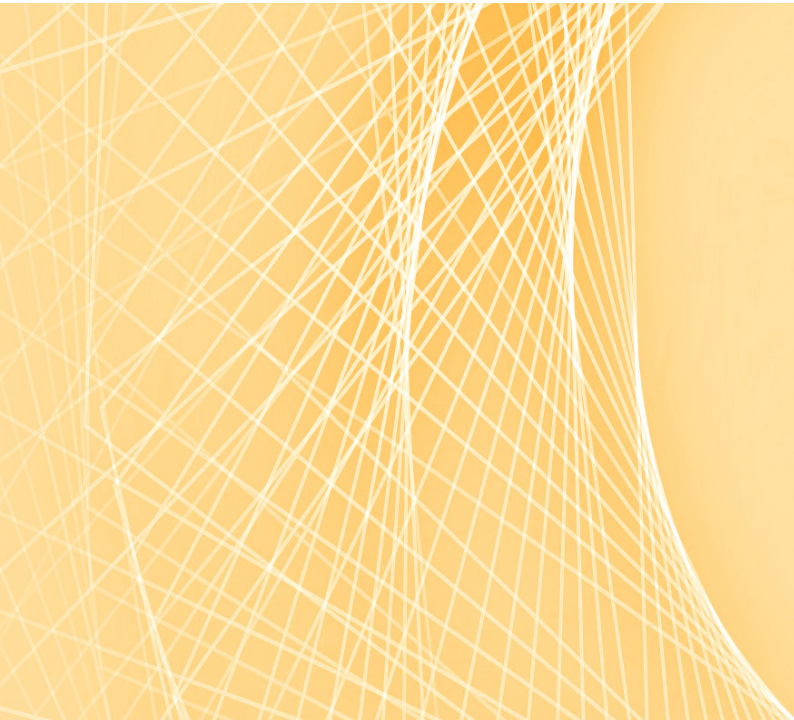


GateMate™ FPGA User Guide

Primitives Library





Cologne Chip AG
Eintrachtstr. 113
50668 Köln

Tel.: +49 (0) 221 / 91 24-0
Fax: +49 (0) 221 / 91 24-100

<https://colognechip.com>
info@colognechip.com

Copyright 2019 - 2024 Cologne Chip AG

All Rights Reserved

The information presented can not be considered as assured characteristics. Data can change without notice. Parts of the information presented may be protected by patent or other rights. Cologne Chip products are not designed, intended, or authorized for use in any application intended to support or sustain life, or for any other application in which the failure of the Cologne Chip product could create a situation where personal injury or death may occur.

Contents

About this Document	11
1 Introduction	13
2 I/O Buffers	15
2.1 CC_IBUF	16
2.2 CC_OBUF	19
2.3 CC_TOBUF	22
2.4 CC_IOBUF	25
2.5 CC_LVDS_IBUF	28
2.6 CC_LVDS_OBUF	31
2.7 CC_LVDS_TOBUF	34
2.8 CC_LVDS_IOBUF	37
2.9 CC_IDDR	40
2.10 CC_ODDR	42
3 Registers / Latches	45
3.1 CC_DFF	46
3.2 CC_DLT	49

4	LUT / MUX	51
4.1	Overview	52
4.2	CC_L2T4	53
4.3	CC_L2T5	55
4.4	CC_LUT{1,2}	57
4.5	CC_LUT{3,4} (for compatibility reasons)	59
4.6	CC_MX2	61
4.7	CC_MX4	62
5	Arithmetic Functions	65
5.1	Overview	66
5.2	CC_ADDF	67
5.3	CC_MULT	69
6	Block RAM	75
6.1	CC_BRAM_{20K,40K}	76
6.2	CC_FIFO_40K	94
7	Special Function Blocks	105
7.1	CC_BUFG	106
7.2	CC_USR_RSTN	107
7.3	CC_PLL	109
7.4	CC_PLL_ADV	114
7.5	CC_SERDES	117
7.6	CC_CFG_CTRL	125
	Acronyms	127

List of Figures

2.1	CC_IBUF primitive schematic	16
2.2	CC_OBUF primitive schematic	19
2.3	CC_T0BUF primitive schematic	22
2.4	CC_I0BUF primitive schematic	25
2.5	CC_LVDS_IBUF primitive schematic	28
2.6	CC_LVDS_OBUF primitive schematic	31
2.7	CC_LVDS_T0BUF primitive schematic	34
2.8	CC_LVDS_I0BUF primitive schematic	37
2.9	CC_IDDR primitive schematic	40
2.10	CC_IDDR function diagram	40
2.11	CC_ODDR primitive schematic	42
2.12	CC_ODDR function diagram	42
3.1	CC_DFF primitive schematic	46
3.2	CC_DLT primitive schematic	49
4.1	CC_L2T4 primitive schematic	53
4.2	Two independend CC_L2T4 primitives inside a CPE	53
4.3	CC_L2T5 primitive schematic	55
4.4	Combined CC_L2T4 and CC_L2T5 primitives forming an 8-input LUT-tree	55
4.5	CC_LUT primitive schematics	57
4.6	CC_LUT primitive schematics	59

4.7	CC_MX2 primitive schematic	61
4.8	CC_MX4 primitive schematic	62
5.1	CC_ADDF primitive schematic	67
5.2	CC_MULT primitive schematic	69
6.1	CC_BRAM_20K primitive schematic	76
6.2	CC_BRAM_40K primitive schematic	76
6.3	Timing diagram of a single read access	81
6.4	Timing diagram of a NO CHANGE access	82
6.5	Timing diagram of a WRITE THROUGH access	82
6.6	Logical data mapping of memory at physical address 0	84
6.7	CC_FIFO primitive schematic	94
6.8	Writing to an empty synchronous FIFO	99
6.9	Writing to an almost full synchronous FIFO	99
6.10	Reading from a full synchronous FIFO	100
6.11	Reading from an almost empty synchronous FIFO	100
6.12	Writing to an empty asynchronous FIFO	101
6.13	Writing to an almost full asynchronous FIFO	102
6.14	Reading from a full asynchronous FIFO	102
6.15	Reading from an almost empty asynchronous FIFO	103
7.1	CC_BUFG primitive schematic	106
7.2	CC_USR_RSTN primitive schematic	107
7.3	CC_PLL primitive schematic	109
7.4	CC_PLL output signals including frequency doubling on ports CLK180 and CLK270	109
7.5	CC_PLL_ADV primitive schematic	114
7.6	CC_SERDES primitive blockdiagram	118
7.7	CC_CFG_CTRL primitive schematic	125
7.8	Configuration from CPE array	125

List of Tables

2.1	CC_IBUF port description	16
2.2	CC_IBUF parameter description	17
2.3	CC_OBUF port description	19
2.4	CC_OBUF parameter description	20
2.5	CC_T0BUF port description	22
2.6	CC_T0BUF parameter description	23
2.7	CC_I0BUF port description	25
2.8	CC_I0BUF parameter description	26
2.9	CC_LVDS_IBUF port description	28
2.10	CC_LVDS_IBUF parameter description	29
2.11	CC_LVDS_OBUF port description	31
2.12	CC_LVDS_OBUF parameter description	32
2.13	CC_LVDS_T0BUF port description	34
2.14	CC_LVDS_T0BUF parameter description	35
2.15	CC_LVDS_I0BUF port description	37
2.16	CC_LVDS_I0BUF parameter description	38
2.17	CC_IDDR port description	40
2.18	CC_IDDR parameter description	41
2.19	CC_ODDR port description	43
2.20	CC_ODDR parameter description	43

3.1	CC_DFF port description	46
3.2	CC_DFF parameter description	47
3.3	CC_DLT port description	49
3.4	CC_DLT parameter description	49
4.1	CC_L2T4 port description	53
4.2	CC_L2T4 parameter description	54
4.3	CC_L2T5 port description	55
4.4	CC_L2T5 parameter description	56
4.5	CC_LUT{1, 2} port description	57
4.6	CC_LUT parameter descriptions for LUT-1 and LUT-2	57
4.7	CC_LUT{3, 4} port description	59
4.8	CC_LUT parameter descriptions for LUT-3 and LUT-4	59
4.9	CC_MX2 port description	61
4.10	CC_MX4 port description	62
5.1	CC_ADDF port description	67
5.2	CC_MULT port description	69
5.3	CC_MULT parameter description	69
6.1	20K configurations	77
6.2	40K configurations	77
6.3	CC_BRAM_{20K, 40K} port description	77
6.4	CC_BRAM_{20K, 40K} parameter description	78
6.4	Pin wiring in SDP 20K mode	79
6.5	Pin wiring in SDP 40K mode	79
6.6	Pin wiring in TDP 20K mode	80
6.7	Pin wiring in TDP 40K mode	80
6.8	Access combinations in NO CHANGE mode with {A B}_EN = 1 (SDP and TDP)	81
6.9	Access combinations in WRITE THROUGH mode with {A B}_EN = 1 (TDP only)	81
6.10	Block RAM initialization parameters	84
6.11	FIFO status flags	95
6.12	FIFO 40 bit data input concatenations	95
6.13	FIFO 80 bit data input concatenations	95
6.14	CC_FIFO_40K port description	96

6.15	CC_FIFO_40K parameter description	97
7.1	CC_BUFG port description	106
7.2	CC_USR_RSTN port description	107
7.3	CC_PLL port description	112
7.4	Dedicated clock pins	112
7.5	CC_PLL parameter description	113
7.6	CC_PLL_ADV port description	116
7.7	CC_PLL_ADV parameter description	116
7.8	CC_SERDES transmitter port description	119
7.9	CC_SERDES receiver port description	120
7.10	CC_SERDES register file port description	121
7.11	CC_SERDES PLL and miscellaneous port description	121
7.12	CC_SERDES tansmitter parameter list	122
7.13	CC_SERDES receiver parameter list	123
7.14	CC_SERDES PLL and miscellaneous parameter list	124
7.15	CC_CFG_CTRL port description	126

About this Document

This User Guide covers the Primitives Library of the Cologne Chip GateMate™ FPGA Series and is part of the GateMate™ documentation collection.

File `ug1001-gatemate1-attachment-latest.zip` [↗](#) is published together with this document. It contains the Verilog and VHDL instantiations `cc_serdes_inst.v` and `cc_serdes_inst.vhd` of the SerDes primitive.

For more information please refer to the following documents:

- Technology Brief of GateMate™ FPGA [↗](#)
- **DS1001** – GateMate™ FPGA CCGM1A1 Datasheet [↗](#)
- **UG1002** – GateMate™ FPGA Toolchain Installation User Guide [↗](#)

Cologne Chip provides a comprehensive technical support. Please visit our website for more information or contact our support team.

