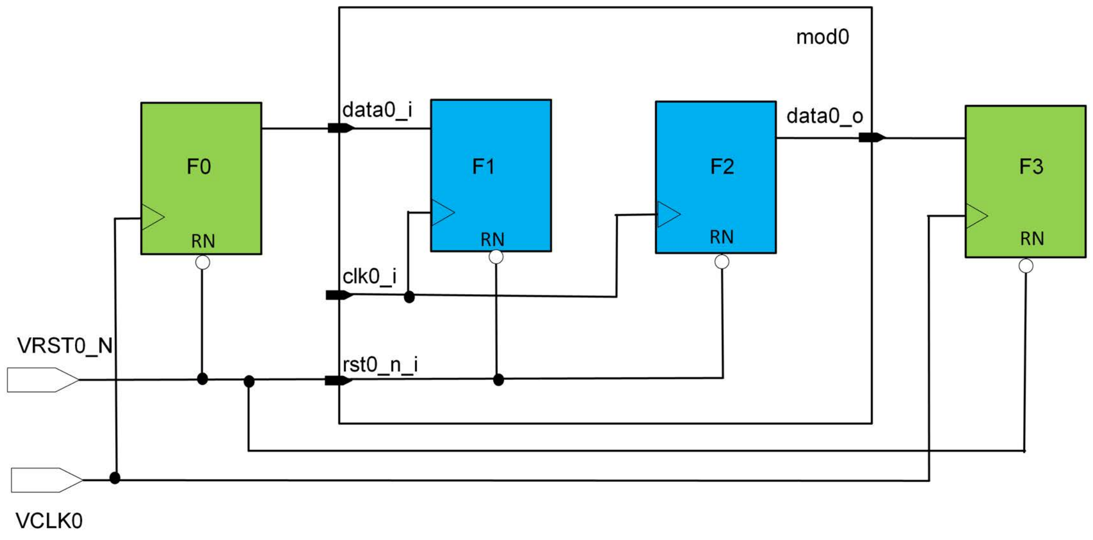

Figure 6—port domain attributes: -associated_from_reset, -associated_to_reset

4.3.6 port domain 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` (see Example 8). For example, the `hanging` value is used when an input port is dangling as shown in Figure 7.

`accellera_cdc::set_port -name data1_i \`	Name of port
`    -ignore hanging`	No analysis for CDC that is hanging

Example 8—port domain attribute: -ignore hanging

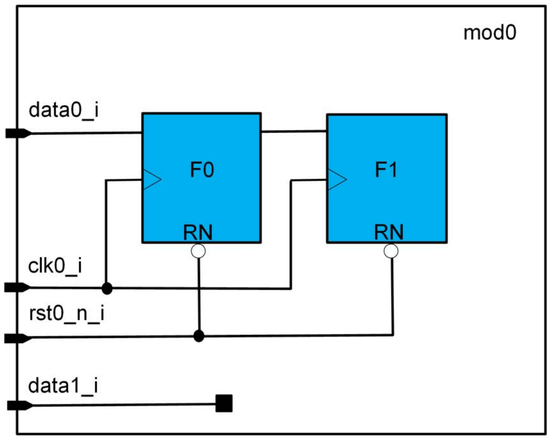

Figure 7—port domain attribute -ignore hanging for input port data1_i

The `blocked` value is used when the propagation of an interface port is blocked as shown in Figure 8. `indata0_i` in Figure 8 is blocked due to a constant constraint at `accellera_cdc::set_port mode_i` (see Example 9). RTL tie-off can also cause an interface port to be `blocked`, and therefore, ignored (see Example 10). 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.

`accellera_cdc::set_port -name mode_i \`	Name of port
`    -constant 0`	Constant value is 0

Example 9—Port blocked due to a constant constraint

`accellera_cdc::set_port -name data0_i \`	Name of port
`    -ignore blocked`	No analysis for CDC that is blocked

Example 10—port domain attribute: -ignore blocked

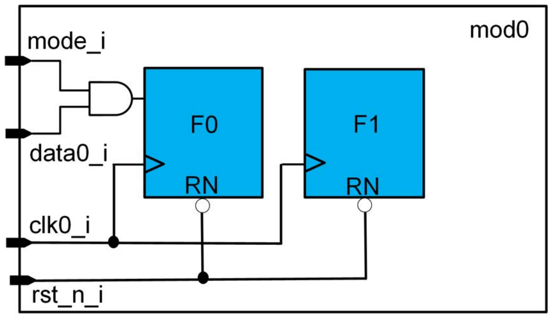

Figure 8—port domain attribute -ignore blocked for input port data0_i

4.3.7 port domain attribute: -cdc_static

This section discusses the need and usage 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, mod2 only works on clk0_i. mod1 has a clock gating block on the clock path driving mod2 and a data gating block on the data path for the incoming ports data0_i and data1_i. If it is guaranteed by the design that mod1's clk0_i shall always be gated before the driver for port data0_i, and port data1_i changes (i.e., port data0_o and port data1_o of mod0), the -cdc_static attribute can be applied to port data0_i and port data1_i during mod1's CDC analysis. Hence, CDC analysis shall not be done on these primary interface ports of mod1. It is important to have all -cdc_static attributes validated in the assertion flow to ensure the functionality of the design. mod1's internal nets datacond and clkcond illustrated in Figure 9 are not supported because this LRM focuses on capturing the interface information. However, mod1'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 data0_i and port data1_i.

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.

See Example 11 and Example 12 for usage based on mod0 of Figure 9:

```
accellera_cdc::set_port -name data0_i \   Name of port
    -direction input                  \   Direction of port
    -cdc_static clk0_i                    Gated clock
```

Example 11—Describing a pseudo static behavior on the data crossing domains for data0_i

```
accellera_cdc::set_port -name data1_i \   Name of port
    -direction input                  \   Direction of port
    -cdc_static clk0_i                    Gated clock
```

Example 12—Describing a pseudo static behavior on the data crossing domains for data1_i

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

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

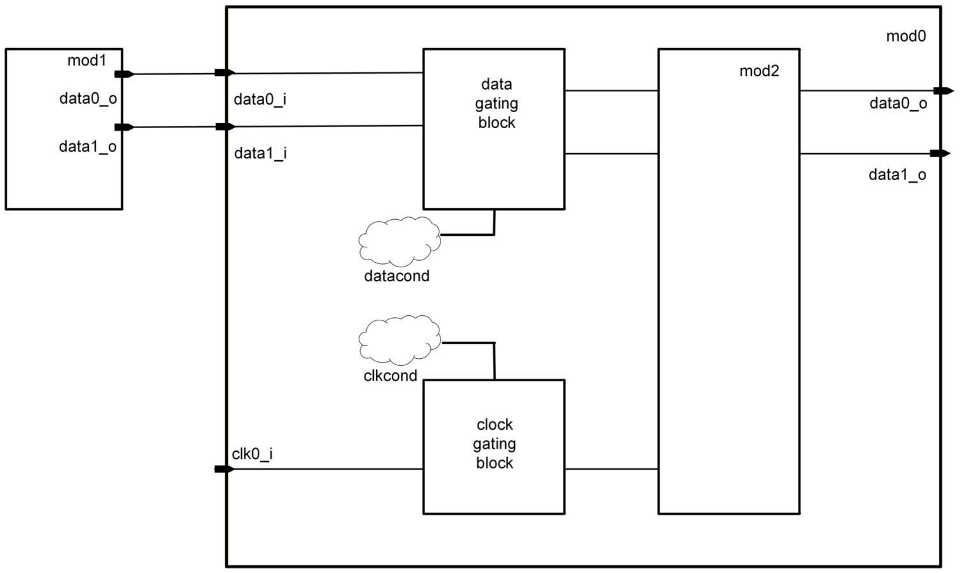

Figure 9—Example showing the need to describe a pseudo static behavior on the data crossing domains

4.3.8 port domain attribute: -constant

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

Figure 10 below shows two different flops in two asynchronous clock domains. When the clock select (`clksel_i`) is set to 0, then the clocks `clk0_i` and `clk1_i` are routed to their respective flops, which indicates a clock domain crossing violation on the data crossing between `F0` and `F1`. Conversely, when the clock select (`clksel_i`) is set to 1, clock `clk2_i` is routed to both the flops, resulting in no clock domain crossing in the data crossing between `F0` and `F1`. Hence, when verifying for the mode when the flops need to be clocked with `clk0_i` and `clk1_i`, we can safely mark both `clksel_i` and `clk2_i` as constants, which are then ignored during CDC analysis.

Using the example in Figure 10, for a CDC analysis for the mode when the flops need to be clocked with `clk0_i` and `clk1_i`, the ports `clksel_i` and `clk2_i` can be declared as constants using the constant attribute. See Example 13 and Example 14.

```
accellera_cdc::set_port -name clksel_i \     Name of port
    -direction input                   \     Direction of port
    -constant 0                              Constant value is 0
```

Example 13—port domain attribute -constant for cksel_i

```
accellera_cdc::set_port -name clk2_i \    Name of port
    -direction input                 \    Direction of port
    -constant 0                           Constant value is 0
```

Example 14—port domain attribute -constant for clk2_i

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 C.

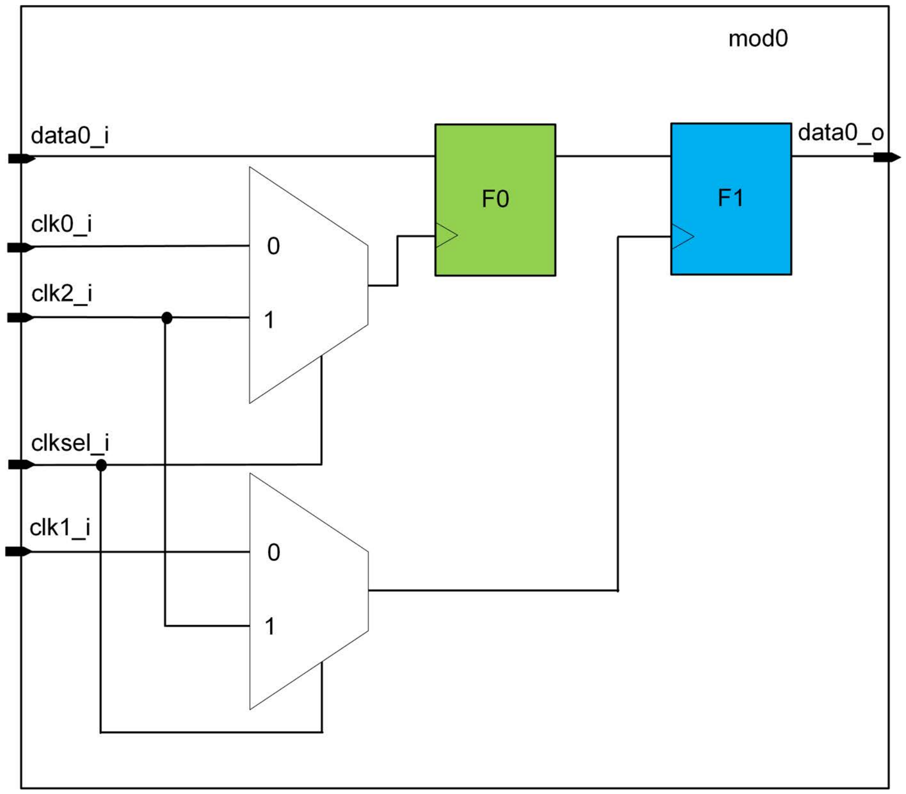

Figure 10—Example for -constant attribute

4.3.9 port domain attribute: -associated_outputs

This section discusses the need and usage for the `-associated_outputs` attribute. Figure 11 depicts feedthrough logic. The associated case-sensitive Tcl shown in Example 15 includes the optional attribute `-associated_outputs` for the feedthrough output.

```
accellera_cdc::set_port -name data0_i \      Name of port
    -type data                        \      Type of port
    -direction input                  \      Direction of port
    -associated_outputs data0_o              Output port controlled by data0_o
```

Example 15—port domain attribute: -associated outputs

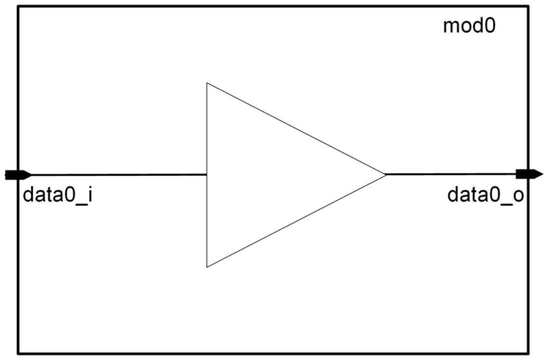

Figure 11—Feedthrough logic

4.3.10 port domain 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 (see Example 16).

— `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 (see Example 17).

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

```
accellera_cdc::set_port -name data0_i \   Name of port
    -direction input                  \   Direction of port
    -type data                        \   Type of port
    -associated_to_clocks clk0_i      \   Internal receiver clock
    -associated_to_reset rst0_n_i     \   Internal receiver reset
    -logic internal_sync                  Internal synchronizer
```

Example 16—Internal synchronizer on paths through ports

```
accellera_cdc::set_port -name data0_o \   Name of port
    -direction output                 \   Direction of port
    -type data                        \   Type of port
    -associated_from_clocks clk0_i    \   External driver clock
    -associated_from_reset rst0_n_i   \   The internal reset of a port
    -logic glitch_free_combo              The path through the port contains combinatorial
                                          logic that does not generate glitches
```

Example 17—Internal combinatorial logic on paths through ports

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

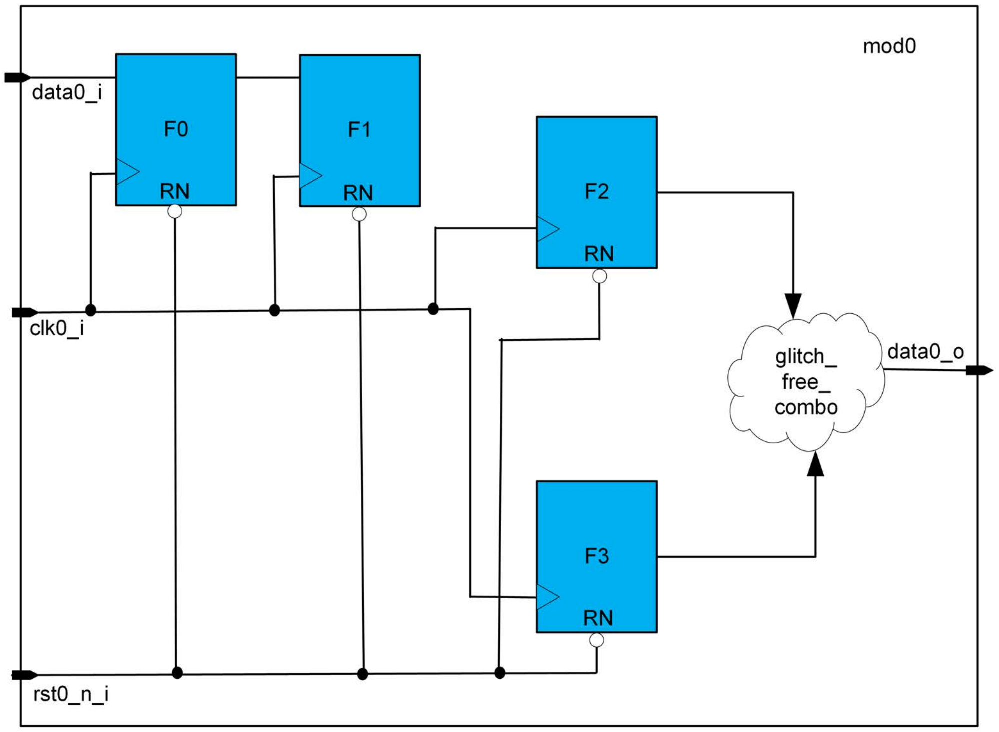

Figure 12—Internal synchronizer and internal combinatorial logic on paths through ports

4.3.11 port domain attribute: -type async_reset

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

4.3.11.1 Supported async_reset attributes

If the `port` command's `-type` attribute has the value `async_reset`, the attributes in Table 10 will be applicable and the rest of the attributes of the `port` command can be ignored.

Table 10—port domain attributes used with async_reset

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.
`-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 shall match.

Table 10—port domain attributes used with async_reset

Attribute	Values	Definition
`-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 shall match.
`-logic`	`{combo, glitch_free_combo, internal_sync}`	`combo`: Signifies that there is combo logic just inside the port. `glitch_free_combo`: `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. If the `logic` attribute is not provided in the code, the port is assumed to be connected to a sequential component directly. Defaults: — `combo`: the default is *combo 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.

4.3.11.2 Example async_reset cases

This section provides example async_reset cases with sample commands and supporting diagrams.

4.3.11.2.1 active_low reset on an input port with virtual associated_from_clocks

An active_low reset on an input port with a virtual `associated_from_clocks` means that the signal driving `rst0_n_i` is asynchronous to `clk0_i`. With this modeling, the IP will have an internal reset deassertion violation. If the signal driving `rst0_n_i` is from `clk0_i`, the `associated_from_clocks` should be `clk0_i`. Example 18 provides sample commands, and Figure 13 provides a diagram for this case.

```
accellera_cdc::set_module -name mod0          Name of module

accellera_cdc::set_port -name rst0_n_i \      Name of port
    -direction input                   \      Direction of port
    -type async_reset                  \      Type of port
    -polarity low                      \      Active low asynchronous reset
    -associated_from_clocks vclk0      \      External driver clock
    -associated_to_clocks clk0_i              Internal receiver clock
```

Example 18—active_low reset on an input port with virtual associated_from_clocks

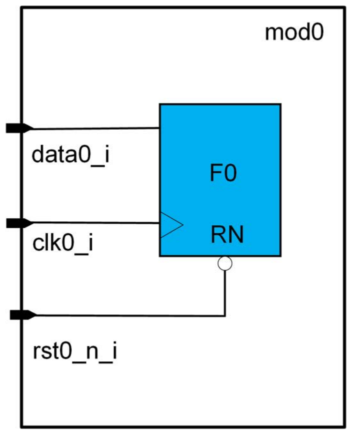

Figure 13—active_low reset on an input port with virtual -associated_from_clocks

4.3.11.2.2 active_high reset on an input port with virtual -associated_from_clocks

An active_high reset on an input port with a virtual `-associated_from_clocks` means that the signal driving `rst0_n_i` is asynchronous to `clk0_i`. With this modeling, the IP will have an internal reset deassertion violation. If the signal driving `rst0_n_i` is from `clk0_i`, the `-associated_from_clocks` should be `clk0_i`. Example 19 provides sample commands, and Figure 14 provides a diagram for this case.

```
accellera_cdc::set_module -name mod0          Name of module

accellera_cdc::set_port -name rst0_i \        Name of port
    -direction input                 \        Direction of port
    -type async_reset                \        Type of port
    -polarity high                   \        Active high asynchronous reset
    -associated_to_clocks clk0_i     \        Internal receiver clock
    -associated_from_clocks vclk0             External driver clock
```

Example 19—active_high reset on an input port with a virtual -associated_from_clocks

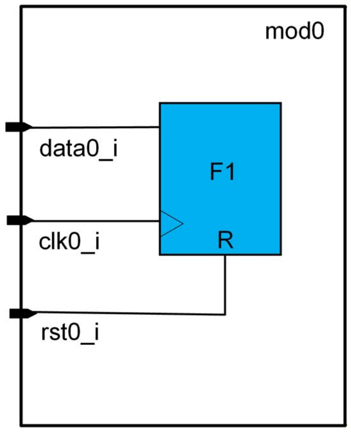

Figure 14—active_high reset on an input port with a virtual -associated_from_clocks

4.3.11.2.3 active_high reset on an input port feeding a reset synchronizer

This section describes an active_high reset on an input port feeding a reset synchronizer. Example 20 provides sample commands, and Figure 15 provides a diagram for this case.

```
accellera_cdc::set_module -name mod0       Name of module

accellera_cdc::set_port -name rst0_i \     Name of port
    -direction input                 \     Direction of port
    -type async_reset                \     Type of port
    -polarity high                   \     Active high asynchronous reset
    -logic internal_sync             \     Internal synchronizer
    -associated_to_clocks clk0_i           Internal receiver clock
```

Example 20—active_high reset on an input port feeding a reset synchronizer

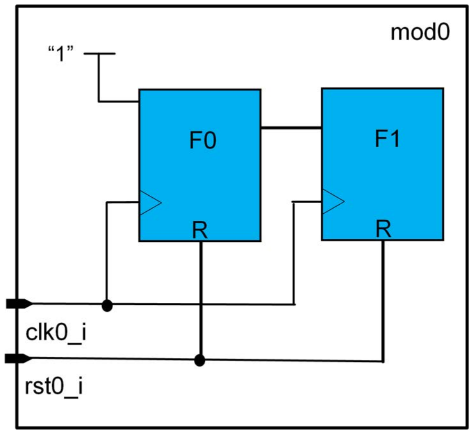

Figure 15—active_high reset on an input port feeding a reset synchronizer

4.3.11.2.4 active_high reset on an input port feeding an inverter

This section describes an active_high reset on an input port that is feeding an inverter. Example 21 provides sample commands, and Figure 16 provides a diagram for this case.

```
accellera_cdc::set_module -name mod0        Name of module

accellera_cdc::set_port -name rst0_i \      Name of port
    -direction input                 \      Direction of port
    -type async_reset                \      Type of port
    -polarity high                   \      Active high asynchronous reset
    -logic inverter                  \      The signal is inverted
    -associated_to_clocks clk0_i            Internal receiver clock
```

Example 21—active_high reset on an input port feeding an inverter

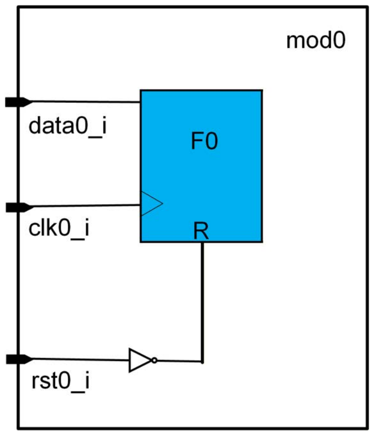

Figure 16—active_high reset on an input port feeding an inverter

4.3.11.2.5 active_high reset on input with -associated_to_clocks

This section describes an active_high reset on an input port with `-associated_to_clocks`. Example 22 provides sample commands, and Figure 17 provides a diagram for this case.

```
accellera_cdc::set_module -name mod1        Name of module

accellera_cdc::set_port -name vclk0_o \     Name of port
    -direction input                  \     Direction of port
    -type virtual_clock                     Type of port

accellera_cdc::set_port -name rst0_i    \   Name of port
    -direction input                    \   Direction of port
    -type async_reset                   \   Type of port
    -polarity high                      \   Active high asynchronous reset
    -associated_to_clocks clk0_i            Internal receiver's clock for multiple
                                            fanouts of port rst0_i
```

Example 22—active_high reset on input with -associated_to_clocks

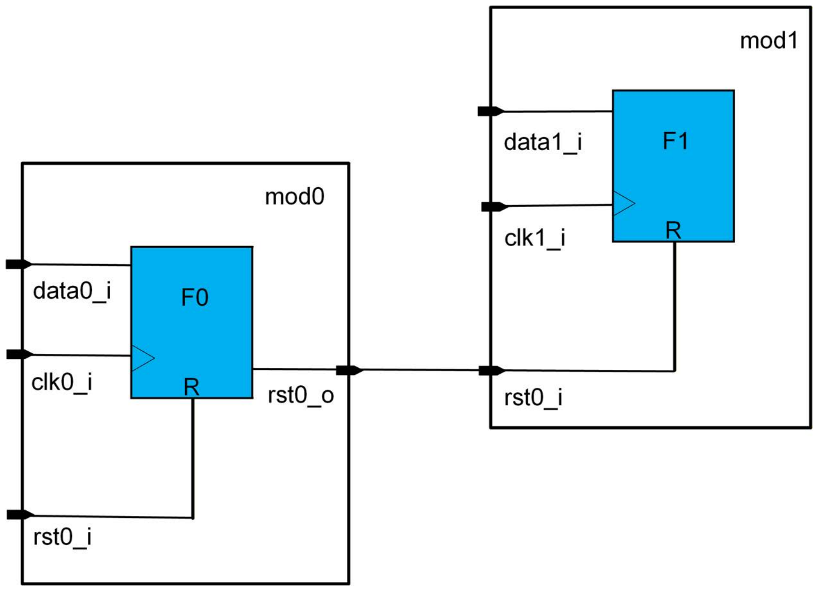

Figure 17—active_high reset on input with -associated_to_clocks

4.4 clock_group domain attributes

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 shall 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.

Example 23 shows clock relationships defined in one group:

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

Example 23—clock_group domain: defined in one group

Example 24 shows clock relationships defined in three groups:

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

accellera_cdc::set_clock_group \
    -name center_and_right     \      Name of clock group
    -clocks {clk2 clk3 clk4}          Names of clocks in the clock group

accellera_cdc::set_clock_group \
    -name balanced_branches    \      Name of clock group
    -clocks {clk1 clk4}               Names of clocks in the clock group
```

Example 24—clock_group domain: defined in three groups

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 21 for an example of a non-transitive use model.

The following examples show different cases of synchronicity. Solid arrows denote synchronous clock relationships. Dashed arrows denote asynchronous clock relationships.

Figure 18 depicts three clocks. All are pairwise synchronous to each other so that they build a common clock group (see Example 25).

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

Example 25—clock_group domain: three clocks building a common clock group

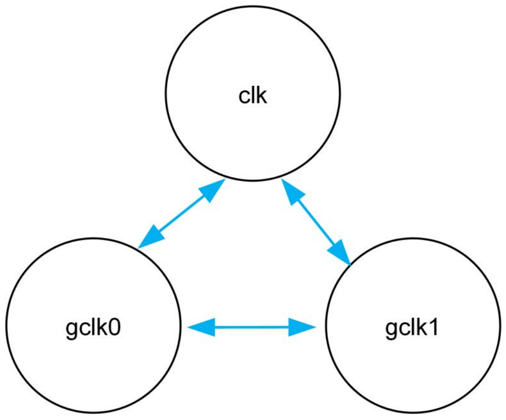

Figure 18—One clock group

Figure 19 depicts two clock groups. The clock `clk` builds a group on its own because it is asynchronous to the other two clocks (see Example 26).

```
accellera_cdc::set_clock_group \
    -name small_domain         \   Name of clock group
    -clocks {clk}                  Name of clock in the clock group

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

Example 26—clock_group domain: defining two clock groups

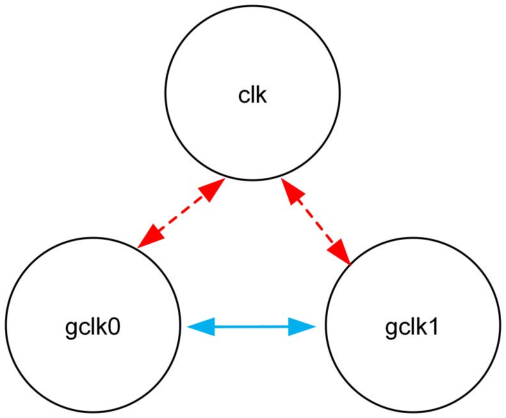

Figure 19—Two clock groups

Figure 20 depicts three clocks that are all pairwise asynchronous to each other so that every clock builds an individual clock group (see Example 27).

```
accellera_cdc::set_clock_group \
    -name domain_c             \   Name of clock group
    -clocks {clk}                  Name of clock in the clock group

accellera_cdc::set_clock_group \
    -name domain_0             \   Name of clock group
    -clocks {gclk0}                Name of clock in the clock group

accellera_cdc::set_clock_group \
    -name domain_1             \   Name of clock group
    -clocks {gclk1}                Name of clock in the clock group
```

Example 27—clock_group domain: defined in individual clock groups

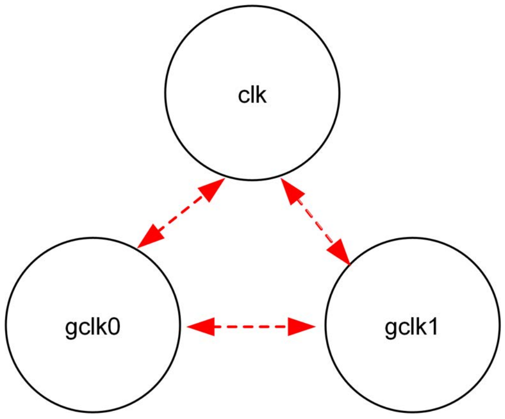

Figure 20—Three clock groups

Figure 21 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 (see Example 28).

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

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

Example 28—clock_group domain: defining non-transitive synchronicity between three clocks

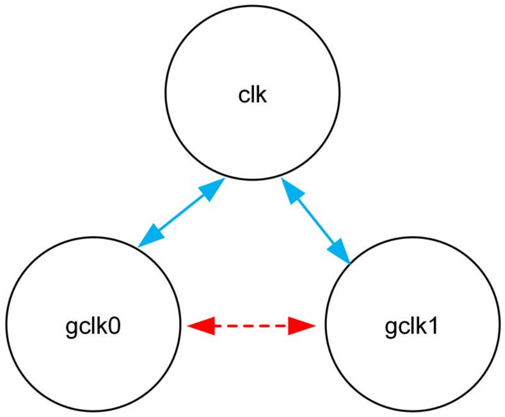

Figure 21—Two clock groups

4.5 port domain attribute: -type clock and clock_group domain attributes usage

This section shows three examples of clock definitions. Figure 22 depicts two asynchronous clocks `clk0_i` and `clk1_i` (see Example 29).

```
accellera_cdc::set_port -name clk0_i \        Name of port
    -direction input                 \        Direction of port
    -type clock                               Type of port

accellera_cdc::set_clock_group -name tx \     Name of clock group
    -clocks {clk0_i}                          Name of clock in the clock group

accellera_cdc::set_port -name clk1_i \        Name of port
    -direction input                 \        Direction of port
    -type clock                               Type of port

accellera_cdc::set_clock_group -name rx \     Name of clock group
    -clocks {clk1_i}                          Name of clock in the clock group
```

Example 29—Clock definition A: two asynchronous clocks

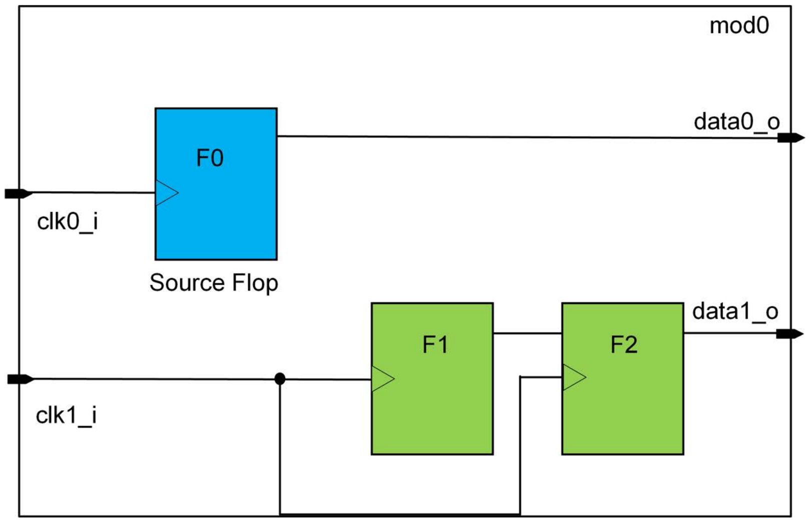

Figure 22—Clock definition A

Figure 23 depicts three asynchronous clocks `clk0_i`, `clk1_i`, and `clk2_i` (see Example 30).

```
accellera_cdc::set_port -name clk0_i \        Name of port
    -direction input                 \        Direction of port
    -type clock                               Type of port

accellera_cdc::set_clock_group -name tx \     Name of clock group
    -clocks {clk0_i}                          Name of clock in the clock group

accellera_cdc::set_port -name clk1_i \        Name of port
    -direction input                 \        Direction of port
    -type clock                               Type of port

accellera_cdc::set_clock_group -name rx \     Name of clock group
    -clocks {clk1_i}                          Name of clock in the clock group

accellera_cdc::set_port -name clk2_i \        Name of port
    -direction input                 \        Direction of port
    -type clock                               Type of port

accellera_cdc::set_clock_group -name new \    Name of clock group
    -clocks {clk2_i}                          Name of clock in the clock group
```

Example 30—Clock definition B: three asynchronous clocks

For `accellera_cdc::set_clock_group`, refer to 4.4.