Revision History

This User Guide is constantly updated. The latest version of this document can be found following the link below:

UG1001 – GateMate™ FPGA Primitives Library [↗](#)

Date	Remarks
July 2024	PLL parameter type of REF_CLK and OUT_CLK changed from STRING to REAL in Section 7.3 from page 109.
May 2024	Parameter DYN_STAT_SELECT added for CC_FIFO_40K in Table 6.15 on page 97.
March 2024	VHDL instantiation code corrected in Section 4.2 on page 54.
October 2023	<ul style="list-style-type: none"> Added information in Section 6.2 from page 94. Added CC_SERDES primitive in Section 7.5 from page 117.
May 2023	Register width of F_ALMOST_FULL_OFFSET and F_ALMOST_EMPTY_OFFSET changed from 13 to 15 bit.
February 2023	Register initialization during configuration introduced.
January 2023	<ul style="list-style-type: none"> Added CC_USR_RSTN primitive in Section 7.2 from page 107. Changed LVDS buffer port names IN, IP, ON, OP, ION and IOP to I_N, I_P, O_N, O_P, IO_N and IO_P in Sections 2.5 to 2.8 from page 28. Changed LUT and LUT-tree output port names Y to 0 in Sections 4.2 to 4.5 from page 53.
December 2022	Double frequency option added to primitive CC_PLL in Section 7.3 from page 109.
November 2022	Improved VHDL inference examples in Section 6.1, part 'VHDL Inference Examples' from page 89.
September 2022	<ul style="list-style-type: none"> Removed CC_MUX8 primitive. Added VHDL instantiation and inference examples.
March 2022	Ports of primitive CC_FIFO_40K corrected (Figure 6.7 on page 94, Table 6.11 on page 95 and Table 6.14 on page 96).
...	
June 2021	Initial release.

Chapter 1

Introduction

This user guide covers the Primitives Library of the Cologne Chip GateMate™ series. During the FPGA synthesis process, a design described in a high level language such as Verilog is transformed into a gate level representation consisting of so called technology-mapped primitives. The resulting netlist is passed to the Place & Route tool for architecture-specific implementation and bitstream generation.

The Yosys Open SYnthesis Suite [↗](#) is the official synthesis tool and considered approved by Cologne Chip. The verilog simulation models [↗](#) for post-synthesis simulation can be found in the framework's repository.

This user guide is organized as follows:

- Chapter 2 describes the I/O components of the device interface.
- Chapter 3 describes the register primitives in the CPE and I/O blocks.
- Chapter 4 describes the LUT and MUX primitives in a CPE.
- Chapter 5 describes the arithmetic primitives that can be implemented in CPEs .
- Chapter 6 describes the block RAM and FIFO primitives.
- Chapter 7 describes special function primitives such as PLLs.

Chapter 2

I/O Buffers

Content Overview

2.1	CC_IBUF	16
2.2	CC_OBUF	19
2.3	CC_TOBUF	22
2.4	CC_IOBUF	25
2.5	CC_LVDS_IBUF	28
2.6	CC_LVDS_OBUF	31
2.7	CC_LVDS_TOBUF	34
2.8	CC_LVDS_IOBUF	37
2.9	CC_IDDR	40
2.10	CC_ODDR	42

2.1 CC_IBUF

The CC_IBUF primitive is an unidirectional single-ended input buffer. Input buffers must be inserted on all input signals that are directly connected to the top-level of a design. Additional I/O functionality for pull-up and pull-down resistors, input flip-flop¹, bus keeper functionality and Schmitt trigger can be configured using the primitive's parameters.

The I/O pads support low voltage CMOS (LVCMOS) standards up to 2.5 V nominal supply voltage compliant to the standard JESD8-7(A), which means that the general purpose input/output (GPIO) voltage can be 1.8 V \pm 0.15 V (normal range) or 1.2 V..1.95 V (wide range).

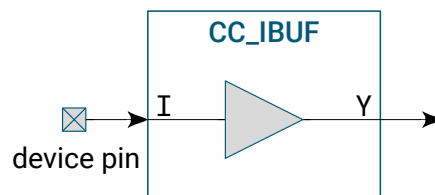


Figure 2.1: CC_IBUF primitive schematic

In general, input buffer primitives are inserted automatically by the synthesis tool. Usually, there is no need to instantiate them in the source code. However, if desired, these primitives can be instantiated manually in the source code as shown in the Verilog instantiation examples, e.g. to adjust the primitive's parameters manually.

Port Description

Table 2.1: CC_IBUF port description

Port	Direction	Width	Description
I	Input	1	Input from device pin
Y	Output	1	Output to FPGA-internal circuitry

¹ Please note, that flip-flops in I/O cells have no reset or set inputs.

Parameter Description

Table 2.2: CC_IBUF parameter description

Parameter	Type	Default	Description
PIN_NAME	STRING	"UNPLACED"	IO_<Dir><Bank>_<Pin><Pin#> Dir: N, E, S, W Bank: A, B, C Pin: A or B Pin#: 0..8
V_IO	STRING	"UNDEFINED"	I/O voltage: "1.2", "1.8" or "2.5" Volt
PULLUP	INT[0:0]	0	Enable or disable pull- ¹ or keeper feature. ²
PULLDOWN	INT[0:0]	0	
KEEPER	INT[0:0]	0	
SCHMITT_TRIGGER	INT[0:0]	0	Schmitt trigger (hysteresis) option: 0: disable (default) 1: enable
DELAY_IBF	INT[3:0]	0	Input delay parameter: 0..15
FF_IBF	INT[0:0]	0	Merge flip-flop into input cell ³ : 0: disable (default) 1: enable

¹ Pull resistances are nominal 50 kΩ each. See electrical characteristics in Datasheet [↗](#).

² Pullup, pulldown and keeper features are mutually exclusive.

³ Only plain flip-flops can be merged into input cells. Enable and reset signals are not supported.

Verilog Instantiation

```
CC_IBUF #(
  .PIN_NAME("IO_NB_A0"), // IO_<Dir><Bank>_<Pin><Pin#>
  .V_IO("1.8"),           // "1.2", "1.8" or "2.5" Volt
  .PULLUP(0),              // 0: disable, 1: enable
  .PULLDOWN(0),           // 0: disable, 1: enable
  .KEEPER(0),              // 0: disable, 1: enable
  .SCHMITT_TRIGGER(0),     // 0: disable, 1: enable
  .DELAY_IBF(4'd0),        // input delay: 0..15
  .FF_IBF(1'b0)            // 0: disable, 1: enable
) ibuf_inst (
  .I(I), // Input from device pin
  .Y(Y) // Output to FPGA-internal circuitry
);
```

VHDL Instantiation

```
ibuf_inst: CC_IBUF
generic map (
  PIN_NAME => "IO_NB_A0", -- IO_<Dir><Bank>_<Pin><Pin#>
  V_IO => "1.8", -- "1.2", "1.8" or "2.5" Volt
  PULLUP => 0, -- 0: disable, 1: enable
  PULLDOWN => 0, -- 0: disable, 1: enable
  KEEPER => 0, -- 0: disable, 1: enable
  SCHMITT_TRIGGER => 0, -- 0: disable, 1: enable
  DELAY_IBF => 0, -- input delay: 0..15
  FF_IBF => 0 -- 0: disable, 1: enable
)
port map (
  I => I, -- Input from device pin
  Y => Y -- Output to FPGA-internal circuitry
);
```


2.2 CC_OBUF

The CC_OBUF primitive is an unidirectional single-ended output buffer. Output buffers must be inserted on all output signals that are directly connected to the top-level of a design. Additional I/O functionality for drive strength, output flip-flop² and slew rate control can be configured using the primitive's parameters.

The I/O pads support LVCMOS standards up to 2.5 V nominal supply voltage compliant to the standard JESD8-7(A), which means that the GPIO voltage can be 1.8 V \pm 0.15 V (normal range) or 1.2 V .. 1.95 V (wide range).

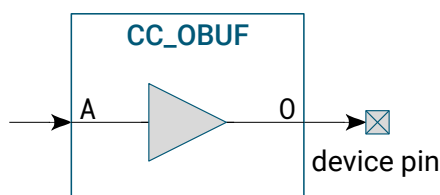


Figure 2.2: CC_OBUF primitive schematic

In general, output buffer primitives are inserted automatically by the synthesis tool. Usually, there is no need to instantiate them in the source code. However, if desired, these primitives can be instantiated manually in the source code as shown in the Verilog instantiation examples, e.g. to adjust the primitive's parameters manually.

Port Description

Table 2.3: CC_OBUF port description

Port	Direction	Width	Description
A	Input	1	Input from FPGA-internal circuitry
O	Output	1	Output to device pin

² Please note, that flip-flops in I/O cells have no reset or set inputs.

Parameter Description

Table 2.4: CC_OBUF parameter description

Parameter	Type	Default	Description
PIN_NAME	STRING	"UNPLACED"	IO_<Dir><Bank>_<Pin><Pin#> Dir: N, E, S, W Bank: A, B, C Pin: A or B Pin#: 0..8
V_IO	STRING	"UNDEFINED"	I/O voltage: "1.2", "1.8" or "2.5" Volt
DRIVE	STRING	"3"	Output drive strength: "3": 3 mA (default) "6": 6 mA "9": 9 mA "12": 12 mA
SLEW	STRING	"UNDEFINED"	Slew rate control: "SLOW" or "FAST"
DELAY_OBF	INT[3:0]	0	Output delay parameter: 0..15
FF_OBF	INT[0:0]	0	Merge flip-flop into output cell ¹ : 0: disable (default) 1: enable

¹ Only plain flip-flops and flip-flops with synchronous reset can be merged into output cells. Enable and asynchronous reset signals are not supported.