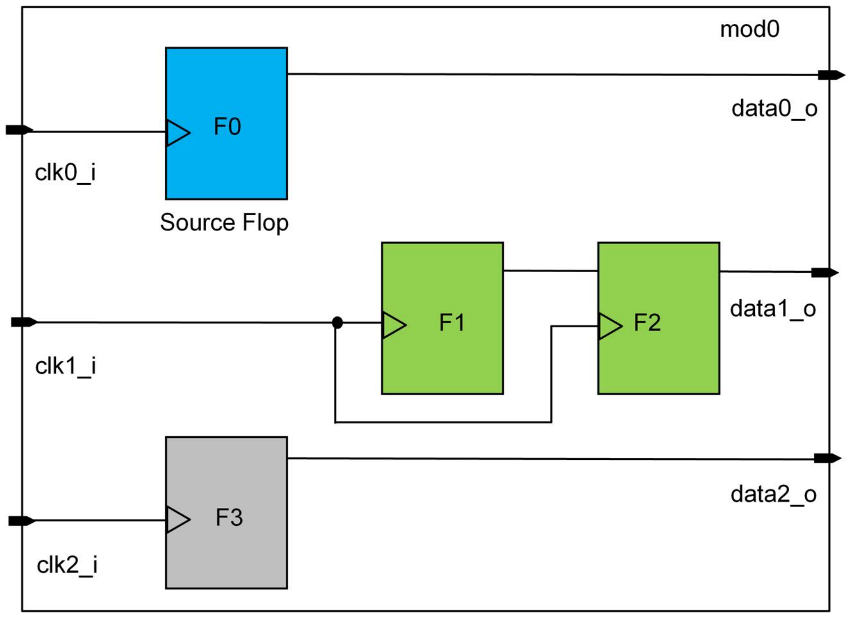

Figure 23—Clock definition B

Figure 24 depicts three clocks `clk0_i`, `clk1_i`, and `clk2_i`. In this example, `clk0_i` and `clk2_i` are asynchronous to `clk1_i`, but `clk0_i` and `clk2_i` are synchronous to each other (see Example 31).

```
accellera_cdc::set_port -name clk0_i \      Name of port
    -direction input                 \      Direction of port
    -type clock                             Type of port

accellera_cdc::set_port -name clk2_i \      Name of port
    -direction input                 \      Direction of port
    -type clock                             Type of port

accellera_cdc::set_clock_group -name tx \   Name of clock group
    -clocks {clk0_i clk2_i}                 Names of clocks in the clock group

accellera_cdc::set_port -name clk1_i \      Name of port
    -direction input                 \      Direction of port
    -type clock                             Type of port

accellera_cdc::set_clock_group -name rx \   Name of clock group
    -clocks {clk1_i}                        Name of clock in the clock group
```

Example 31—Clock definition C: mixture of synchronous and asynchronous clocks

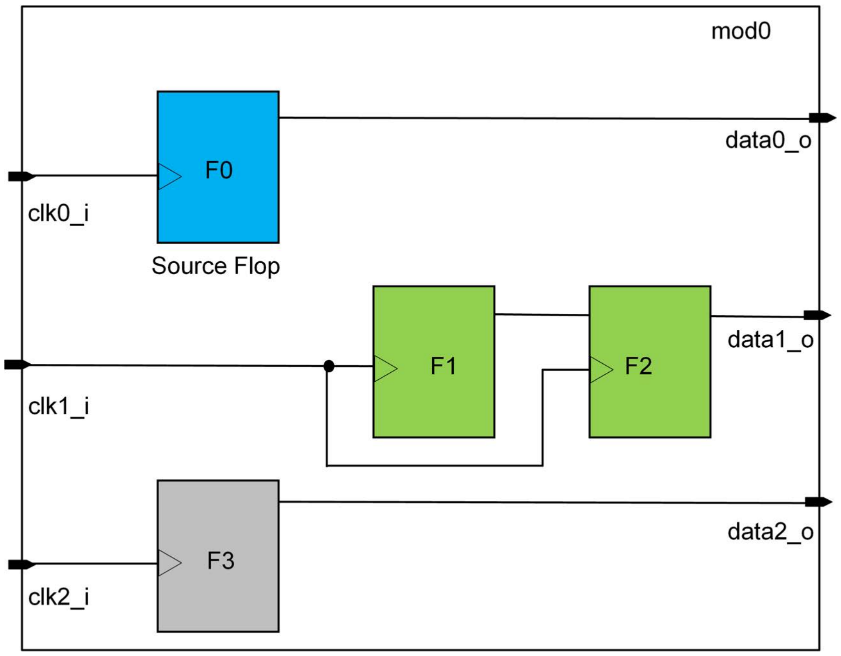

Figure 24—Clock definition C

5. Support for RDC Verification

This clause details output collateral requirements for IP with multiple resets from the top level and RDC control within and outside the IP. This section is applicable to EDA tools that support RDC verification. For EDA tools that do not require RDC verification, the requirements specified in this section are recommendations rather than mandatory requirements. For EDA tools that support the RDC verification features of this standard, the requirements are mandatory.

To support integration-level RDC verification, the CDC attribute tables (See Clause 4) include the following content.

a) New domains
   1) `reset_group`: Defines the reset relation. There are no RDC violations between resets that appear together in the same reset group.
   2) `reset_assertion_sequence`: Specifies the required order in which multiple reset signals shall be asserted
b) New attributes: port domain
   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.
   8) `-blocking_polarity`: Specifies the active value of the `rdc_control` signal that places the associated port into a blocking state for RDC handling.
      i) A value of 0 indicates that the port is in the blocking state when the `rdc_control` signal is logic 0.
      i) A value of 1 indicates that the port is in the blocking state when the `rdc_control` signal is logic 1.

      The `-blocking_polarity` attribute shall be specified whenever an `rdc_control` signal is associated with a port for RDC handling.
c) New attribute: reset_assertion_sequence domain
   `assert_sequence`: Specifies the required order in which multiple reset signals shall be asserted.
d) New values for type attributes
   1) `rdc_control` for the `-type` attribute: Defines the RDC control. The usage for this attribute is the same as `cdc_control`. The `rdc_control` for the `-type` attribute is used differently than `-rdc_control`, which is an attribute of the port domain.
   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 items above will be described with scenarios in the following sections.

5.1 Output Collateral Requirements for Basic RDC Scenarios

This clause specifies the requirements for supporting RDC verification at the integration level using output collateral generated by the IP. The approach does not require the RTL description of the IP to be read or processed by the verification tool.

At the time of generating the output collateral, the IP is not assumed to have knowledge of the number of resets that will drive its reset ports at the integration level.

The following subclauses describe scenarios corresponding to the supported RDC handling methods.

5.2 Scenario 1: Input interface drives internal synchronizer

The IP shown in Figure 32 shall capture the following information in its output collateral to enable integration-level RDC verification.

Example 32 is illustrative code provided to demonstrate the required modeling.

```
accellera_cdc::set_port           \
    -name data0_i                 \      Name of port
    -direction input              \      Direction of port
    -type data                    \      Type of port
    -associated_to_reset rst0_n_i \      The receiver reset of a port
    -associated_to_clocks clk0_i  \      Internal clock of the IP
    -logic internal sync                 Internal synchronizer
```

Example 32—Internal synchronization logic

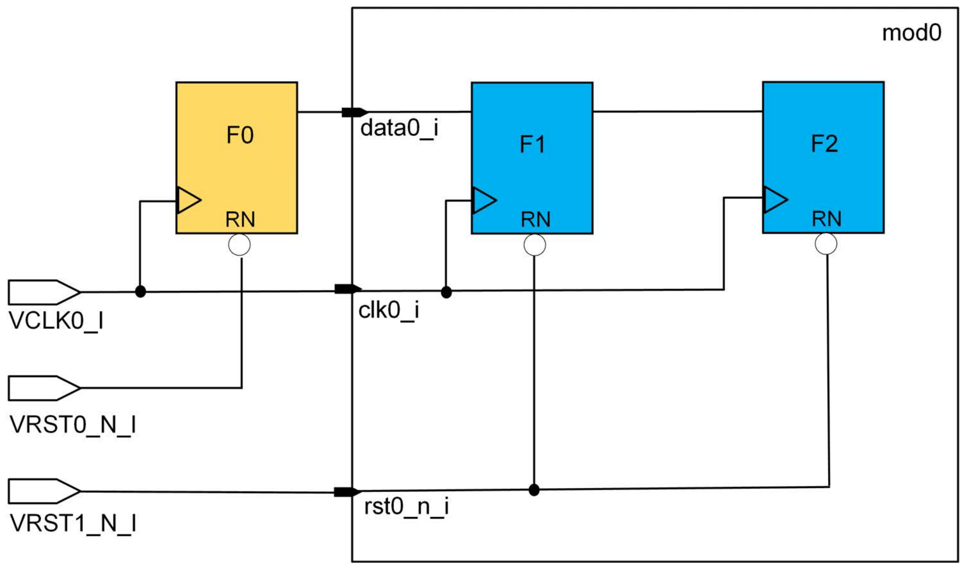

Figure 25—Internal synchronization logic

When a receiving port is declared with a synchronizer attribute, any RDC from that port shall be considered resolved, since the synchronizer is present within the IP.

In the example shown in Figure 25, port `data0_i` of the IP is a receiving port connected to the input of the synchronizer.

When the IP is integrated at the integration level shown in Figure 25, the connectivity, clock domain, and reset domain of the IP instance at the integration level shall be consistent with the IP's modeling assumptions. Accordingly, the following checks shall be performed:

Synchronizer handling requirement: The signal pulse driving a synchronizer input shall be wide enough to be reliably sampled by the destination clock.

Verification requirement: Verification of synchronizer handling requirement (minimum pulse width check) shall be performed by means of formal and/or dynamic verification.

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

5.3 Scenario 2: IP-level reset-sequence for safe internal RDC

The IP shown in Figure 26 shall capture the following information in its output collateral to enable integration-level RDC verification.

Example 33 is illustrative code provided to demonstrate the required modeling.

```
accellera_cdc::set_reset_assertion_sequence \
-name reset_seq_1                           \    Name of the reset sequence
-assert_sequence {rst1_n_i rst0_n_i}             The required order for the reset signals to be
                                                 asserted
```

Example 33—Reset assertion sequence

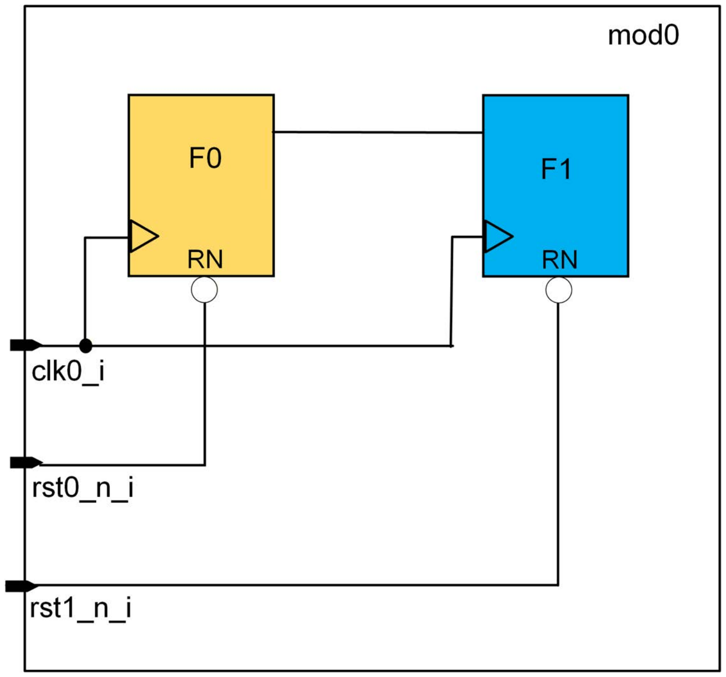

Figure 26—Reset assertion sequence

When the IP is integrated at the integration level shown in Figure 26, the connectivity, clock domain, and reset domain of the IP instance at the integration level shall be consistent with the IP's modeling assumptions. Accordingly, the following checks shall be performed:

Integration requirement: The reset sequence shall be specified in the IP collateral.

Verification requirement: Compliance with the integration requirement above (reset sequence specification) shall be verified by means of formal and/or dynamic verification to ensure correctness of the reset sequence.

For a description of the related test case, see Annex C.

5.4 Scenario 3: Interface RDC Control for safe internal RDC

The IP shown in Figure 27 shall capture the following information in its output collateral to enable integration-level RDC verification, assuming design-intent is that `rdcq0_i` goes low prior to the assertion of `rst0_n_i` and can be used to block the RDC path from `F0` to `F1`.

Example 34 is illustrative code provided to demonstrate the required modeling.

```
accellera_cdc::set_port               \
    -name rdcq0_i                     \   Name of port
    -direction input                  \   Direction of port
    -type rdc_control                 \   Type of port
    -blocking_polarity low            \   Active value of the rdc_control signal that places the
                                      \   associated port into a blocking state for RDC handling
    -associated_to_reset rst1_n_i \   The receiver reset of a port
    -associated_to_clocks clk0_i  \   Internal receiver clock
    -rdc_data_from_reset rst0_n_i     The source reset being blocked by -rdc_control
```

Example 34— Input port modeling: Internal data/clock gating with internal data source

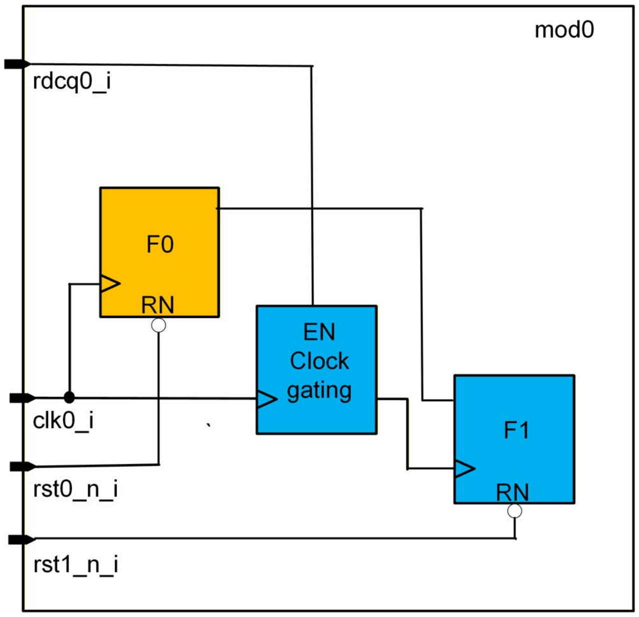

Figure 27—Input port modeling: Internal data/clock gating with internal data source

When the IP is integrated at the integration level shown in Figure 27, the connectivity, clock domain, and reset domain of the IP instance at the integration level shall be consistent with the IP's modeling assumptions. Accordingly, the following checks shall be performed:

Integration requirements:

a) A signal used as an `rdc_control` shall satisfy all of the following conditions:
   1) The `rdc_control` signal shall be synchronous to the receiving clock domain.
   2) The `rdc_control` signal shall be generated within, and remain in, the same reset domain as the receiving logic.
   3) The `rdc_control` signal shall not be derived from, or directly controlled by, an asynchronous reset source unless it is guaranteed to be free of hazards and glitches when sampled in the receiving domain.

b) The `rdc_control` signal shall be in the blocking state (`-blocking_polarity`) upon assertion of the source asynchronous reset (`-rdc_data_from_reset`).

Verification requirements:

a) Compliance with `rdc_control` conditions (integration requirement a) shall be verified by static structural analysis.

b) Compliance with the `rdc_control` blocking behavior (integration requirement b) shall be verified by means of formal and/or dynamic verification.

For a description of the related test case, see Annex C.

5.5 Scenario 4: Interface RDC control for safe interface RDC crossing

The IP shown in Figure 28 shall capture the following information in its output collateral to enable integration-level RDC verification, assuming design-intent is that `rdcq0_i` goes low prior to the assertion of the reset that drives input `data0_i`, and can be used to block the RDC crossing on `data0_i`.

Example 35 is illustrative code provided to demonstrate the required modeling.

```
accellera_cdc::set_port             \
    -name rdcq0_i                   \    Name of port
    -direction input                \    Direction of port
    -type rdc_control               \    Type of port
    -blocking_polarity low          \    Active value of the rdc_control signal that places the asso-
                                    \    ciated port into a blocking state for RDC handling
    -associated_to_reset rst1_n_i \    The receiver reset of a port
    -associated_to_clocks clk0_i    \    Internal receiver clock

accellera_cdc::set_port             \
    -name data0_i                   \    Name of port
    -direction input                \    Direction of port
    -type data                      \    Type of port
    -rdc_control rdcq0_i            \    The RDC control associated with the data port to avoid a
                                    \    timing violation
    -associated_to_reset rst1_n_i \    The receiver reset of a port
    -associated_to_clocks clk0_i         Internal receiver clock
```

Example 35— Input port modeling: Internal data/clock gating with external data source

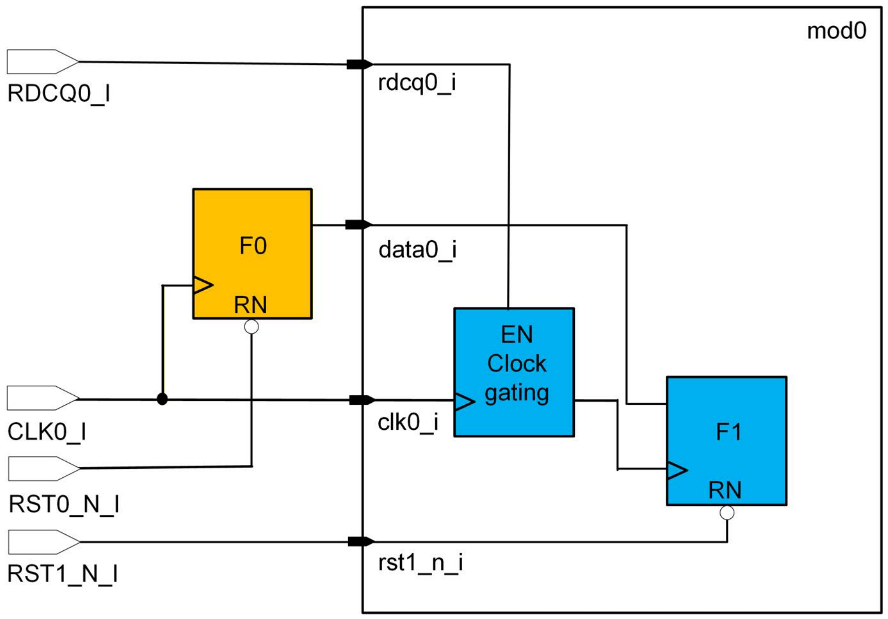

Figure 28— Input port modeling: Internal data/clock gating with external data source

When the IP is integrated at the integration level shown in Figure 28, the connectivity, clock domain, and reset domain of the IP at the integration level shall be consistent with the IP's modeling assumptions. Accordingly, the following checks shall be performed:

Integration requirements:

a) A signal used as an `rdc_control` shall satisfy all of the following conditions:
   1) The `rdc_control` signal shall be synchronous to the receiving clock domain.
   2) The `rdc_control` signal shall be generated within, and remain in, the same reset domain as the receiving logic.
   3) The `rdc_control` signal shall not be derived from, or directly controlled by, an asynchronous reset source unless it is guaranteed to be free of hazards and glitches when sampled in the receiving domain.

b) The `rdc_control` signal shall be in the blocking state (`-blocking_polarity`) upon assertion of the source asynchronous reset at the integration level (`RST0_N_I`).

Verification requirements:

a) Compliance with `rdc_control` conditions (integration requirement a) shall be verified by static structural analysis.

b) Compliance with the `rdc_control` blocking behavior (integration requirement b) shall be verified by means of formal and/or dynamic verification.

For a description of the related test case, see Annex C.

5.6 Scenario 5: Output interface RDC control for safe split RDC

The IP shown in Figure 29 shall capture the following information in its output collateral to enable integration-level RDC verification, assuming the design-intent is for `rdcq0_o` to assert low prior to the asynchronous reset on output `data0_o` occurs, thus can be used to block the RDC path on output `data0_o`.

Example 36 is illustrative code provided to demonstrate the required modeling.

```
accellera_cdc::set_port             \
    -name rdcq0_o                   \     Name of port
    -direction output               \     Direction of port
    -type rdc_control               \     Type of port
    -blocking_polarity low          \     Active value of the rdc_control signal that places the
                                    \     associated port into a blocking state for RDC handling
    -associated_to_reset rst0_n_i   \     The receiver reset of a port
    -associated_to_clocks clk0_i    \     Internal receiver clock
    -rdc_data_from_reset rst0_n_i         The source reset being blocked by -rdc_control

accellera_cdc::set_port               \
    -name data0_o                     \   Name of port
    -direction output                 \   Direction of port
    -type data                        \   Type of port
    -rdc_control rdcq0_o              \   The RDC control associated with the data port to avoid a
                                      \   timing violation
    -associated_from_reset rst0_n_i   \   The source reset being blocked by -rdc_control
    -associated_from_clock clk0_i         External driver clock
```

Example 36—Output port modeling: External data/clock gating for output port

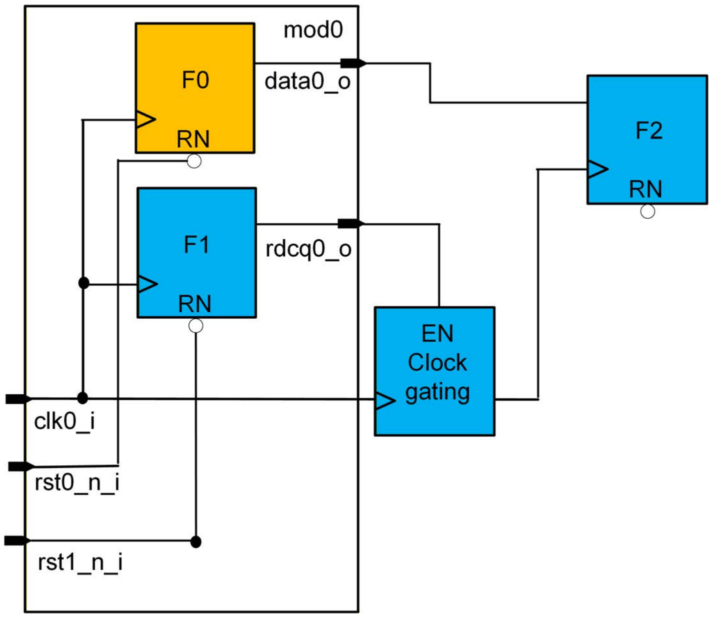

Figure 29—Output port modeling: External data/clock gating for output port

When the IP is integrated at the integration level shown in Figure 29, the connectivity, clock domain, and reset domain of the IP instance at the integration level shall be consistent with the IP's modeling assumptions. Accordingly, the following checks shall be performed:

Integration requirements:

a) A signal used as an `rdc_control` shall satisfy all of the following conditions:

   1) The `rdc_control` signal shall be synchronous to the receiving clock domain

   2) The `rdc_control` signal shall be generated within, and remain in, the same reset domain as the receiving logic.

b) The `rdc_control` signal shall be in the blocking state (`-blocking_polarity`) upon assertion of the source asynchronous reset (`-rdc_data_from_reset`).

Data/clock gating requirements: The data-gating structure shall be implemented in accordance with the blocking polarity of the `rdc_control` signal.

a) If the blocking polarity is 0, the gating structure shall block data when `rdc_control = 0`.

b) If the blocking polarity is 1, the gating structure shall block data when `rdc_control = 1`.

Verification requirements:

a) Compliance with `rdc_control` conditions (integration requirement a) shall be verified by static structural analysis.

b) Compliance with the `rdc_control` blocking behavior (integration requirement b) shall be verified by means of formal and/or dynamic verification.

c) Verification of the data gating requirement (implementation of gating structure based on blocking polarity) shall be performed by static structural analysis.

For a description of the related test case, see Annex C.

5.7 Scenario 6: Output interface prequalified for safe RDC crossing on output

The IP shown in Figure 30 shall capture the following information in its output collateral to enable integration-level RDC verification. The design-intent is that prior to `rst0_n_i` asserting (potentially creating an RDC path from `F0` to `F2` through output port `data0_o`), `F2` driving `rdcq0` is set to a blocking value (in this case "0" blocking at the AND gate), guaranteeing that any RDC path from `rst0_n_i` is blocked, and the output `data0_o` functionally looks like it is only influenced by `rst1_n_i`.

Example 37 is illustrative code provided to demonstrate the required modeling.

```
accellera_cdc::set_port                \
    -name data0_o                      \   Name of port
    -direction output                  \   Direction of port
    -type data                         \   Type of port
    -associated_from_reset rst1_n_i \      Internal reset of a port
    -associated_from_clock clk0_i          Internal driver clock
```

Example 37—Output port modeling: Internal data gating

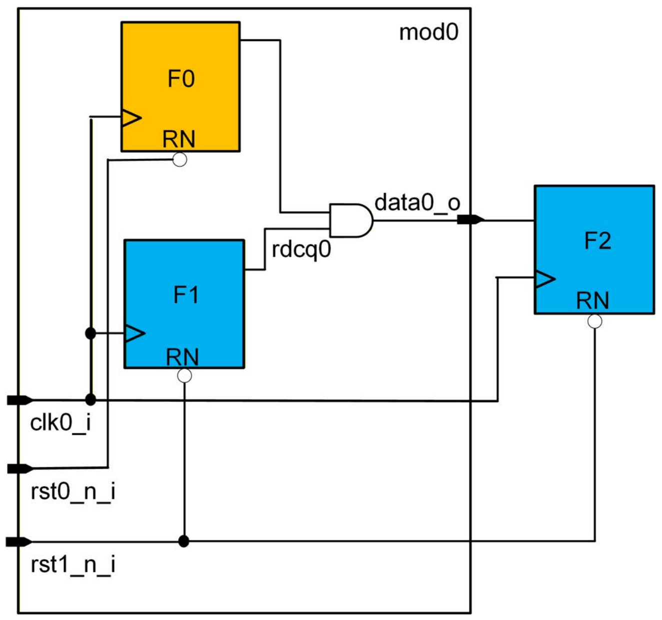

Figure 30—Output port modeling: Internal data gating

When the IP is integrated at the integration level shown in Figure 30, the connectivity, clock domain, and reset domain of the IP instance at the integration level shall be consistent with the IP's modeling assumptions. Accordingly, the following checks shall be performed:

Integration Requirement: The data signal shall be in the same reset domain (`-associated_from_reset`) as the receiving logic.

Verification Requirement: Compliance with the integration requirement (no RDC; i.e., safe RDC) shall be verified by means of static structural analysis.

For a description of the related test case, see Annex C.

6. CDC Tcl format

This clause defines the output specification using Tcl, focusing on output collateral. Typically, each block or module has a separate Tcl file where users can define CDC and RDC attributes for ports and specify clock groups. While each Tcl file usually corresponds to an RTL module, users may choose to handle these files independent of RTL. The Tcl API commands are designed according to current requirements for output collateral, with the possibility of adding more commands in the future to support input collateral. This clause explains the syntax and semantics for each Tcl command.

NOTE—The Tcl language used here is case sensitive.

6.1 accellera_cdc Namespace

EDA tools shall support the use of an `accellera_cdc namespace`, so these commands can be used with the namespace as shown in Example 38.

NOTE: The `accellera_cdc  namespace` cannot include non-standard extensions. If custom extensions are needed, a different namespace name should be used.

```
# Defines a namespace (within the EDA tool - transparent to the user)
namespace eval accellera_cdc {
    proc set_module {} {
        ...
    }

    proc set_port {} {
        ...
    }

    proc set_clock_group {} {
        ...
    }

    proc set_param {} {
        ...
    }
}
# Users Tcl for CDC
accellera_cdc::set_module -name ALU
accellera_cdc::set_port CLK -type
clock ...
...
```

Example 38—Defining an accellera_cdc namespace

6.2 accellera_cdc::set_module

There can be an RTL file defining the specified module, but it is not mandatory. Refer to Table 1 for details of the module domain attribute and also see Example 39.

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 `accellera_cdc::set_module` command.

There shall be an error if *module-name* is not specified.

```
# Set a module for CDC specification
accellera_cdc::set_module -name ALU

# Set attributes of a port
accellera_cdc::set_port -name CLK -type clock -virtual_port p2
    -frequency value ...

accellera_cdc::set_port -name {RegA[7:0]} -type data ...

accellera_cdc::set_port -name {RegA[WIDTH-1:0]} -type data ...

accellera_cdc::set_port -name {[4:0]RegA[7:0]} -type data ...

accellera_cdc::set_port -name RESET -type reset -polarity high ...
...
```

Example 39—accellera_cdc::set_module

6.3 accellera_cdc::set_param

The `accellera_cdc::set_param` command allows users to define parameters within the scope of a module. Refer to Table 2 for details of the parameter domain attribute and also see Example 40.

There shall be an error if:

a) *parameter-name* is not specified.

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

```
# Set a module
accellera_cdc::set_module -name ALU
# Set a parameter for the module
accellera_cdc::set_param -name MSB -type int -value 15
accellera_cdc::set_param -name LSB -type int -value 0

accellera_cdc::set_port -name {RegA[MSB:LSB]} -type data ...
# EDA tool shall resolve these parameters
```

Example 40—accellera_cdc::set_module

6.4 accellera_cdc::set_port

The `accellera_cdc::set_port` command allows users to set attributes of a port on a single line. The CDC attributes for a port are listed in the Table 3 and Example 41. A single `accellera_cdc::set_port` command defines all CDC and RDC attributes for a port. Multiple `accellera_cdc::set_port` commands with the same port name within a single module override the previous values of attributes.