Verilog Instantiation

```
CC_OBUF #(
  .PIN_NAME("IO_NB_A0"), // IO_<Dir><Bank>_<Pin><Pin#>
  .V_IO("1.8"),          // "1.2", "1.8" or "2.5" Volt
  .DRIVE("3"),            // "3", "6", "9" or "12" mA
  .SLEW("SLOW"),          // "SLOW" or "FAST"
  .DELAY_OBF(4'd0),       // input delay: 0..15
  .FF_OBF(1'b0)           // 0: disable, 1: enable
) obuf_inst (
  .A(A), // Input from FPGA-internal circuitry
  .O(0)  // Output to device pin
);
```

VHDL Instantiation

```
obuf_inst: CC_OBUF
generic map (
  PIN_NAME  => "IO_NB_A0", -- IO_<Dir><Bank>_<Pin><Pin#>
  V_IO      => "1.8",      -- "1.2", "1.8" or "2.5" Volt
  DRIVE     => "3",        -- "3", "6", "9" or "12" mA
  SLEW      => "SLOW",     -- "SLOW" or "FAST"
  DELAY_OBF => 0,         -- input delay: 0..15
  FF_OBF    => 0           -- 0: disable, 1: enable
)
port map (
  A => A, -- Input from FPGA-internal circuitry
  O => O  -- Output to device pin
);
```

2.3 CC_TOBUF

The CC_TOBUF primitive is a controllable single-ended output buffer with an active-low output enable signal. This type of output buffers must be inserted on all output signals that are connected to the top-level of a design. When the output enable signal T is low, data from FPGA-internal circuitry on the input A is passed to the output O that is further connected to the device pins. When the output enable signal T is high, the output is high impedance.

Additional I/O functionality for drive strength, output flip-flop³ and slew rate control can be configured using the primitive's parameters.

The I/O pads support LVCMOS standards up to 2.5 V nominal supply voltage compliant to the standard JESD8-7(A), which means that the GPIO voltage can be 1.8 V \pm 0.15 V (normal range) or 1.2 V .. 1.95 V (wide range).

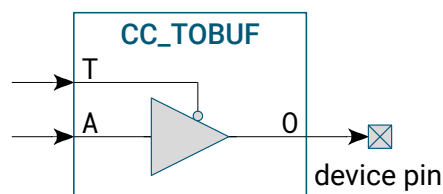


Figure 2.3: CC_TOBUF primitive schematic

In general, tri-state controllable output buffer primitives are inserted automatically by the synthesis tool. Usually, there is no need to instantiate them in the source code. However, if desired, these primitives can be instantiated manually in the source code as shown in the Verilog instantiation examples, e.g. to adjust the primitive's parameters manually.

Port Description

Table 2.5: CC_TOBUF port description

Port	Direction	Width	Description
A	Input	1	Input from FPGA-internal circuitry
T	Input	1	Active-low output enable signal from FPGA-internal circuitry
O	Output	1	Output to device pin, tri-state if T = 1

³ Please note, that flip-flops in I/O cells have no reset or set inputs.

Parameter Description

Table 2.6: *CC_TOBUF parameter description*

Parameter	Type	Default	Description
PIN_NAME	STRING	"UNPLACED"	IO_<Dir><Bank>_<Pin><Pin#> Dir: N, E, S, W Bank: A, B, C Pin: A or B Pin#: 0..8
V_IO	STRING	"UNDEFINED"	I/O voltage: "1.2", "1.8" or "2.5" Volt
DRIVE	STRING	"3"	Output drive strength: "3": 3 mA (default) "6": 6 mA "9": 9 mA "12": 12 mA
SLEW	STRING	"UNDEFINED"	Slew rate control: "SLOW" or "FAST"
PULLUP	INT[0:0]	0	Enable or disable pull- ¹ or keeper feature. ²
PULLDOWN	INT[0:0]	0	
KEEPER	INT[0:0]	0	
DELAY_OBF	INT[3:0]	0	Output delay parameter: 0..15
FF_OBF	INT[0:0]	0	Merge flip-flop into output cell ³ : 0: disable (default) 1: enable

¹ Pull resistances are nominal 50 kΩ each. See electrical characteristics in Datasheet [↗](#).

² Pullup, pulldown and keeper features are mutually exclusive.

³ Only plain flip-flops and flip-flops with synchronous reset can be merged into output cells. Enable and asynchronous reset signals are not supported.

Verilog Instantiation

```
CC_Tobuf #(
  .PIN_NAME("IO_NB_A0"), // IO_<Dir><Bank>_<Pin><Pin#>
  .V_IO("1.8"),          // "1.2", "1.8" or "2.5" Volt
  .DRIVE("3"),            // "3", "6", "9" or "12" mA
  .SLEW("SLOW"),         // "SLOW" or "FAST"
  .PULLUP(0),             // 0: disable, 1: enable
  .PULLDOWN(0),          // 0: disable, 1: enable
  .KEEPER(0),             // 0: disable, 1: enable
  .DELAY_OBF(4'd0),      // input delay: 0..15
  .FF_OBF(1'b0)          // 0: disable, 1: enable
) tobuf_inst (
  .A(A), // Input from FPGA-internal circuitry
  .T(T), // Active-low output enable signal from FPGA-internal circuitry
  .O(0)  // Output to device pin, tri-state if T=1
);
```

VHDL Instantiation

```
tobuf_inst: CC_Tobuf
generic map (
  PIN_NAME => "IO_NB_A0", -- IO_<Dir><Bank>_<Pin><Pin#>
  V_IO     => "1.8",      -- "1.2", "1.8" or "2.5" Volt
  DRIVE    => "3",        -- "3", "6", "9" or "12" mA
  SLEW     => "SLOW",     -- "SLOW" or "FAST"
  PULLUP    => 0,         -- 0: disable, 1: enable
  PULLDOWN  => 0,         -- 0: disable, 1: enable
  KEEPER    => 0,         -- 0: disable, 1: enable
  DELAY_OBF => 0,         -- input delay: 0..15
  FF_OBF    => 0          -- 0: disable, 1: enable
)
port map (
  A => A, -- Input from FPGA-internal circuitry
  T => T, -- Active-low output enable signal from FPGA-internal circuitry
  O => O  -- Output to device pin, tri-state if T=1
);
```

2.4 CC_IOBUF

The CC_IOBUF primitive is a bidirectional single-ended I/O buffer with an active-low output enable signal. I/O buffers must be inserted on all bidirectional signals that are directly connected to the top-level of a design. When the output enable signal T is low, data from FPGA-internal circuitry on the input A is passed to the bidirectional port IO that is further connected to the device pins. When the output enable signal T is high, data from the bidirectional port IO is passed to the output Y that is further FPGA-internal circuitry.

In input mode, the values set in the parameters for pull-up and pull-down resistors, input flip-flop, bus keeper functionality and Schmitt trigger option apply. In output mode, parameters for drive strength, output flip-flop and slew rate control apply.⁴

The I/O pads support LVCMOS standards up to 2.5 V nominal supply voltage compliant to the standard JESD8-7(A), which means that the GPIO voltage can be 1.8 V \pm 0.15 V (normal range) or 1.2 V .. 1.95 V (wide range).

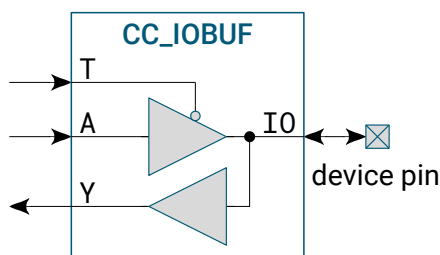


Figure 2.4: CC_IOBUF primitive schematic

In general, I/O buffer primitives are inserted automatically by the synthesis tool. Usually, there is no need to instantiate them in the source code. However, if desired, these primitives can be instantiated manually in the source code as shown in the Verilog instantiation examples, e.g. to adjust the primitive's parameters manually.

Port Description

Table 2.7: CC_IOBUF port description

Port	Direction	Width	Description
A	Input	1	Input from FPGA-internal circuitry
T	Input	1	Active-low output enable signal from FPGA-internal circuitry
Y	Output	1	Output to FPGA-internal circuitry
IO	Inout	1	Bidirectional in- or output to device pin

⁴ Please note, that flip-flops in I/O cells have no reset or set inputs.

Parameter Description

Table 2.8: CC_I0BUF parameter description

Parameter	Type	Default	Description
PIN_NAME	STRING	"UNPLACED"	IO_<Dir><Bank>_<Pin><Pin#> Dir: N, E, S, W Bank: A, B, C Pin: A or B Pin#: 0..8
V_IO	STRING	"UNDEFINED"	I/O voltage: "1.2", "1.8" or "2.5" Volt
DRIVE	STRING	"3"	Output drive strength: "3": 3 mA (default) "6": 6 mA "9": 9 mA "12": 12 mA
SLEW	STRING	"UNDEFINED"	Slew rate control: "SLOW" or "FAST"
PULLUP	INT[0:0]	0	Enable or disable pull- ¹ or keeper feature. ²
PULLDOWN	INT[0:0]	0	
KEEPER	INT[0:0]	0	
SCHMITT_TRIGGER	INT[0:0]	0	Schmitt trigger (hysteresis) option: 0: disable (default) 1: enable
DELAY_IBF	INT[3:0]	0	Input delay parameter: 0..15
DELAY_OBF	INT[3:0]	0	Output delay parameter: 0..15
FF_IBF	INT[0:0]	0	Merge input flip-flop into cell ³ : 0: disable (default) 1: enable
FF_OBF	INT[0:0]	0	Merge output flip-flop into cell ⁴ : 0: disable (default) 1: enable

¹ Pull resistances are nominal 50 kΩ each. See electrical characteristics in Datasheet [↗](#).

² Pullup, pulldown and keeper features are mutually exclusive.

³ Only plain flip-flops can be merged into input cells. Enable and reset signals are not supported.

⁴ Only plain flip-flops and flip-flops with synchronous reset can be merged into output cells. Enable and asynchronous reset signals are not supported.