There shall be an error if:

a) a module has not been set using the `accellera_cdc::set_module` command.
b) an attribute is specified for a particular type of port that is not applicable to that type of port.
c) range is not specified correctly for a vector or multidimensional port.

```
# Set a module
accellera_cdc::set_module -name ALU
# Set attributes of a port
accellera_cdc::set_port CLK -type clock -virtual_port p2 -frequency value ...

accellera_cdc::set_port {RegA[7:01]} -type data ...; #Vector

accellera_cdc::set_port {RegA[WIDTH-1:0]} -type data; #Parameterized Vector

accellera_cdc::set_port -name {RegA[7:0][4:0]} -type data; # 2D

accellera_cdc::set_port -name {RegA[WIDTH-1:0][DEPTH-1:0]}; #Parameterized 2D

accellera_cdc::set_port {{RegA[7:5]} {RegA[2]} {RegA[0]}} -type data;
   #Concatenation
accellera_cdc::set_port -name {RegA[MSB:-LSB]} -type data; #Parameterized
   vector with negative index

accellera_cdc::set_port RESET -type reset -polarity high ...
...
```

Example 41—accellera_cdc::set_port

6.5 accellera_cdc::set_clock_group

The `accellera_cdc::set_clock_group` command allows users to set clock groups of synchronous clocks. A clock group can be specified as a Tcl list having one or more clock names that are synchronous. Refer to Table 6 for details of the `clock_group` domain attribute and Example 42.

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

```
# Set a module
accellera_cdc::set_module ALU
# Set attributes of a port
accellera_cdc::set_port CLK -type clock -virtual_port p2 -frequency value ...

accellera_cdc::set_port RegA -type data ...

accellera_cdc::set_port RESET -type reset -polarity high ...
...
# set cdc clock group
accellera_cdc::set_clock_group -clocks {clk1 clk2}
```

Example 42—accellera_cdc::set_clock_group

7. CDC IP-XACT format

This section describes the IP-XACT format for CDC specification based on IEEE Std 1685-2022. It is a vendor 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 at different elements in the design hierarchy. They are described in detail in the following sections.

7.1 Top-level elements

The CDC vendor extensions allow specifying the following sets of information:

- `accellera-cdc:component`
- `accellera-cdc:wire`

This is demonstrated in the schema diagram in Figure 31.

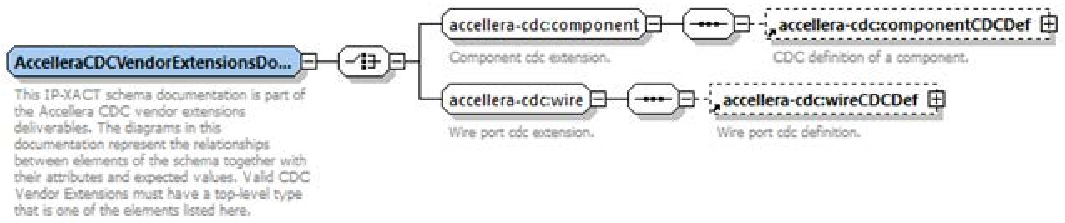
AcceleraCDCVendorExtensionsDo...
This IP-XACT schema documentation is part of the Accellera CDC vendor extensions deliverables. The diagrams in this documentation represent the relationships between elements of the schema together with their attributes and expected values. Valid CDC Vendor Extensions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

Figure 31—Schema for top-level elements

7.2 Top element—accellera-cdc:wireCDCDef

The top-level element `accellera-cdc:wireCDCDef` is used to define a CDC-specific wire port extension. 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 32.

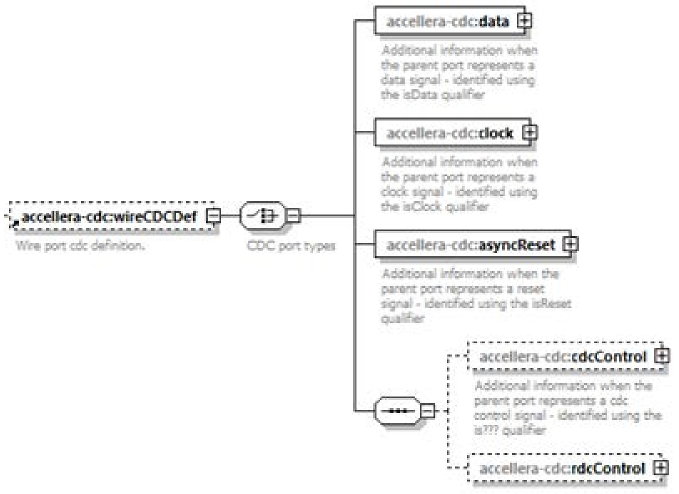
accellera-cdc:wireCDCDef
Wire port cdc definition.
CDC port types
accellera-cdc:data
Additional information when the parent port represents a data signal - identified using the isData qualifier
accellera-cdc:clock
Additional information when the parent port represents a clock signal - identified using 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 32—Schema for accellera-cdc:wireCDCDef

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 33. Each element of the `accellera-cdc:data` extension map to an attribute for the data port as defined in Clause 4, Table 1.

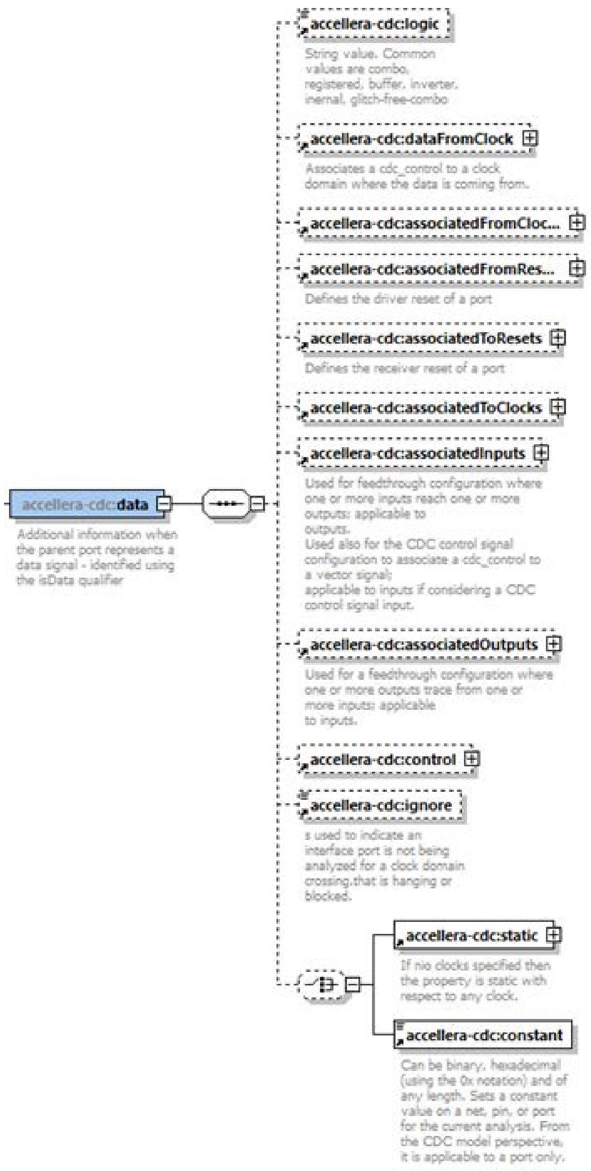
accellera-cdc:data
Additional information when the parent port represents a data signal - identified using the isData qualifier
accellera-cdc:logic
String value. Common values are combo, registered, buffer, inverter, inernal, glitch-free-combo
accellera-cdc:dataFromClock
Associates a cdc_control to a clock domain where the data is coming from.
accellera-cdc:associatedFromCloc...
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: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 33—Schema 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:wireCDCDef>
         <accellera-cdc:data>
            <accellera-cdc:associatedFromClocks>
               <accellera-cdc:clockPortReference>i_clk</accellera-
   cdc:clockPortReference>
            </accellera-cdc:associatedFromClocks>
         </accellera-cdc:data>
          </accellera-cdc:wireCDCDef>
   </ipxact:vendorExtensions>
</ipxact:port>
```

Example 43—IP-XACT code for accellera-cdc:data

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 34. It allows specifying the `accellera-cdc:logic` attribute in the `accellera-cdc:clock` extension.

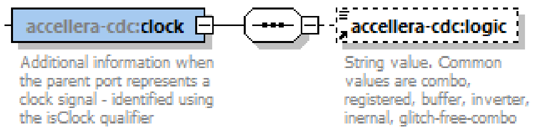
accellera-cdc:clock
Additional information when the parent port represents a clock signal - identified using the isClock qualifier
accellera-cdc:logic
String value. Common values are combo, registered, buffer, inverter, inernal, glitch-free-combo

Figure 34—Schema 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:wireCDCDef>
                    <accellera-cdc:clock>
                        <accellera-cdc:logic>combo</accellera-cdc:logic>
                            <accellera-cdc:clock>
                </accellera-cdc:wireCDCDef>
    </ipxact:vendorExtensions>
    </ipxact:port>
```

Example 44—IP-XACT code for 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:vendorExtensions>
            <accellera-cdc:wire>
                <accellera-cdc:wireCDCDef>
                    <accellera-cdc:clock>
                        <accellera-cdc:logic>internal</accellera-cdc:logic>
                    <accellera-cdc:clock>
                </accellera-cdc:wireCDCDef>
    </ipxact:vendorExtensions>
    </ipxact:port>
```

Example 45—IP-XACT code for virtual accellera-cdc:clock

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 35. 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.

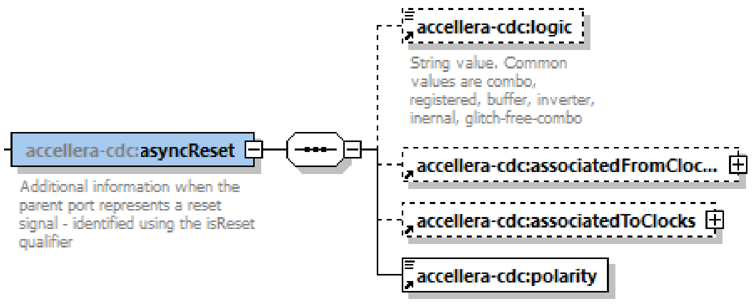
accellera-cdc:asyncReset
Additional information when the parent port represents a reset signal - identified using the isReset qualifier
accellera-cdc:logic
String value. Common values are combo, registered, buffer, inverter, inernal, glitch-free-combo
accellera-cdc:associatedFromCloc...
accellera-cdc:associatedToClocks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

Figure 35—Schema 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:wireCDCDef>
                <accellera-cdc:asyncReset>
                   <accellera-cdc:associatedFromClocks>
                      <accellera-cdc:clockPortReference>i_clk
                      </accellera-cdc:clockPortReference>
                   </accellera-cdc:associatedFromClocks>
                 <accellera-cdc:asyncReset>
    </accellera-cdc:wire>
       </ipxact:vendorExtensions>
    </ipxact:port>
```

Example 46—IP-XACT code for accellera-cdc:asyncReset

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 36.

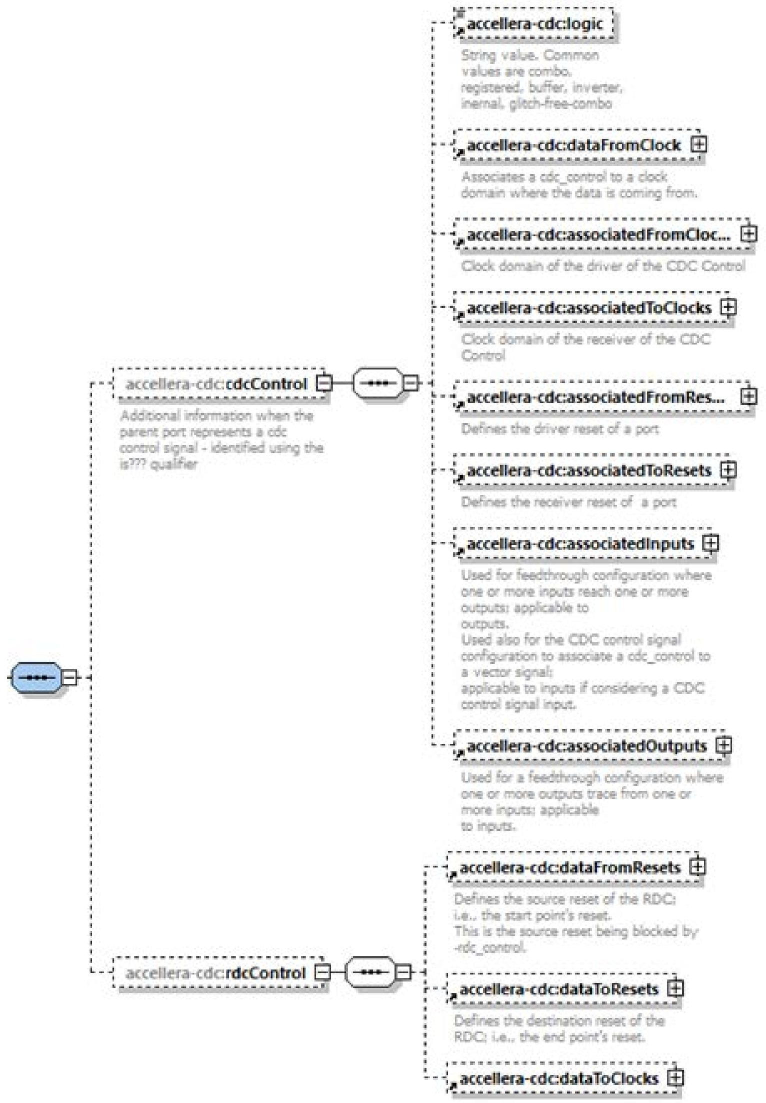
accellera-cdc:cdcControl
Additional information when the parent port represents a cdc control signal - identified using the is??? qualifier
accellera-cdc:logic
String value. Common values are combo, registered, buffer, inverter, inernal, glitch-free-combo
accellera-cdc:dataFromClock
Associates a cdc_control 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 36—Schema for accellera-cdc:cdcControl and accellera-cdc:rdcControl

```
    <ipxact:port>
       <ipxact:name>r_valid</ipxact:name>
       <ipxact:wire>
          <ipxact:direction>in</ipxact:direction>
       </ipxact:wire>
       <ipxact:vendorExtensions>
             <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:wire>
       </ipxact:vendorExtensions>
    </ipxact:port>
```

Example 47—IP-XACT code for accellera-cdc:cdcControl

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 37. It contains references to clock ports or phantom ports (virtual clock) in addition to `name`, `displayName`, and `description`.

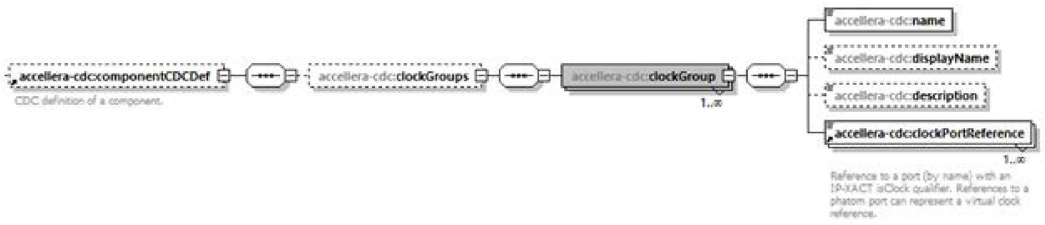
accellera-cdc:componentCDCDef
CDC definition of a component.
accellera-cdc:clockGroups
accellera-cdc:clockGroup
1..∞
accellera-cdc:name
accellera-cdc:displayName
accellera-cdc:description
accellera-cdc:clockPortReference
1..∞
Reference to a port (by name) with an IP-XACT isClock qualifier. References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

Figure 37—Schema for accellera-cdc:componentCDCDef

```
<ipxact:component>
   <!-- Specify VLNV -->
   ...
   ...
   <ipxact:vendorExtensions>
<accellera-cdc:componentCDCDef>
      <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>
</accellera-cdc:componentCDCDef>
   </ipxact:vendorExtensions>
</ipxact:component>
```

Example 48—IP-XACT code for accellera-cdc:componentCDCDef

7.8 Accellera CDC Vendor Extensions SCRs

Accellera IP-XACT CDC vendor extensions shall obey the semantic consistency rules listed in Table 11 to be valid.

Table 11—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	

Table 11—CDC semantic consistency rules *(continued)*

Name	Rule	Single doc check	Notes
`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 closed box CDC integrity verification

8.1 Overview

Closed 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 shall be performed using glass 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 closed box CDC integrity verification:

- The leaf-level IP is a closed 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 shall 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 closed box-to-closed box single-bit crossings with external synchronization between the output port of the source IP and the input port of the destination IP.

The closed box IP can be categorized as:

- Closed box with external synchronization.
- Closed box with internal synchronization.

The crossing type can be categorized as:

- SOC glue logic to closed box, single-bit crossing.
- SOC glue logic to closed box, multi-bit crossing.
- Closed box to closed box, single-bit crossing.
- Closed box to closed 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 closed box crossings can also be used to signify SOC with only one IP. Closed 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 closed box. It can be grouped under a closed 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 be 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 closed 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 38 shows an internal synchronizer for an internal clock `dest_clk`, with an active positive edge sampling.

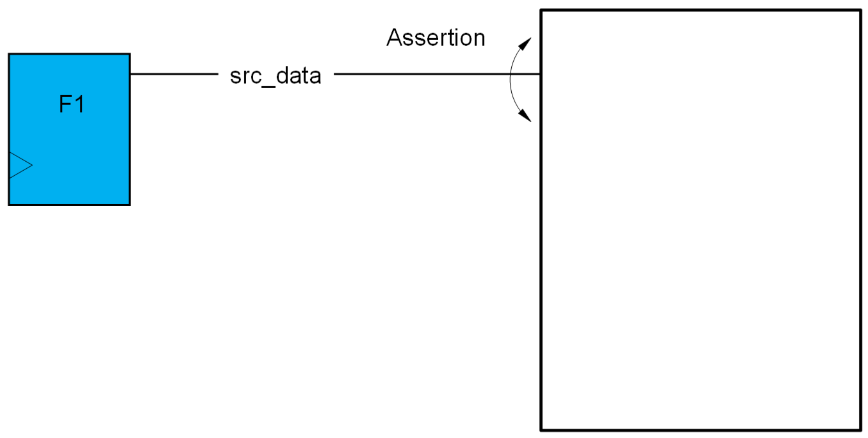

Figure 38—Sampling edge requirement example

```
accellera_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.

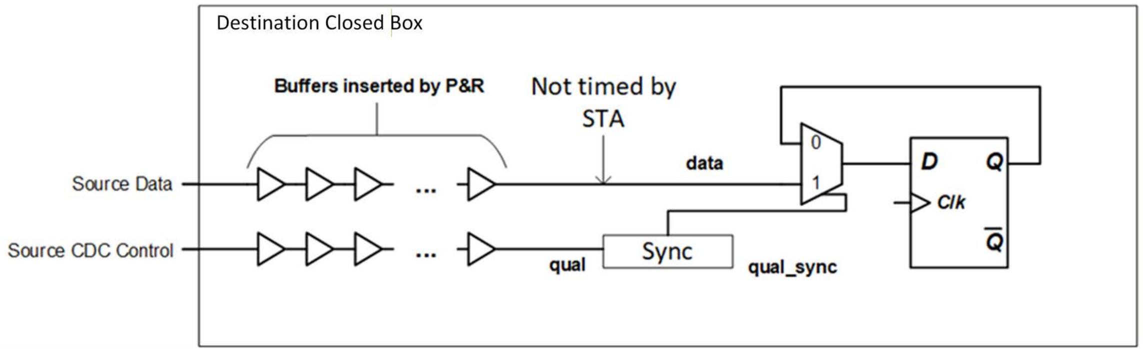

Figure 39—Example of an untimed path at the clock domain boundary

The path delay between the interface port and the sink within the destination closed 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`.

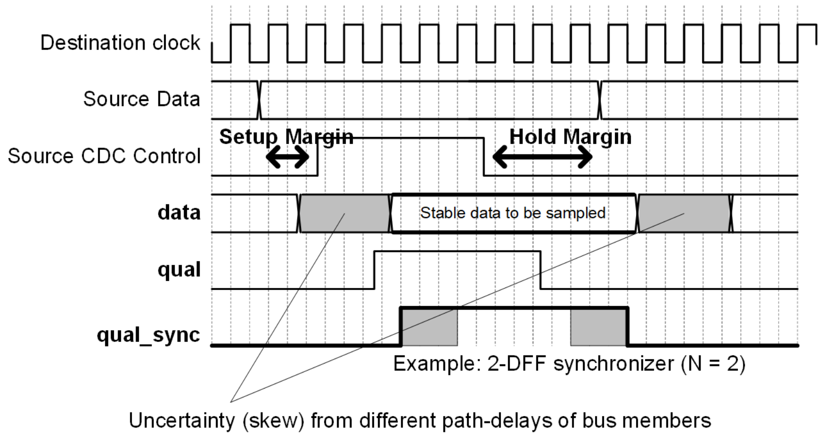
Destination clock
Source Data
Source CDC Control
Setup Margin
Hold Margin
data
Stable data to be sampled
qual
qual_sync
Example: 2-DFF synchronizer (N = 2)
Uncertainty (skew) from different path-delays of bus members

Figure 40—Implementation headroom

The setup margin of implementation headroom is defined as the amount of duration a crossing signal shall 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 shall 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 Annex A.

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 as shown in Figure 41. 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.

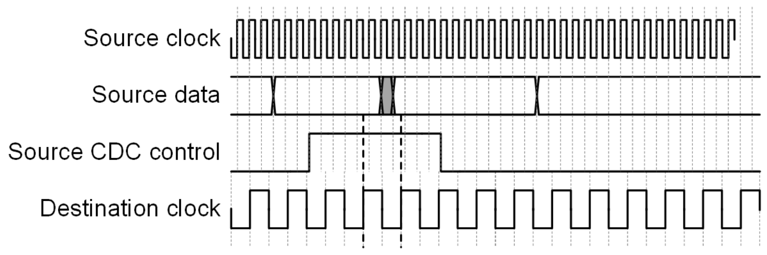

Figure 41—Necessity of a faster clock

Annex A

(informative)

CDC and RDC use case examples

This LRM outlines the primary use cases for the CDC/RDC interface standard such as, IP packaging and IP integration.

A.1 IP packaging

The IP packaging use case enables comprehensive capture of the capabilities and constraints of an IP block. This includes defining clock signals and determining whether they adhere to appropriate synchronization schemes (synchronous/asynchronous), as well as associating interface signals with specific clock and reset signals.

A.2 IP Integration

The standard facilitates the integration of CDC/RDC-aware IP blocks into system-level designs. IP-level CDC/RDC verification would not be necessary with the use of the CDC/RDC interface standard during integration at the system-level. The extent of automation and validation depends on the level of support provided by the design tool environment.

A.3 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 (see Example 49, Example 50, and Example 51, Example 52, Example 53, and Example 54). The following examples describe the circuit in Figure 42.

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 shall 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 shall be stable after the CDC control port disables the data crossing.

```
accellera_cdc::set_port -name data0_i \   Name of port
    -direction input                  \   Direction of port
    -type data                        \   Type of port
    -associated_from_clocks clk1_i    \   External driver clock
    -cdc_control cdcq0_i              \   Name of control port
    -associated_to_clocks clk0_i      \   Internal receiver's clock
    -logic combo                          Port has combo logic on path to reg
```

Example 49—Modeling abstracted blocks, port data0_i

```
accellera_cdc::set_port -name cdcq0_i \   Name of port
    -direction input                  \   Direction of port
    -type cdc_control                 \   Control port used to qualify cdcq0_i
    -cdc_data_from_clock clk1_i       \   External clock driving this port
    -associated_from_clocks clk0_i    \   External driver clock
    -associated_inputs data0_i        \   Data ports controlled by this port
    -associated_to_clocks clk0_i      \   Internal receiver's clock
    -logic combo                      \   Port has combo logic on path to register
    -cdc_control_setup 2              \   data0_i shall be stable for 2 clocks before cdcq0_i
    -cdc_control_hold 1                   data0_i shall be stable for 1 clock after cdcq0_i
```

Example 50—Modeling abstracted blocks, port cdcq0_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.

```
accellera_cdc::set_port -name data1_i \   Name of port
    -direction input                  \   Direction of port
    -type data                        \   Type of port
    -associated_from_clocks clk1_i    \   External driver clock
    -associated_to_clocks {           \   Internal receiver's clocks for multiple fanouts of port
                                      \   data1_i:
      clk1_i                          \   The first fanout is to a sequential running on clk1_i.
      {clk0_i internal_sync}          \   The second fanout is to a synchronizer running on
                                      \   clk0_i.
      clk0_i                          \   The third fanout is a sequential running on clk0_i.
      }
```

Example 51—Modeling abstracted blocks, port data1_i

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

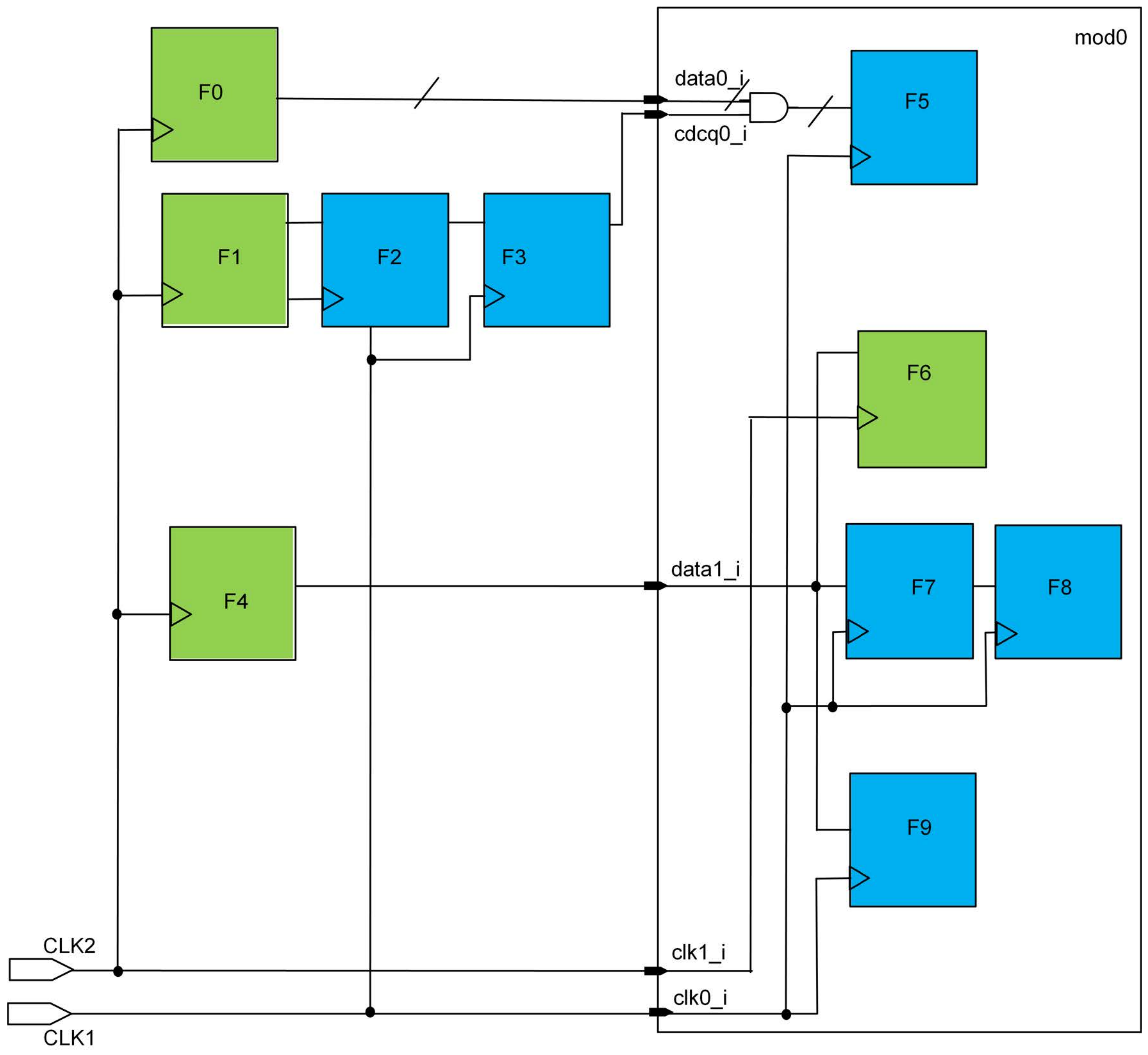

Figure 42—Module mod0 with abstracted input ports

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 43.

```
accellera_cdc::set_port -name data1_o \      Name of port
    -direction output                 \      Direction of port
    -type data                        \      Type of port
    -associated_from_clocks clk1_i    \      External driver clock
    -cdc_control cdcq0_o                     Output port controlled by data1_o
```

Example 52—Abstracted output port data1_o

```
accellera_cdc::set_port -name cdcq0_o \      Name of port
    -direction output                 \      Direction of port
    -type cdc_control                 \      Type of port
    -cdc_data_from_clock clk1_i       \      Internal clock driving this port
    -associated_from_clocks clk0_i    \      Internal driver clock
    -associated_outputs data1_o              Output port controlled by cdcq0_o
```

Example 53—Abstracted output port cdcq0_o

`-associated_outputs data1_o` is redundant with the information on port `data1_o` above, but this information can be added optionally:

```
accellera_cdc::set_port -name data1_o \      Name of port
    -direction output                 \      Direction of port
    -type data                        \      Type of port
    -associated_from_clocks clk1_i           External driver clock
```

Example 54—Abstracted output port data1_o

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

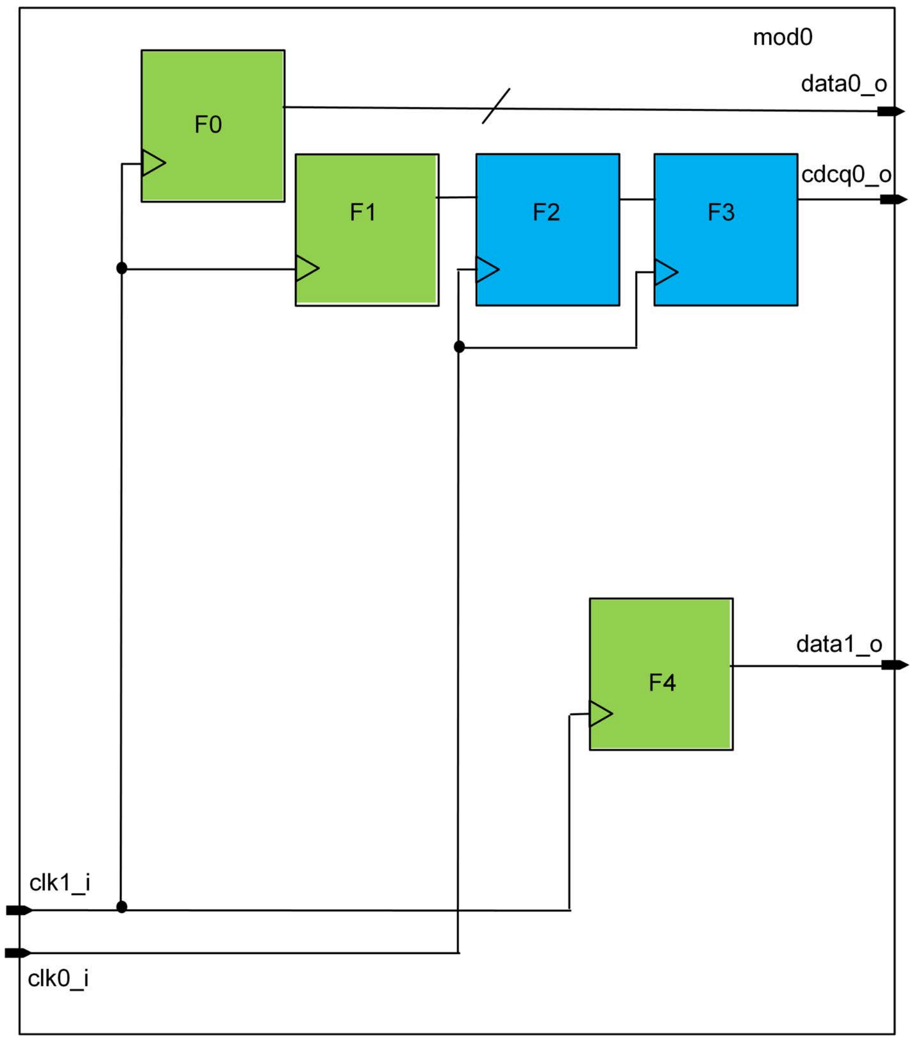

Figure 43—Module mod0 with abstracted output ports

A.4 Failure modes

This section includes failure mode cases with sample commands and supporting diagrams.

A.4.1 Reset driven by the wrong clock domain

Example 55 has a reset driven by the wrong clock domain. This example has a clock mismatch, assuming `clk1` and `clk2` are asynchronous. Example 55 provides sample commands, and Figure 44 provides a diagram for this case. With this example, 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 C.