Verilog Instantiation

```
CC_IOBUF #(
  .PIN_NAME("IO_NB_A0"), // IO_<Dir><Bank>_<Pin><Pin#>
  .V_IO("1.8"),           // "1.2", "1.8" or "2.5" Volt
  .DRIVE("3"),             // "3", "6", "9" or "12" mA
  .SLEW("SLOW"),          // "SLOW" or "FAST"
  .PULLUP(0),              // 0: disable, 1: enable
  .PULLDOWN(0),            // 0: disable, 1: enable
  .KEEPER(0),              // 0: disable, 1: enable
  .SCHMITT_TRIGGER(0),     // 0: disable, 1: enable
  .DELAY_IBF(4'd0),        // input delay: 0..15
  .DELAY_OBF(4'd0),        // input delay: 0..15
  .FF_IBF(1'b0),           // 0: disable, 1: enable
  .FF_OBF(1'b0)            // 0: disable, 1: enable
) iobuf_inst (
  .A(A), // Input from FPGA-internal circuitry
  .T(T), // Active-low output enable signal from FPGA-internal circuitry
  .Y(Y), // Output to FPGA-internal circuitry
  .IO(IO) // Bidirectional inout to device pin
);
```

VHDL Instantiation

```
iobuf_inst: CC_IOBUF
generic map (
  PIN_NAME => "IO_NB_A0", -- IO_<Dir><Bank>_<Pin><Pin#>
  V_IO => "1.8",           -- "1.2", "1.8" or "2.5" Volt
  DRIVE => "3",            -- "3", "6", "9" or "12" mA
  SLEW => "SLOW",          -- "SLOW" or "FAST"
  PULLUP => 0,              -- 0: disable, 1: enable
  PULLDOWN => 0,            -- 0: disable, 1: enable
  KEEPE => 0,               -- 0: disable, 1: enable
  SCHMITT_TRIGGER => 0,     -- 0: disable, 1: enable
  DELAY_IBF => 0,           -- input delay: 0..15
  DELAY_OBF => 0,           -- input delay: 0..15
  FF_IBF => 0,              -- 0: disable, 1: enable
  FF_OBF => 0               -- 0: disable, 1: enable
)
port map (
  A => A, -- Input from FPGA-internal circuitry
  T => T, -- Active-low output enable signal from FPGA-internal
    ↪ circuitry
  Y => Y, -- Output to FPGA-internal circuitry
  IO => IO -- Bidirectional inout to device pin
);
```

2.5 CC_LVDS_IBUF

The CC_LVDS_IBUF primitive is an unidirectional differential input buffer. Just like conventional input buffers those elements belong to input signals that are directly connected to the top-level. All low-voltage differential signaling (LVDS) pads are compliant to the LVDS 2.5 V standard. It can further operate down to 1.8 V nominal supply voltage. A LVDS on-chip termination resistor of 100 Ω can be enabled using the corresponding parameter.

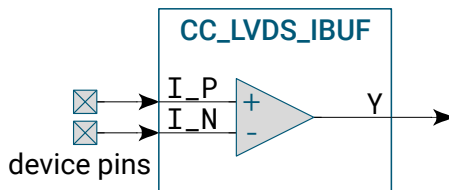


Figure 2.5: CC_LVDS_IBUF primitive schematic

Port Description

Table 2.9: CC_LVDS_IBUF port description

Port	Direction	Width	Description
I_P	Input	1	Positive differential input from device pin
I_N	Input	1	Negative differential input from device pin
Y	Output	1	Data output to FPGA-internal circuitry

Parameter Description

Table 2.10: *CC_LVDS_IBUF parameter description*

Parameter	Type	Default	Description
PIN_NAME_{P N}	STRING	"UNPLACED"	IO_<Dir><Bank>_<Pin><Pin#> Dir: N, E, S, W Bank: A, B, C Pin: A or B Pin#: 0..8
V_IO	STRING	"UNDEFINED"	I/O voltage: "1.8" or "2.5" Volt
LVDS_RTERM	INT[0:0]	0	On-chip termination resistor ¹ enable: 0: disable (default) 1: enable
DELAY_IBF	INT[3:0]	0	Input delay parameter: 0..15
FF_IBF	INT[0:0]	0	Merge flip-flop into input cell ² : 0: disable (default) 1: enable

¹ LVDS termination resistance is nominal 100 Ω . See electrical characteristics in Datasheet [↗](#).

² Only plain flip-flops can be merged into input cells. Enable and reset signals are not supported.

Verilog Instantiation

```
CC_LVDS_IBUF #(
  .PIN_NAME_P("IO_NB_A0"), // IO_<Dir><Bank>_<Pin><Pin#>
  .PIN_NAME_N("IO_NB_B0"), // secondary diff. signal
  .V_IO("1.8"),             // "1.8" or "2.5" Volt
  .LVDS_RTERM(0),           // 0: disable, 1: enable
  .DELAY_IBF(4'd0),         // input delay: 0..15
  .FF_IBF(1'b0)             // 0: disable, 1: enable
) lvds_ibuf_inst (
  .I_P(I_P), // Positive diff. input from device pin
  .I_N(I_N), // Negative diff. input from device pin
  .Y(Y)      // Output to FPGA-internal circuitry
);
```

VHDL Instantiation

```
lvds_ibuf_inst: CC_LVDS_IBUF
generic map (
  PIN_NAME_P => "IO_NB_A0", -- IO_<Dir><Bank>_<Pin><Pin#>
  PIN_NAME_N => "IO_NB_B0", -- secondary diff. signal
  V_IO       => "1.8",      -- "1.8" or "2.5" Volt
  LVDS_RTERM => 0,         -- 0: disable, 1: enable
  DELAY_IBF  => 0,         -- input delay: 0..15
  FF_IBF     => 0          -- 0: disable, 1: enable
)
port map (
  I_P => I_P, -- Positive diff. input from device pin
  I_N => I_N, -- Negative diff. input from device pin
  Y   => Y    -- Output to FPGA-internal circuitry
);
```

2.6 CC_LVDS_OBUF

The CC_LVDS_OBUF primitive is an unidirectional differential output buffer. Just like conventional output buffers those elements belong to output signals that are directly connected to the top-level. All LVDS pads are compliant to the LVDS 2.5 V standard. It can further operate down to 1.8 V nominal supply voltage. The LVDS output current can be configured using the corresponding parameter.

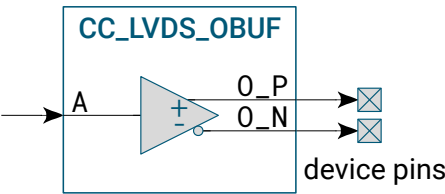


Figure 2.6: CC_LVDS_OBUF primitive schematic

Port Description

Table 2.11: CC_LVDS_OBUF port description

Port	Direction	Width	Description
A	Input	1	Input from FPGA-internal circuitry
O_P	Output	1	Positive differential output to device pin
O_N	Output	1	Negative differential output to device pin

Parameter Description

Table 2.12: *CC_LVDS_OBUF parameter description*

Parameter	Type	Default	Description
PIN_NAME_{P N}	STRING	"UNPLACED"	IO_<Dir><Bank>_<Pin><Pin#> Dir: N, E, S, W Bank: A, B, C Pin: A or B Pin#: 0..8
V_IO	STRING	"UNDEFINED"	I/O voltage: "1.8" or "2.5" Volt
LVDS_BOOST	INT[0:0]	0	Configure LVDS output current boost: 0: 3.2 mA nominal current (default) 1: 6.4 mA increased current
DELAY_OBF	INT[3:0]	0	Output delay parameter: 0..15
FF_OBF	INT[0:0]	0	Merge flip-flop into output cell ¹ : 0: disable (default) 1: enable

¹ Only plain flip-flops and flip-flops with synchronous reset can be merged into output cells. Enable and asynchronous reset signals are not supported.

Verilog Instantiation

```
CC_LVDS_OBUF #(
    .PIN_NAME_P("IO_NB_A0"), // IO_<Dir><Bank>_<Pin><Pin#>
    .PIN_NAME_N("IO_NB_B0"), // secondary diff. signal
    .V_IO("1.8"),           // "1.8" or "2.5" Volt
    .LVDS_BOOST(0),          // 0: 3.2 mA, 1: 6.4 mA
    .DELAY_OBF(4'd0),        // input delay: 0..15
    .FF_OBF(1'b0)            // 0: disable, 1: enable
) lvds_obuf_inst (
    .A(A),                   // Input from FPGA-internal circuitry
    .O_P(O_P),              // Positive diff. output to device pin
    .O_N(O_N)               // Negative diff. output to device pin
);
```

VHDL Instantiation

```
lvds_obuf_inst: CC_LVDS_OBUF
generic map (
  PIN_NAME_P => "IO_NB_A0", -- IO_<Dir><Bank>_<Pin><Pin#>
  PIN_NAME_N => "IO_NB_B0", -- secondary diff. signal
  V_IO       => "1.8",      -- "1.8" or "2.5" Volt
  LVDS_BOOST => 0,          -- 0: 3.2 mA, 1: 6.4 mA
  DELAY_OBF  => 0,          -- input delay: 0..15
  FF_OBF     => 0           -- 0: disable, 1: enable
)
port map (
  A   => A,    -- Input from FPGA-internal circuitry
  O_P => O_P,  -- Positive diff. output to device pin
  O_N => O_N   -- Negative diff. output to device pin
);
```

2.7 CC_LVDS_T0BUF

The CC_LVDS_T0BUF primitive is a controllable differential output buffer with an active-low output enable signal. This type of output buffers must be inserted on all differential output signal pairs that are connected to the top-level of a design. When the output enable signal T is low, data from FPGA-internal circuitry on the input A is passed to the differential outputs O_P and O_N that are further connected to the device pins. When the output enable signal T is high, the output is high impedance.

All LVDS pads are compliant to the LVDS 2.5 V standard. It can further operate down to 1.8 V nominal supply voltage. The LVDS output current can be configured using the corresponding parameter.

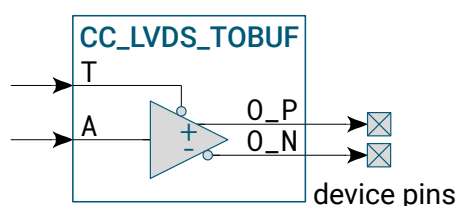


Figure 2.7: CC_LVDS_T0BUF primitive schematic

Port Description

Table 2.13: CC_LVDS_T0BUF port description

Port	Direction	Width	Description
A	Input	1	Input from FPGA-internal circuitry
T	Input	1	Active-low output enable signal from FPGA-internal circuitry
O_P	Output	1	Positive differential output to device pin
O_N	Output	1	Negative differential output to device pin

Parameter Description

Table 2.14: *CC_LVDS_Tobuf parameter description*

Parameter	Type	Default	Description
PIN_NAME_{P N}	STRING	"UNPLACED"	IO_<Dir><Bank>_<Pin><Pin#> Dir: N, E, S, W Bank: A, B, C Pin: A or B Pin#: 0..8
V_IO	STRING	"UNDEFINED"	I/O voltage: "1.8" or "2.5" Volt
LVDS_BOOST	INT[0:0]	0	Configure LVDS output current boost: 0: 3.2 mA nominal current (default) 1: 6.4 mA increased current
DELAY_OBF	INT[3:0]	0	Output delay parameter: 0..15
FF_OBF	INT[0:0]	0	Merge flip-flop into output cell ¹ : 0: disable (default) 1: enable

¹ Only plain flip-flops and flip-flops with synchronous reset can be merged into output cells. Enable and asynchronous reset signals are not supported.