```
accellera_cdc::set_module -name mod0            Name of module

accellera_cdc::set_port -name rst1_n_o \        Name of port
    -direction output                  \        Direction of port
    -type async_reset                  \        Type of port
    -polarity low                      \        Active low asynchronous reset
    -associated_from_clocks clk1                External driver clock

accellera_cdc::set_module -name mod1   \        Name of module

accellera_cdc_set_port -name rst1_n_i \         Name of port
    -type async_reset                 \         Type of port
    -polarity low                     \         Active low asynchronous reset
    -associated_to_clocks clk1_i                Internal receiver clock
```

Example 55—Unsynchronized reset crossing clock domains

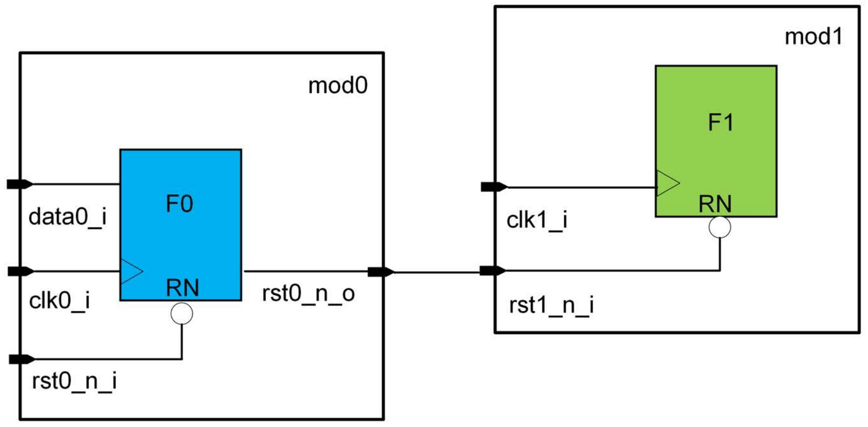

Figure 44—Unsynchronized reset crossing clock domains

A.4.2 Reset driven by correct clock domain but incorrect reset polarity

The example in this section demonstrates simple incorrect polarity. Example 56 provides sample commands, and Figure 45 provides a diagram for this case. In the `accellera_cdc::set_port -name rst1_i` command in Example 56, `-associated_from_clocks` is optional. With Example 56, a conflict for polarity should be reported on port `rst1_i` during top CDC analysis.

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

```
accellera_cdc::set_module -name mod0        Name of module

accellera_cdc::set_port -name rst1_n_o \    Name of port
    -direction output                  \    Direction of port
    -type async_reset                  \    Type of port
    -polarity low                      \    Active low asynchronous reset
    -associated_from_clocks clk0_i          External driver clock

accellera_cdc::set_module -name mod1        Name of module

accellera_cdc::set_port -name rst1_i \      Name of port
    -direction input                 \      Direction of port
    -type async_reset                \      Type of port
    -polarity high                   \      Active high asynchronous reset
    -associated_from_clocks clk0_i   \      External driver clock
    -associated_to_clocks clk0_i            Internal receiver clock
```

Example 56—Reset driven by correct clock domain but incorrect polarity

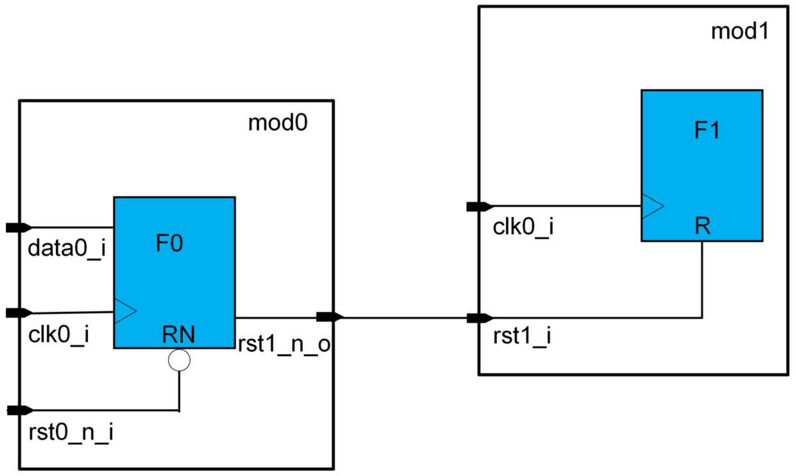

Figure 45—Reset driven by correct clock domain but incorrect polarity

A.4.3 Multiple ports conflicts

This section includes two examples of failure due to multiple ports.

A.4.3.1 Failure due to multiple clock domains with the same reset domain

The example in this section demonstrates a polarity conflict. Example 57 includes multiple input ports in the same clock domain but with different polarity driven by the same reset. Figure 46 provides a diagram for this case. In this example, a polarity conflict should be reported for the second port `rst1_n_i` but not for `rst0_n_i`.

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

```
accellera_cdc::set_module -name mod0      Name of module

accellera_cdc::set_port -name rst0_n_o \   Name of port
    -direction output                  \   Direction of port
    -type async_reset                  \   Type of port
    -polarity low                      \   Active low asynchronous reset
    -associated_from_clocks clk0_i         External driver clock

accellera_cdc::set_module -name mod1      Name of module

accellera_cdc::set_port -name rst0_n_i \   Name of port
    -direction input                   \   Direction of port
    -type async_reset                  \   Type of port
    -polarity low                      \   Active low asynchronous reset
    -associated_from_clocks clk0_i     \   External driver clock
    -associated_to_clocks clk0_i           Internal receiver clock

accellera_cdc::set_port -name rst1_i \     Name of port
    -direction input                 \     Direction of port
    -type async_reset                \     Type of port
    -polarity high                   \     Active high asynchronous reset
    -associated_from_clocks clk0_i   \     External driver clock
    -associated_to_clocks clk1_i           Internal receiver clock
```

Example 57—Failure due to reset polarity conflict

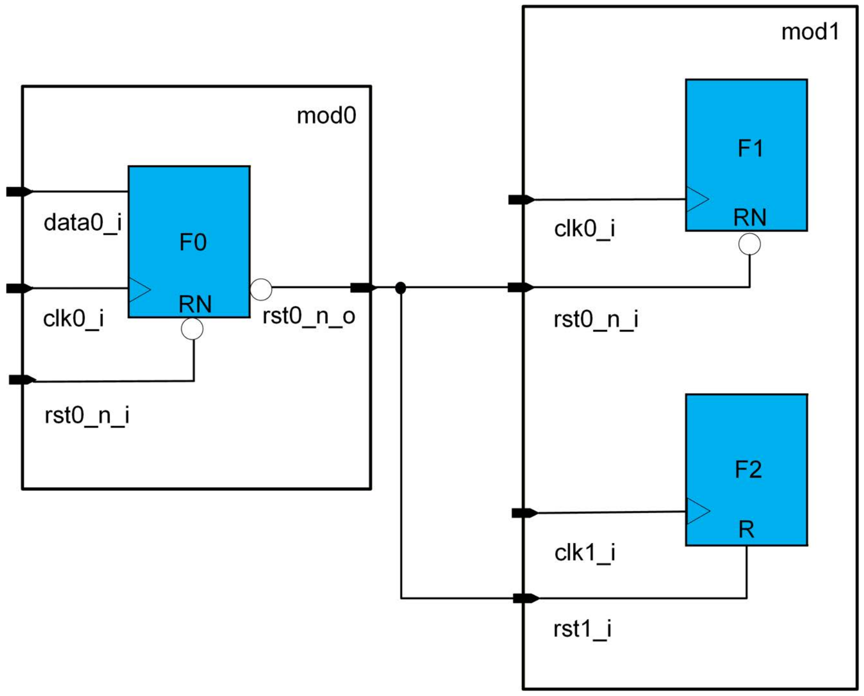

Figure 46—Failure due to reset polarity conflict

A.4.3.2 Failure due to inverter driving one of multiple resets

Example 58 demonstrates a failure due to multiple input ports in the same clock domain with the same polarity, but with one that is inverted and driven by the same reset. Figure 47 provides a diagram for this case. In this example, a conflict for polarity should be reported for the second port `rst1_i` and not for `rst0_n_i`.

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

```
accellera_cdc::set_module -name mod0       Name of module

accellera_cdc::set_port -name rst0_n_o \   Name of port
    -direction output                  \   Direction of port
    -type async_reset                  \   Type of port
    -polarity low                      \   Active low asynchronous reset
    -associated_from_clocks clk0_i         External driver clock

accellera_cdc::set_module -name mod1       Name of module

accellera_cdc::set_port -name rst0_n_i \   Name of port
    -direction input                   \   Direction of port
    -type async_reset                  \   Type of port
    -polarity low                      \   Active low asynchronous reset
    -associated_to_clocks clk0_i           Internal receiver clock

accellera_cdc::set_port -name rst1_i   \   Name of port
    -direction input                   \   Direction of port
    -type async_reset                  \   Type of port
    -polarity high                     \   Active high asynchronous reset
    -associated_to_clocks clk1_i           Internal receiver clock
```

Example 58—Failure due to inverter driving one of multiple resets

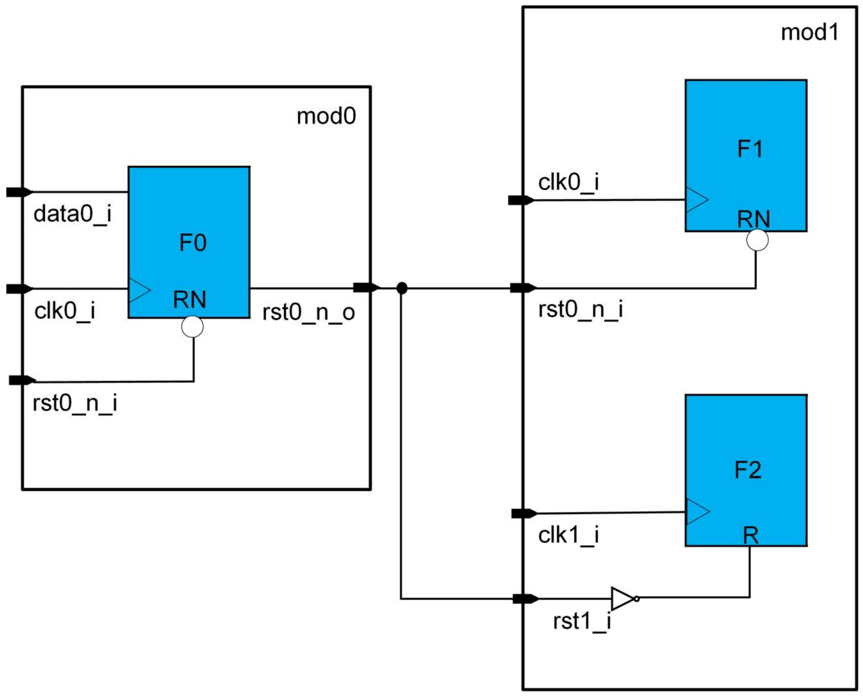

Figure 47—Failure due to inverter driving one of multiple resets

A.4.3.3 Failure due to non-reset signal feeding into reset signal

Example 59 demonstrates a failure due to a non-reset signal feeding into a reset signal. Figure 48 provides a diagram for this case. In this example, a conflict for the signal type should be reported for port `rst0_n_i`. This might be expected, but it shall be reviewed during top CDC analysis.

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

```
accellera_cdc::set_module -name mod0        Name of module

accellera_cdc::set_port -name data0_o \     Name of port
    -direction output                 \     Direction of port
    -type data                        \     Type of port
    -associated_from_clocks clk0_i          External driver clock

accellera_cdc::set_module -name mod1        Name of module

accellera_cdc::set_port -name rst0_n_i \    Name of port
    -direction input                   \    Direction of port
    -type async_reset                  \    Type of port
    -polarity low                      \    Active low asynchronous reset
    -associated_from_clocks clk0_i     \    External driver clock
    -associated_to_clocks clk0_i            Internal receiver clock
```

Example 59—Failure due to non-reset signal feeding into reset signal

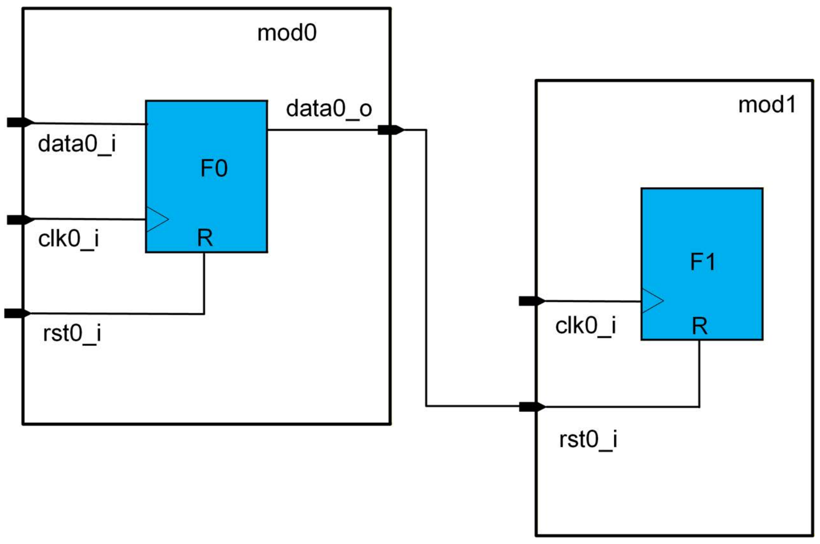

Figure 48—Failure due to non-reset signal feeding into reset signal

A.5 Synchronizer failure modes

This section shows attributes that may help a tool detect failure modes related to synchronizers and their reasons.

A.5.1 Failure mode with type data due to a missing synchronizer

Figure 49 depicts the propagation of metastability with a mean time between failures (MTBF) that is too low. The failure is due to a missing synchronizer. This scenario assumes that `data0_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 (see Example 60).