Verilog Instantiation

```
CC_LVDS_Tobuf #(
    .PIN_NAME_P("IO_NB_A0"), // IO_<Dir><Bank>_<Pin><Pin#>
    .PIN_NAME_N("IO_NB_B0"), // secondary diff. signal
    .V_IO("1.8"),           // "1.8" or "2.5" Volt
    .LVDS_RTERM(0),          // 0: disable, 1: enable
    .LVDS_BOOST(0),          // 0: 3.2 mA, 1: 6.4 mA
    .DELAY_OBF(4'd0),        // input delay: 0..15
    .FF_OBF(1'b0)            // 0: disable, 1: enable
) lvds_tobuf_inst (
    .A(A), // Input from FPGA-internal circuitry
    .T(T), // Active-low output enable signal from FPGA-internal
           ↪ circuitry
    .O_P(O_P), // Positive diff. output to device pin
    .O_N(O_N) // Negative diff. output to device pin
);
```

VHDL Instantiation

```
lvds_tobuf_inst: CC_LVDS_TOBUF
generic map (
  PIN_NAME_P => "IO_NB_A0", -- IO_<Dir><Bank>_<Pin><Pin#>
  PIN_NAME_N => "IO_NB_B0", -- secondary diff. signal
  V_IO       => "1.8",      -- "1.8" or "2.5" Volt
  LVDS_RTERM => 0,          -- 0: disable, 1: enable
  LVDS_BOOST => 0,          -- 0: 3.2 mA, 1: 6.4 mA
  DELAY_OBF  => 0,          -- input delay: 0..15
  FF_OBF     => 0           -- 0: disable, 1: enable
)
port map (
  A  => A,  -- Input from FPGA-internal circuitry
  T  => T,  -- Active-low output enable signal from FPGA-internal
        ↪ circuitry
  O_P => O_P, -- Positive diff. output to device pin
  O_N => O_N -- Negative diff. output to device pin
);
```


2.8 CC_LVDS_IOBUF

The CC_LVDS_IOBUF primitive is an unidirectional differential I/O buffer. I/O buffers must be inserted on all bidirectional signals that are directly connected to the top-level of a design. When the output enable signal T is low, data from FPGA-internal circuitry on the input A is passed to the bidirectional differential ports IO_P and IO_N that are further connected to the device pins. When the output enable signal T is high, data from the bidirectional differential ports IO_P and IO_N is passed to the output Y that is further FPGA-internal circuitry.

In input mode, the on-chip LVDS termination resistor of $100\ \Omega$ is enabled according to the value set in the parameters. In output mode, the values set in the parameters for output drive strength and slew rate control apply. All LVDS pads are compliant to the LVDS 2.5 V standard. It can further operate down to 1.8 V nominal supply voltage. The LVDS output current can be configured using the corresponding parameter.

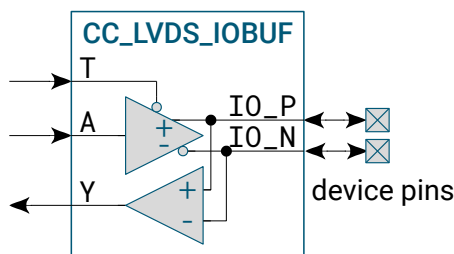


Figure 2.8: CC_LVDS_IOBUF primitive schematic

Port Description

Table 2.15: CC_LVDS_IOBUF port description

Port	Direction	Width	Description
A	Input	1	Input from FPGA-internal circuitry
T	Input	1	Active-low output enable signal from FPGA-internal circuitry
Y	Output	1	Output to FPGA-internal circuitry
IO_P	Inout	1	Positive differential bidirectional signal to device pin
IO_N	Inout	1	Negative differential bidirectional signal to device pin

Parameter Description

Table 2.16: *CC_LVDS_IOBUF parameter description*

Parameter	Type	Default	Description
PIN_NAME_{P, N}	STRING	"UNPLACED"	IO_<Dir><Bank>_<Pin><Pin#> Dir: N, E, S, W Bank: A, B, C Pin: A or B Pin#: 0..8
V_IO	STRING	"UNDEFINED"	I/O voltage: "1.8" or "2.5" Volt
LVDS_RTERM	INT[0:0]	0	On-chip termination resistor ¹ enable: 0: disable (default) 1: enable
LVDS_BOOST	INT[0:0]	0	Configure LVDS output current boost: 0: 3.2 mA nominal current (default) 1: 6.4 mA increased current
DELAY_IBF	INT[3:0]	0	Input delay parameter: 0..15
DELAY_OBF	INT[3:0]	0	Output delay parameter: 0..15
FF_IBF	INT[0:0]	0	Merge input flip-flop into cell ² : 0: disable (default) 1: enable
FF_OBF	INT[0:0]	0	Merge output flip-flop into cell ³ : 0: disable (default) 1: enable

¹ LVDS termination resistance is nominal 100 Ω. See electrical characteristics in Datasheet [↗](#).

² Only plain flip-flops can be merged into input cells. Enable and reset signals are not supported.

³ Only plain flip-flops and flip-flops with synchronous reset can be merged into output cells. Enable and asynchronous reset signals are not supported.

Verilog Instantiation

```

CC_LVDS_IOBUF #(
  .PIN_NAME_P("IO_NB_A0"), // IO_<Dir><Bank>_<Pin><Pin#>
  .PIN_NAME_N("IO_NB_B0"), // secondary diff. signal
  .V_IO("1.8"),           // "1.8" or "2.5" Volt
  .LVDS_RTERM(0),          // 0: disable, 1: enable
  .LVDS_BOOST(0),          // 0: 3.2 mA, 1: 6.4 mA
  .DELAY_IBF(4'd0),        // input delay: 0..15
  .DELAY_OBF(4'd0),        // input delay: 0..15
  .FF_IBF(1'b0),           // 0: disable, 1: enable
  .FF_OBF(1'b0),           // 0: disable, 1: enable
) lvds_iobuf_inst (
  .A(A),                  // Input from FPGA-internal circuitry
  .T(T),                  // Active-low output enable signal from FPGA-internal
    ↳ circuitry
  .Y(Y),                  // Output to FPGA-internal circuitry
  .IO_P(IO_P),            // Positive diff. bidirectional signal to device pin
  .IO_N(IO_N)             // Negative diff. bidirectional signal to device pin
);

```

VHDL Instantiation

```

lvds_iobuf_inst: CC_LVDS_IOBUF
generic map (
  PIN_NAME_P => "IO_NB_A0", -- IO_<Dir><Bank>_<Pin><Pin#>
  PIN_NAME_N => "IO_NB_B0", -- secondary diff. signal
  V_IO       => "1.8",      -- "1.8" or "2.5" Volt
  LVDS_RTERM => 0,          -- 0: disable, 1: enable
  LVDS_BOOST => 0,          -- 0: 3.2 mA, 1: 6.4 mA
  DELAY_IBF  => 0,          -- input delay: 0..15
  DELAY_OBF  => 0,          -- input delay: 0..15
  FF_IBF     => 0,          -- 0: disable, 1: enable
  FF_OBF     => 0,          -- 0: disable, 1: enable
)
port map (
  A  => A,    -- Input from FPGA-internal circuitry
  T  => T,    -- Active-low output enable signal from FPGA-internal
    ↳ circuitry
  Y  => Y,    -- Output to FPGA-internal circuitry
  IO_P => IO_P, -- Positive diff. bidirectional signal to device pin
  IO_N => IO_N -- Negative diff. bidirectional signal to device pin
);

```

2.9 CC_IDDR

The CC_IDDR primitive is a dedicated input register for double data rate (DDR) support, i.e. for capturing external data on both positive and negative clock edges. Input D must be attached to an input buffer. Further I/O functions such as delay, pull-up or pull-down control etc. are therefore still available. Additional registering in the input buffer is not supported. All GPIOs support DDR functionality.

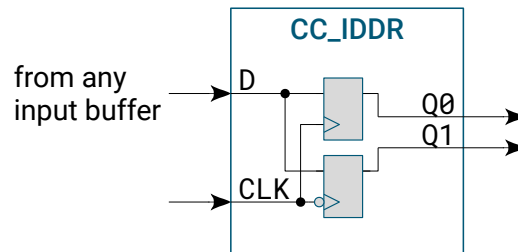


Figure 2.9: CC_IDDR primitive schematic

Input DDR registers sample incoming data from the input buffer to the FPGA-internal circuitry on both the rising and falling edges of the clock signal. The input data is fed into the FPGA-internal circuitry via the two signals Q0 and Q1, as shown in Figure 2.10.

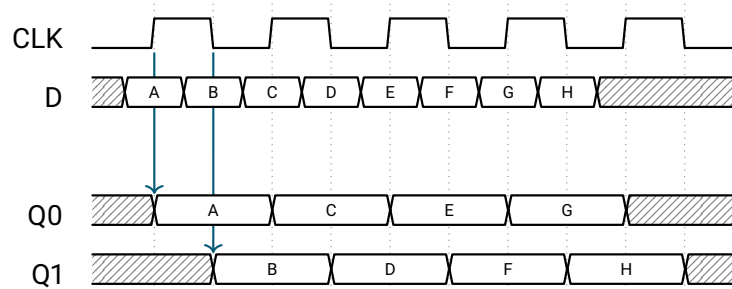


Figure 2.10: CC_IDDR function diagram

Port Description

Table 2.17: CC_IDDR port description

Port	Direction	Width	Description
D	Input	1	Data input from device pin
CLK	Input	1	Clock signal input
Q0	Output	1	Data output to FPGA-internal circuitry
Q1	Output	1	Data output to FPGA-internal circuitry

Parameter Description

Table 2.18: *CC_IDDR parameter description*

Parameter	Type	Default	Description
CLK_INV	INT[0:0]	0	Clock polarity for Q0: 0: rising edge (default) 1: falling edge

Verilog Instantiation

```
CC_IDDR #(
    .CLK_INV(1'b0) // 0: rising edge, 1: falling edge
) iddr_inst (
    .D(D),          // Data input
    .CLK(CLK),      // Clock input
    .Q0(Q0),        // Q0 data output
    .Q1(Q1)         // Q1 data output
);
```

VHDL Instantiation

```
iddr_inst: CC_IDDR
generic map (
    CLK_INV => 0 -- 0: rising edge, 1: falling edge
)
port map (
    D    => D,    -- Data input
    CLK  => CLK,  -- Clock input
    Q0   => Q0,   -- Q0 data output
    Q1   => Q1    -- Q1 data output
);
```

2.10 CC_ODDR

The CC_ODDR primitive is a dedicated output register for DDR support, i.e. for transferring data on both positive and negative clock edges. Both outputs Q0 and Q1 must be attached to an output buffer. Further I/O functions such as delay, drive strength control etc. are therefore still available. Additional registering in the output buffer is not supported. All GPIOs support DDR functionality.