```
accellera_cdc::set_port data0_i \          Name of port
    -direction input            \          Direction of port
    -type data                  \          Type of port
    -associated_to_clocks clk0_i           Internal receiver's clocks for
                                           multiple fanouts of port data0_i
```

Example 60—Failure mode with type data due to a missing synchronizer

Independent from Example 60, the port `data0_i` in Figure 49 could be described as a `cdc_control` input (the synchronized data is not represented in the diagram). See Example 61.

`accellera_cdc::set_port -name data0_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 clk0_i`	Internal receiver's clocks for multiple fanouts of port data0_i

Example 61—Failure mode with type cdc_control due to a missing synchronizer

In both Example 60 and Example 61 above, the encompassing design in Figure 49 introduces a CDC violation. The driver is related to `clk1_i` but should be related to `clk0_i`.

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

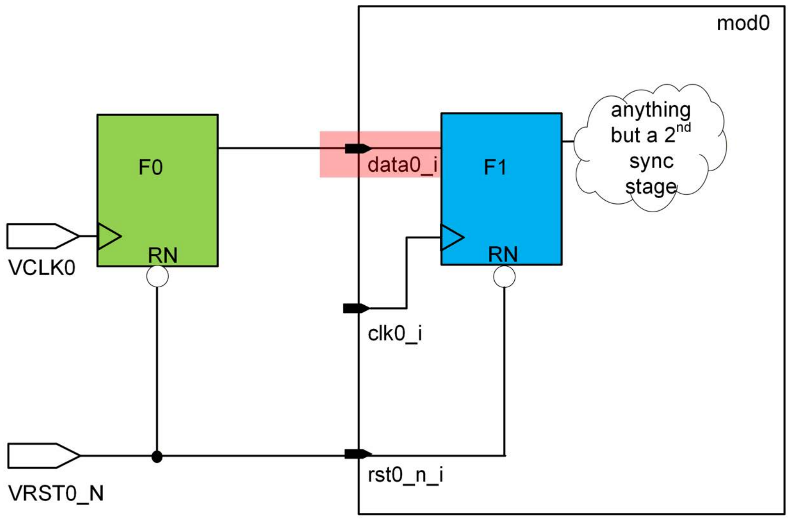

Figure 49—Failure mode due to a missing synchronizer

A.5.2 Failure mode due to a missing synchronized control

Figure 50 depicts the propagation of an intermediate (random or metastable) data value. The failure is due to a missing `cdc_control`.

Example 62 shows a model of the port `cdcq0_i`:

`accellera_cdc::set_port -name cdcq0_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 clk0_i`	Internal receiver's clocks for multiple fanouts of port cdcq0_i

Example 62—Failure mode due to a missing synchronized control

The encompassing design in Figure 50 introduces a CDC violation. The driver is related to `clk1_i`. Instead, the input signal should toggle related to `clk0_i`.

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

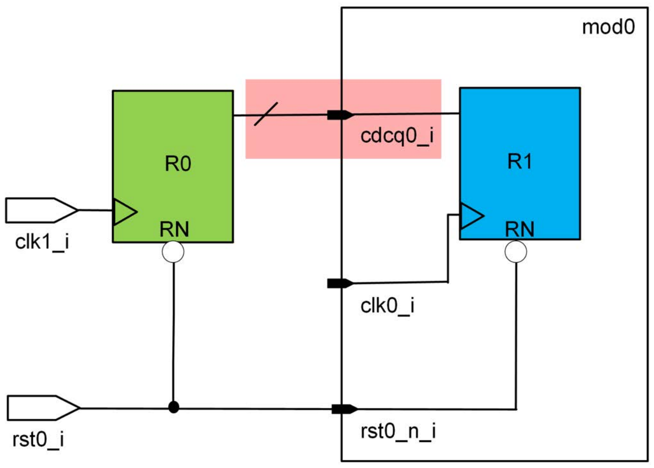

Figure 50—Failure mode due to a missing synchronized control

A.5.3 Failure mode due to a missing reset synchronizer

Figure 51 depicts an asynchronous deassertion of an asynchronous reset (CDC on reset tree). The failure is due to a missing reset synchronizer (see Example 63).

```
accellera_cdc::set_module -name mod0          Name of module

accellera_cdc::set_port -name rst0_n_i \      Name of port
    -direction input                   \      Direction of port
    -type async_reset                  \      Type of port
    -associated_to_clocks clk0_i       \      Internal receiver's clocks for multiple
                                       \      fanouts of port rst0_n_i
    -polarity low                             Active low asynchronous reset

accellera_cdc::set_module -name mod1          Name of module

accellera_cdc::set_port -name rst0_n_o \      Name of port
    -direction output                  \      Direction of port
    -type async_reset                  \      Type of port
    -associated_from_clocks clk1_i     \      External driver clock
    -polarity low                             Active low asynchronous reset
```

Example 63—Failure mode due to a missing reset synchronizer

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

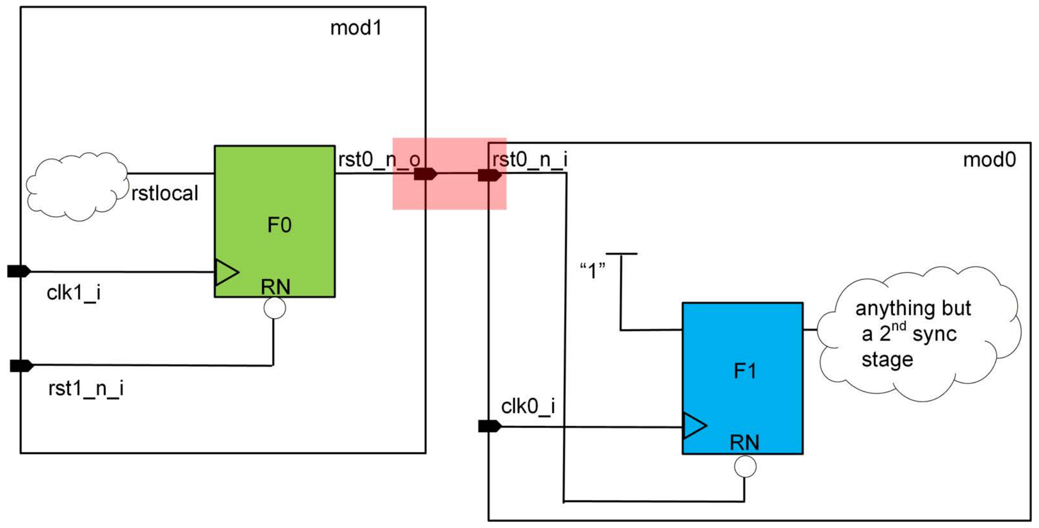

Figure 51—Failure mode due to a missing reset synchronizer

A.6 Failure mode due to combo logic driving clock/reset

We cannot have combo logic that can glitch, for instance, on a clock or a reset tree. Figure 52 shows that if the combo logic can glitch on the clock or reset network, there will be a violation.

```
accellera_cdc::set_module -name mod0        Name of module

accellera_cdc::set_port -name data0_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 clk0_i        \     External driver clock
-associated_inputs {data0_i data1_i}        Data ports controlled by this port

accellera_cdc::set_port -name data1_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_i        \     External driver clock
-associated_inputs {data2_i data3_i}        Data ports controlled by this port
```

Example 64—Failure mode due to combo logic driving clock/reset

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 C.

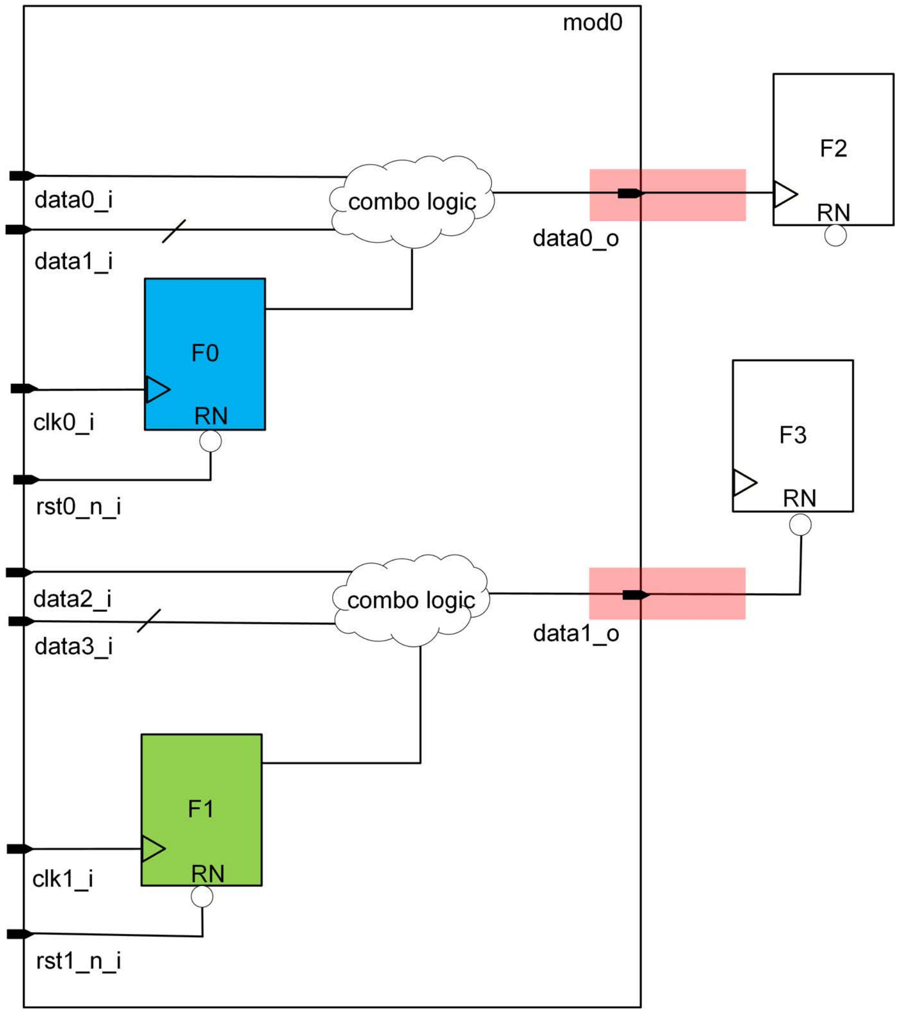

Figure 52—Failure mode due to combo logic driving clock/reset

Annex B

(informative)

Bibliography

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

Annex C

(informative)

CDC and RDC testcases

This annex lists unit testcases for early testing by EDA vendors. The list was created by the CDC Testing SWG members. Testcases are provided in both Verilog and VHDL formats. They will be available on GitHub® in the second quarter of 2026.[8]

Table 12—CDC and RDC testcases

Referenced Figure	Figure caption	Purpose of testcase
Figure 2	module domain attribute	Generic example for module declaration.
Figure 3	Specifying a virtual clock	The tool applies the name, direction, and type attributes to the intended point of the model.
Figure 4	port domain attribute: -associated_from_clocks	The tool applies the associated_from_clocks attribute to the intended point of the model.
Figure 5	Specifying a virtual_clock	The tool correctly applies a virtual clock.
Figure 6	port domain attributes: -associated_from_reset, -associated_to_reset	The tool applies the associated_from_reset and associated_to_reset attributes to the intended point of the model.
Figure 7	port domain attribute -ignore hanging for input port data1_i	The tool applies the ignore attribute for a hanging port.
Figure 8	port domain attribute -ignore blocked for input port data0_i	The tool applies the ignore attribute for a blocked port.
Figure 9	Example showing the need to describe a pseudo static behavior on the data crossing domains	The tool correctly uses the cdc_static attribute to avoid specifying unnecessary synchronization circuitry.
Figure 10	Example for -constant attribute	The tool correctly applies the constant attribute.
Figure 12	Internal synchronizer and internal combinatorial logic on paths through ports	The tool correctly uses the logic attribute and its various values on the associated port.
Figure 13	active_low reset on an input port with virtual -associated_from_clocks	The tool correctly applies attributes for active low asynchronous reset from an associated virtual clock. This IP will have an internal reset deassertion violation.

[8]GitHub® is a registered trademark of GitHub, Inc., registered in the United States and other countries.

Table 12—CDC and RDC testcases *(continued)*

Referenced Figure	Figure caption	Purpose of testcase
Figure 14	active_high reset on an input port with a virtual -associated_from_clocks	The tool correctly applies attributes for active high asynchronous reset from an associated virtual clock. This IP will have an internal reset deassertion violation.
Figure 15	active_high reset on an input port feeding a reset synchronizer	The tool recognizes a deassertion synchronizer.
Figure 16	active_high reset on an input port feeding an inverter	The tool applies attributes for an active high reset feeding an inverter.
Figure 17	active_high reset on input with -associated_to_clocks	The tool applies attributes for an active high reset with associated_to_clocks and associated_from_clocks.
Figure 22	Clock definition A	The tool applies attributes for two asynchronous clocks.
Figure 23	Clock definition B	The tool applies attributes for three asynchronous clocks.
Figure 24	Clock definition C	The tool applies attributes a mixture of synchronous and asynchronous clocks.
Figure 25	Internal synchronization logic	The tool applies attributes for internal reset synchronization circuitry.
Figure 26	Reset assertion sequence	The tool correctly applies attributes for reset synchronization that depends upon the sequence of multiple asynchronous resets.
Figure 27	Input port modeling: Internal data/ clock gating with internal data source	The tool correctly applies attributes for reset synchronization that depends upon internal data or clock gating with internally sourced data.
Figure 28	Input port modeling: Internal data/ clock gating with external data source	The tool correctly applies attributes for reset synchronization that depends upon internal data or clock gating with externally sourced data.
Figure 29	Output port modeling: External data/ clock gating for output port	The tool correctly applies attributes for output reset synchronization that depends upon external clock gating.
Figure 30	Output port modeling: Internal data gating	The tool correctly applies attributes for output reset synchronization that depends upon internal data gating.
Figure 42	Module mod0 with abstracted input ports	The tool properly represents a module with abstracted input ports.
Figure 43	Module mod0 with abstracted output ports	The tool properly represents a module with abstracted output ports.

Table 12—CDC and RDC testcases *(continued)*

Referenced Figure	Figure caption	Purpose of testcase
Figure 44	Unsynchronized reset crossing clock domains	The tool properly flags an unsynchronized reset crossing clock domains.
Figure 45	Reset driven by correct clock domain but incorrect polarity	The tool 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.
Figure 46	Failure due to reset polarity conflict	The tool properly flags multiple issues of a module with resets requiring both a different reset polarity and clock domain.
Figure 47	Failure due to inverter driving one of multiple resets	The tool properly flags multiple issues of a module with resets requiring both a different reset polarity from an inverter and a different clock domain.
Figure 48	Failure due to non-reset signal feeding into reset signal	The tool properly flags the issue of having a reset port driven by a non-reset signal.
Figure 49	Failure mode due to a missing synchronizer	The tool properly flags a data port that is registered but has no synchronizer.
Figure 50	Failure mode due to a missing synchronized control	The tool 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 51	Failure mode due to a missing reset synchronizer	The tool properly flags a reset port that has no synchronizer.
Figure 52	Failure mode due to combo logic driving clock/reset	The tool properly flags the issue of having a reset or clock port driven by a signal that comes from combinational logic.