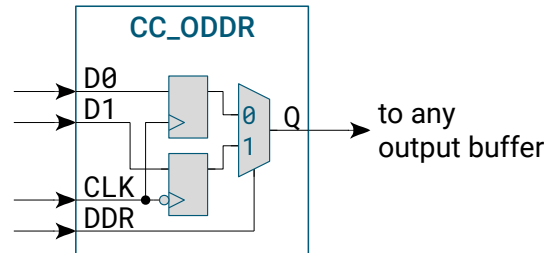


Figure 2.11: CC_ODDR primitive schematic

Output DDR registers sample outgoing data from the FPGA-internal circuitry to the output buffer on both the rising and falling edges of the clock signal. The output data is fed from the FPGA-internal circuitry via the two signals D0 and D1, as shown in Figure 2.12.

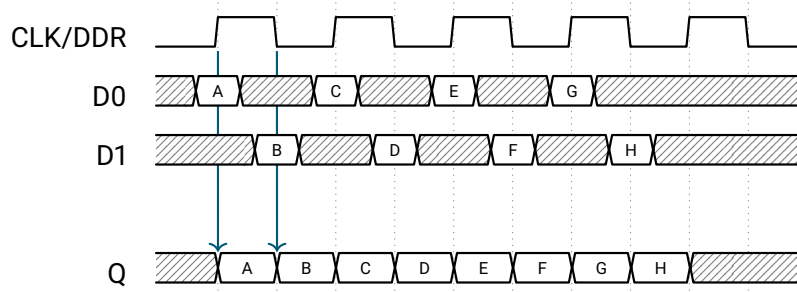


Figure 2.12: CC_ODDR function diagram

Port Description

Table 2.19: *CC_ODDR port description*

Port	Direction	Width	Description
D0	Input	1	Data input from FPGA-internal circuitry
D1	Input	1	Data input from FPGA-internal circuitry
CLK	Input	1	Clock signal input to flip-flops
DDR	Input	1	Clock signal input to flip-flop switch
Q	Output	1	Data output to device pin

Parameter Description

Table 2.20: *CC_ODDR parameter description*


Parameter	Type	Default	Description
CLK_INV	INT[0:0]	0	Clock polarity: 0: rising edge (default) 1: falling edge

Verilog Instantiation

```
CC_ODDR #(
    .CLK_INV(1'b0) // 0: rising edge, 1: falling edge
) oddr_inst (
    .D0(D0),      // D0 data input
    .D1(D1),      // D1 data input
    .CLK(CLK),    // Clock input
    .DDR(DDR),    // FF switch
    .Q(Q)         // Data output
);
```

VHDL Instantiation

```
oddr_inst: CC_ODDR
generic map (
  CLK_INV => 0 -- 0: rising edge, 1: falling edge
)
port map (
  D0  => D0,  -- D0 data input
  D1  => D1,  -- D1 data input
  CLK => CLK, -- Clock input
  DDR => DDR, -- FF switch
  Q   => Q    -- Data output
);
```

Chapter 3

Registers / Latches

Content Overview

3.1	CC_DFF	46
3.2	CC_DLT	49

3.1 CC_DFF

The CC_DFF primitive is a D-type flip-flop with a clock enable pin and a configurable pin for asynchronous set / reset. The clock polarity as well as the inversion of the EN and SR pins are controlled by parameters. Additional circuitry for any other type of flip-flop, i.e. one with synchronous set / reset, must be generated by the synthesis tool. During mapping, flip-flops and additional logic can be combined in one Cologne Programmable Element (CPE). Moreover, up to two flip-flops with identical parameter settings and identical CLK, EN and SR input signals can be combined into a single CPE. Initialization during configuration is supported.

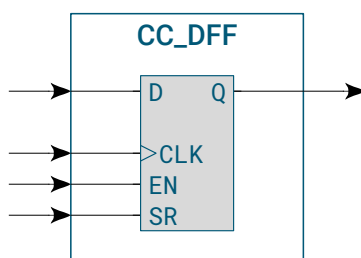


Figure 3.1: CC_DFF primitive schematic

Port Description

Table 3.1: CC_DFF port description

Port	Direction	Width	Description
D	Input	1	Data input
CLK	Input	1	Clock signal
EN	Input	1	Clock enable signal
SR	Input	1	Configurable asynchronous set / reset signal
Q	Output	1	Data output

Parameter Description

Table 3.2: *CC_DFF parameter description*

Parameter	Type	Default	Description
CLK_INV	INT[0:0]	0	Clock polarity: 0: rising edge (default) 1: falling edge
EN_INV	INT[0:0]	0	Enable signal inverting: 0: disable (default) 1: enable
SR_INV	INT[0:0]	0	Set / reset signal inverting: 0: disable (default) 1: enable
SR_VAL	INT[0:0]	0	Set / reset value: 0: reset to zero (default) 1: set to one
INIT	INT[0:0]	X	Initial value of Q output after configuration.

Verilog Instantiation

```
CC_DFF #(
  .CLK_INV(1'b0), // 0: rising edge, 1: falling edge
  .EN_INV(1'b0),  // 0: active high, 1: active low
  .SR_INV(1'b0),  // 0: active high, 1: active low
  .SR_VAL(1'b0),  // 0: reset, 1: set
  .INIT(1'bx)     // x: unknown, 0: zero, 1: one
) dff_inst (
  .D(D),          // Data input
  .CLK(CLK),      // Clock input
  .EN(EN),        // Clock-enable input
  .SR(SR),        // Set/reset input
  .Q(Q)           // Q output
);
```

VHDL Instantiation

```
dff_inst: CC_DFF
generic map (
  CLK_INV => 0, -- 0: rising edge, 1: falling edge
  EN_INV  => 0, -- 0: active high, 1: active low
  SR_INV  => 0, -- 0: active high, 1: active low
  SR_VAL  => 0, -- 0: reset, 1: set
  INIT    => X  -- X: unknown, 0: zero, 1: one
)
port map (
  D  => D,    -- Data input
  CLK => CLK,  -- Clock input
  EN  => EN,   -- Clock-enable input
  SR  => SR,   -- Set/reset input
  Q   => Q     -- Q output
);
```

3.2 CC_DLT

The CC_DLT primitive is a D-type latch with a configurable set / reset pin. The inversion of the EN and SR pins are controlled by parameters. Up to two latches with identical parameter settings can be combined into a single CPE.

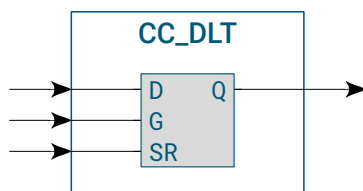


Figure 3.2: CC_DLT primitive schematic

Port Description

Table 3.3: CC_DLT port description

Port	Direction	Width	Description
D	Input	1	Data input
G	Input	1	Enable input
SR	Input	1	Configurable asynchronous set / reset signal
Q	Output	1	Data output

Parameter Description

Table 3.4: CC_DLT parameter description

Parameter	Type	Default	Description
G_INV	INT[0:0]	0	Enable signal inverting: 0: disable (default) 1: enable
SR_INV	INT[0:0]	0	Set / reset signal inverting: 0: disable (default) 1: enable
SR_VAL	INT[0:0]	0	Set / reset value: 0: reset to zero (default) 1: set to one
INIT	INT[0:0]	X	Initial value of Q output after configuration.

Verilog Instantiation

```
CC_DLT #(
    .G_INV(1'b0),    // 0: active high, 1: active low
    .SR_INV(1'b0),    // 0: active high, 1: active low
    .SR_VAL(1'b0),    // 0: reset, 1: set
    .INIT(1'bx)       // x: unknown, 0: zero, 1: one
) dlt_inst (
    .D(D),           // Data input
    .G(G),           // Enable input
    .SR(SR),         // Set/reset input
    .Q(Q)            // Q output
);
```

VHDL Instantiation

```
dlt_inst: CC_DLT
generic map (
    E_INV  => 0, -- 0: active high, 1: active low
    SR_INV => 0, -- 0: active high, 1: active low
    SR_VAL => 0, -- 0: reset, 1: set
    INIT   => X  -- X: unknown, 0: zero, 1: one
)
port map (
    D  => D, -- Data input
    E  => E, -- Enable input
    SR => SR, -- Set/reset input
    Q  => Q  -- Q output
);
```

Chapter 4

LUT / MUX

Content Overview

4.1	Overview	52
4.2	CC_L2T4	53
4.3	CC_L2T5	55
4.4	CC_LUT{1,2}	57
4.5	CC_LUT{3,4} (for compatibility reasons)	59
4.6	CC_MX2	61
4.7	CC_MX4	62

4.1 Overview

The Cologne Programmable Element (CPE) implements several 2-input lookup table (LUT-2) functions. These are arranged in so-called *LUT-trees* to combine up to 8 input functions.

The synthesis tool can automatically map logical functions to matching LUT-trees.

Conventional 3-input lookup table (LUT-3) and 4-input lookup table (LUT-4) are available for compatibility.

Finally, multiplexer primitives with 2 and 4 inputs are available.

4.2 CC_L2T4

The CC_L2T4 primitive holds a 4-input LUT-tree function and is automatically inferred by the synthesis tool. The 4-input LUT-tree consists of three LUT-2 elements as shown in Figure 4.1.

As a CPE has 8 inputs, there are two positions where a CC_L2T4 primitive can be located in the CPE, or two independent CC_L2T4 primitives can be used at the same time in one CPE as shown in Figure 4.2.

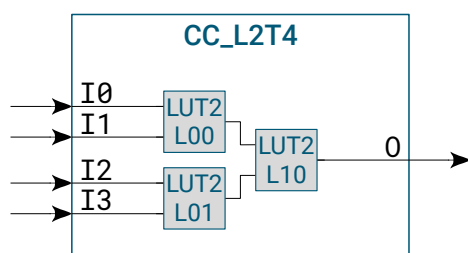


Figure 4.1: CC_L2T4 primitive schematic

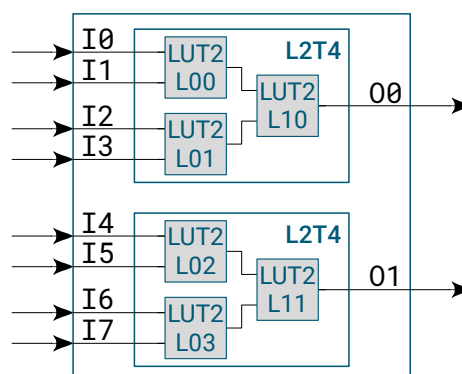


Figure 4.2: Two independent CC_L2T4 primitives inside a CPE

Port Description

Table 4.1: CC_L2T4 port description

Port	Direction	Width	Description
I0	Input	1	LUT L00 data input 0
I1	Input	1	LUT L00 data input 1
I2	Input	1	LUT L01 data input 0
I3	Input	1	LUT L01 data input 1
0	Output	1	L2T4 output

Parameter Description

Table 4.2: *CC_L2T4 parameter description*

Parameter	Type	Default	Description
INIT_L00	INT[3:0]	0	LUT L00 configuration
INIT_L01	INT[3:0]	0	LUT L01 configuration
INIT_L10	INT[3:0]	0	LUT L10 configuration

Verilog Instantiation

```
CC_L2T4 #(
    .INIT_L00(4'h0), // LUT L00 configuration
    .INIT_L01(4'h0), // LUT L01 configuration
    .INIT_L10(4'h0) // LUT L10 configuration
) l2t4_inst (
    .I0(I0), .I1(I1), // L00 inputs
    .I2(I2), .I3(I3), // L01 inputs
    .O(0)           // L2T4 output
);
```

VHDL Instantiation

```
l2t4_inst: CC_L2T4
generic map (
    INIT_L00 => X"0" -- LUT L00 configuration
    INIT_L01 => X"0" -- LUT L01 configuration
    INIT_L10 => X"0" -- LUT L10 configuration
)
port map (
    I0 => I0, I1 => I1, -- L00 inputs
    I2 => I2, I3 => I3, -- L01 inputs
    O => O             -- L2T4 output
);
```

4.3 CC_L2T5

The CC_L2T5 primitive holds a 5-input LUT-tree function where the fifth input I4 is connected to L20. It is automatically inferred by the synthesis tool and consists of four LUT-2 elements as shown in Figure 4.3.

By combining CC_L2T5 with a CC_L2T4 primitive, it can be used to build LUT-tree functions with up to 8 inputs in a single CPE during implementation as shown in Figure 4.4. The optional 4-input tree-structure lookup table (L2T4) function can be routed-out via 00 if it has a fanout greater than 1.

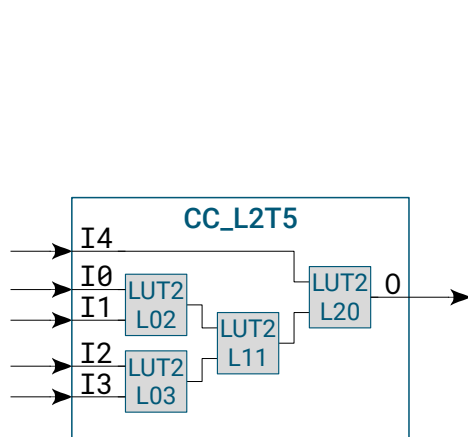


Figure 4.3: CC_L2T5 primitive schematic

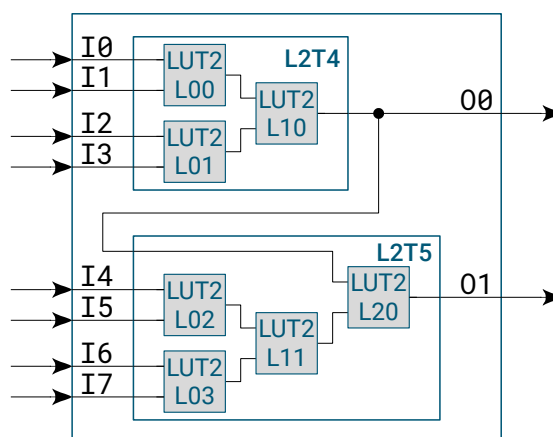


Figure 4.4: Combined CC_L2T4 and CC_L2T5 primitives forming an 8-input LUT-tree

Port Description

Table 4.3: CC_L2T5 port description

Port	Direction	Width	Description
I0	Input	1	LUT L02 data input 0
I1	Input	1	LUT L02 data input 1
I2	Input	1	LUT L03 data input 0
I3	Input	1	LUT L03 data input 1
I4	Input	1	LUT L20 data input 0
0	Output	1	L2T5 output

Parameter Description

Table 4.4: CC_L2T5 parameter description

Parameter	Type	Default	Description
INIT_L02	INT[3:0]	0	LUT L02 configuration
INIT_L03	INT[3:0]	0	LUT L03 configuration
INIT_L11	INT[3:0]	0	LUT L11 configuration
INIT_L20	INT[3:0]	0	LUT L20 configuration

Verilog Instantiation

```
CC_L2T5 #(
    .INIT_L02(4'h0), // LUT L02 configuration
    .INIT_L03(4'h0), // LUT L03 configuration
    .INIT_L11(4'h0), // LUT L11 configuration
    .INIT_L20(4'h0)  // LUT L20 configuration
) l2t5_inst (
    .I0(I0), .I1(I1), // L02 inputs
    .I2(I2), .I3(I3), // L03 inputs
    .I4(I4),           // L20 input
    .O(0)              // L2T5 output
);
```

VHDL Instantiation

```
l2t5_inst: CC_L2T5
generic map (
    INIT_L02 => X"0", -- LUT L02 configuration
    INIT_L03 => X"0", -- LUT L03 configuration
    INIT_L11 => X"0", -- LUT L11 configuration
    INIT_L20 => X"0"  -- LUT L20 configuration
)
port map (
    I0 => I0, I1 => I1, -- LUT L02 data inputs
    I2 => I2, I3 => I3, -- LUT L03 data inputs
    I4 => I4,           -- LUT L20 data input
    O => O              -- L2T5 output
);
```

4.4 CC_LUT{1,2}

The CC_LUT1 and CC_LUT2 primitives hold one or two input functions and are automatically inferred by the synthesis tool.

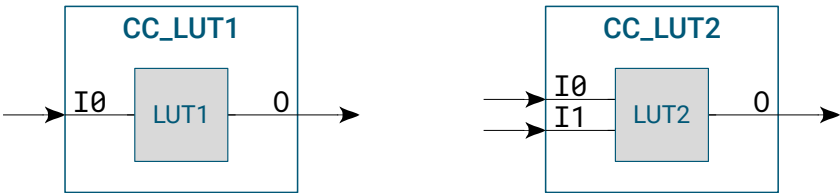


Figure 4.5: CC_LUT primitive schematics

Port Description

Table 4.5: CC_LUT{1,2} port description

Port	Direction	Width	Description
I0	Input	1	Data input
I1	Input	1	Data input for CC_LUT2 only
0	Output	1	LUT output

Parameter Description

Table 4.6: CC_LUT parameter descriptions for LUT-1 and LUT-2

Parameter	Type	Default	Description
INIT	INT[1:0]	0	CC_LUT1 configuration: Any 2-bit value
INIT	INT[3:0]	0	CC_LUT2 configuration: Any 4-bit value

Verilog Instantiation

```
CC_LUT1 #(
  .INIT(2'bXX) // LUT1 configuration
) lut1_inst (
  .I0(I0),
  .O(0)
);

CC_LUT2 #(
  .INIT(4'hX) // LUT2 configuration
) lut2_inst (
  .I0(I0), .I1(I1),
  .O(0)
);
```

VHDL Instantiation

```
lut1_inst: CC_LUT1
  generic map (
    INIT => "X" -- LUT1 configuration
  )
  port map (
    I0 => I0,
    O  => O
  );

lut2_inst: CC_LUT2
  generic map (
    INIT => "X" -- LUT2 configuration
  )
  port map (
    I0 => I0,
    I1 => I1,
    O  => O
  );
```

4.5 CC_LUT{3, 4} (for compatibility reasons)

Unlike typical LUTs, a CPE LUT-tree cannot realize the full set of functions, but a subset. The CC_LUT3 and CC_LUT4 primitives exist for compatibility reasons. During implementation, the GateMate Place & Route tool finds the best possible mapping into LUT-trees.

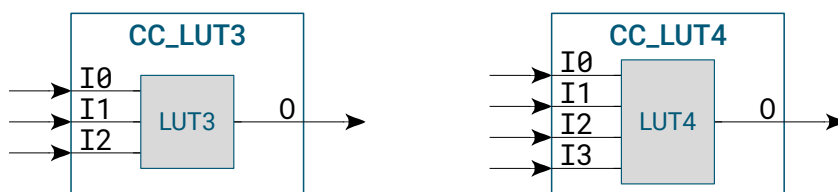


Figure 4.6: CC_LUT primitive schematics

Port Description

Table 4.7: CC_LUT{3, 4} port description

Port	Direction	Width	Description
I0	Input	1	Data input
I1	Input	1	Data input
I2	Input	1	Data input
I3	Input	1	Data input for CC_LUT4 only
0	Output	1	LUT output

Parameter Description

Table 4.8: CC_LUT parameter descriptions for LUT-3 and LUT-4

Parameter	Type	Default	Description
INIT	INT[7:0]	0	CC_LUT3 configuration: Any 8-bit value
INIT	INT[15:0]	0	CC_LUT4 configuration: Any 16-bit value

Verilog Instantiation

```
// NOTE: for compatibility reasons

CC_LUT3 #(
  .INIT(8'hXX) // LUT3 configuration
) lut3_inst (
  .I0(I0), .I1(I1), .I2(I2),
  .O(0)
);

CC_LUT4 #(
  .INIT(16'hXXXX) // LUT4 configuration
) lut4_inst (
  .I0(I0), .I1(I1), .I2(I2), .I3(I3),
  .O(0)
);
```

VHDL Instantiation

```
lut3_inst: CC_LUT3
  generic map (
    INIT => X"XX" -- LUT3 configuration
  )
  port map (
    I0 => I0,
    I1 => I1,
    I2 => I2,
    O  => O
  );

lut4_inst: CC_LUT4
  generic map (
    INIT => X"XXXX" -- LUT4 configuration
  )
  port map (
    I0 => I0,
    I1 => I1,
    I2 => I2,
    I3 => I3,
    O  => O
  );
```


4.6 CC_MX2

The CC_MX2 primitive is a 2-to-1 multiplexer. The two data inputs D0 and D1 and select input S0 lead directly to the multiplexer inputs.

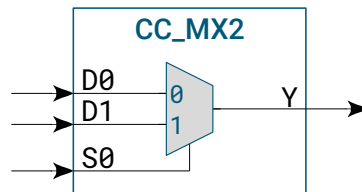


Figure 4.7: CC_MX2 primitive schematic

Port Description

Table 4.9: CC_MX2 port description

Port	Direction	Width	Description
D0	Input	1	Data input
D1	Input	1	Data input
S0	Input	1	Select input
Y	Output	1	Multiplexer output

Verilog Instantiation

```
// 2-to-1 MUX
CC_MX2 mx2_inst (
    .D0(D0), // Data inputs
    .D1(D1), // Data inputs
    .S0(S0), // Select input
    .Y(Y)    // MUX output
);
```

VHDL Instantiation

```
mx2_inst: CC_MX2
port map (
    D0 => D0, -- Data inputs
    D1 => D1, -- Data inputs
    S0 => S0, -- Select input
    Y  => Y   -- MUX output
);
```

4.7 CC_MX4

The CC_MX4 primitive is a dedicated 4-to-1 multiplexer. The four data inputs D0 .. D3 and two select inputs S0 and S1 lead directly to the multiplexer inputs.

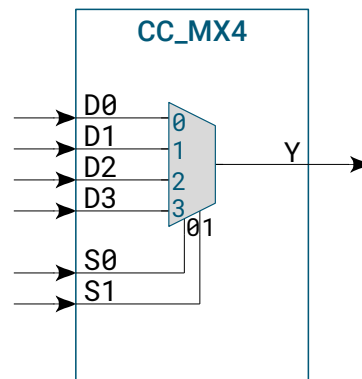


Figure 4.8: CC_MX4 primitive schematic

Port Description

Table 4.10: CC_MX4 port description


Port	Direction	Width	Description
D0	Input	1	Data input
D1	Input	1	Data input
D2	Input	1	Data input
D3	Input	1	Data input
S0	Input	1	Select input
S1	Input	1	Select input
Y	Output	1	Multiplexer output

Verilog Instantiation

```
// 4-to-1 MUX
CC_MX4 mx4_inst (
    .D0(D0), .D1(D1), // Data inputs
    .D2(D2), .D3(D3), // Data inputs
    .S0(S0), .S1(S1), // Select inputs
    .Y(Y)             // MUX output
);
```

VHDL Instantiation

```
mx4_inst: CC_MX4
port map (
    D0 => D0, D1 => D1, -- Data inputs
    D2 => D2, D3 => D3, -- Data inputs
    S0 => S0, S1 => S1, -- Select inputs
    Y  => Y              -- MUX output
);
```

Chapter 5

Arithmetic Functions

Content Overview

5.1	Overview	66
5.2	CC_ADDF	67
5.3	CC_MULT	69

5.1 Overview

Any Cologne Programmable Element (CPE) can have a 1-bit or 2-bit adder function. Neighboring CPEs can form chains of any length in vertical or horizontal orientation.

Furthermore, neighboring carry chains can form an array multiplier where each CPE sets up a 2×2 multiplier element. This feature enables building multipliers of any size.

5.2 CC_ADDF

The CC_ADDF primitive is a full adder using dedicated logic and routing resources inside and between CPE cells for fast arithmetic logic. During mapping, two cascaded CC_ADDF primitives can be combined into a single CPE forming a two-bit full adder.

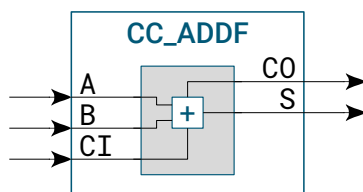


Figure 5.1: CC_ADDF primitive schematic

Adders are automatically inferred by the synthesis tool as shown in the Verilog inference examples.

Port Description

Table 5.1: CC_ADDF port description

Port	Direction	Width	Description
A	Input	1	Summand
B	Input	1	Summand
CI	Input	1	Carry cascade
CO	Output	1	Carry cascade
S	Output	1	Sum

Verilog Instantiation

```
CC_ADDF addf_inst(
  .A(A),    // Adder input A
  .B(B),    // Adder input B
  .CI(CI),  // Carry-in
  .CO(CO),  // Carry-out
  .S(S)     // Sum output A+B
);
```

VHDL Instantiation

```
addf_inst: CC_ADDF
port map (
  A => A,  -- Adder input A
  B => B,  -- Adder input B
  CI => CI, -- Carry-in
  CO => CO, -- Carry-out
  S => S   -- Sum output A+B
);
```


5.3 CC_MULT

The CC_MULT primitive is a scalable, signed multiplier with inputs of any width that extends over several CPEs and can be placed at any position in the CPE array. The M-bit value represented in input A is multiplied by the N-bit value represented in input B. Output P is the M+N-bit product of A and B. Values may be represented as either unsigned or two's complement signed with bit zero as least significant bit (LSB). With additional CC_DFF primitives, optional pipeline registers can be implemented independently at the inputs and output. Depending on the number of pipeline stages, validity of the output data gets delayed by one or two clock cycles.

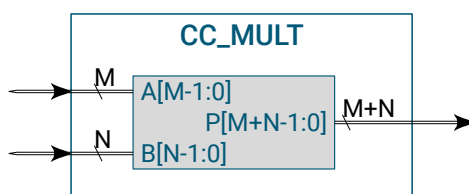


Figure 5.2: CC_MULT primitive schematic

Multipliers are automatically inferred by the synthesis tool. The Verilog examples on page 72 can be used as templates for plain or pipelined multiplier inference.

Port Description

Table 5.2: CC_MULT port description

Port	Direction	Width	Description
A	Input	M	Factor A [M-1:0]
B	Input	N	Factor B [N-1:0]
P	Output	M+N	Product $A*B = P$ [M+N-1:0]

Parameter Description

Table 5.3: CC_MULT parameter description

Parameter	Type	Default	Description
A_WIDTH	INT	2	Factor A input bit width ¹
B_WIDTH	INT	2	Factor B input bit width ¹
P_WIDTH	INT	4	Product P output bit width ¹

¹ The maximum width for M and N is currently limited to 100.

Implementation Notes

Bus widths for inputs A and B may be different¹. The CC_MULT primitive is a signed multiplier. To perform an unsigned multiplication, the most significant bit (MSB) must be set to zero.

```
// Signed 10x8 multiplier
CC_MULT #(
    .A_WIDTH(10), // Factor A bit width
    .B_WIDTH(8),  // Factor B bit width
    .P_WIDTH(18)  // Product P bit width
) mult_signed (
    .A(a[9:0]),           // 10-bit factor A
    .B({b[7], b[7], b[7:0]}), // 8-bit factor B with sign-extend
    .P(p[17:0])           // 10+8-bit product P
);

// Unsigned 10x8 multiplier
CC_MULT #(
    .A_WIDTH(11), // Factor A bit width
    .B_WIDTH(9),  // Factor B bit width
    .P_WIDTH(18)  // Product P bit width
) mult_unsigned (
    .A({1'b0, a[9:0]}), // 10-bit factor A with zero-extend
    .B({2'b0, b[7:0]}), // 8-bit factor B with zero-extend
    .P(p[17:0])         // 10+8-bit product P
);
```

Verilog Instantiation

```
CC_MULT #(
    .A_WIDTH(0), // Factor A bit width
    .B_WIDTH(0), // Factor B bit width
    .P_WIDTH(0)  // Product P bit width
) mult_inst (
    .A(A), // Factor A
    .B(B), // Factor B
    .P(P)  // Product P
);
```

¹ The shortest signal path is achieved if the width of A is greater than the width of B.

VHDL Instantiation

```
mult_inst: CC_MULT
generic map (
  A_WIDTH => 0, -- Factor A bit width
  B_WIDTH => 0, -- Factor B bit width
  P_WIDTH => 0  -- Product P bit width
)
port map (
  A => A, -- Factor A
  B => B, -- Factor B
  P => P  -- Product P
);
```

Verilog Inference Examples

```
input wire clk, rst;

input wire signed [M-1:0] a;
input wire signed [N-1:0] b;
output wire signed [M+N-1:0] p0;
output reg signed [M+N-1:0] p1;
output reg signed [M+N-1:0] p2;

reg signed [M-1:0] a_reg;
reg signed [N-1:0] b_reg;

// Plain M*N multiplier without pipeline registers
always @(*)
begin
    p0 <= a * b;
end

// M*N multiplier with with output pipeline registers
always @(posedge clk)
begin
    p1 <= a * b;
end

// M*N multiplier with
// - input and output pipeline registers
// - asynchronous reset (active high)
always @(posedge clk or posedge rst)
begin
    if (rst) begin
        a_reg <= 0;
        b_reg <= 0;
        p2 <= 0;
    end
    else begin
        a_reg <= a;
        b_reg <= b;
        p2 <= a_reg * b_reg;
    end
end
```

VHDL Inference Examples

```
entity mult is
  generic (
    M : integer := 18;
    N : integer := 6
  );
  port (
    clk : in  std_logic;
    rst : in  std_logic;
    a   : in  signed(M-1 downto 0);
    b   : in  signed(N-1 downto 0);
    p0  : out signed(M+N-1 downto 0);
    p1  : out signed(M+N-1 downto 0);
    p2  : out signed(M+N-1 downto 0)
  );
end entity;

architecture rtl of mult is
  signal a_i : signed(M-1 downto 0);
  signal b_i : signed(N-1 downto 0);
begin

  -- Plain M*N multiplier without pipeline registers
  p0 <= a * b;

  -- M*N multiplier with output pipeline registers
  proc0: process(clk, rst)
  begin
    if rising_edge(clk) then
      p1 <= a * b;
    end if;
  end process proc0;

  -- M*N multiplier with
  -- - input and output pipeline registers
  -- - asynchronous reset (active high)
  proc1: process(clk, rst)
  begin
    if rst = '0' then
      a_i <= (others => '0');
      b_i <= (others => '0');
      p2 <= (others => '0');
    elsif rising_edge(clk) then
      a_i <= a;
      b_i <= b;
      p2 <= a_i * b_i;
    end if;
  end process proc1;

end architecture;
```


Chapter 6

Block RAM

Content Overview

6.1	CC_BRAM_{20K,40K}	76
6.2	CC_FIFO_40K	94

6.1 CC_BRAM_{20K, 40K}

Each GateMate™ random-access memory (RAM) block can be configured as a single 40K or two independent 20K dual port SRAM (DPSRAM) cells. Figures 6.1 and 6.2 show the schematics for both configurations.

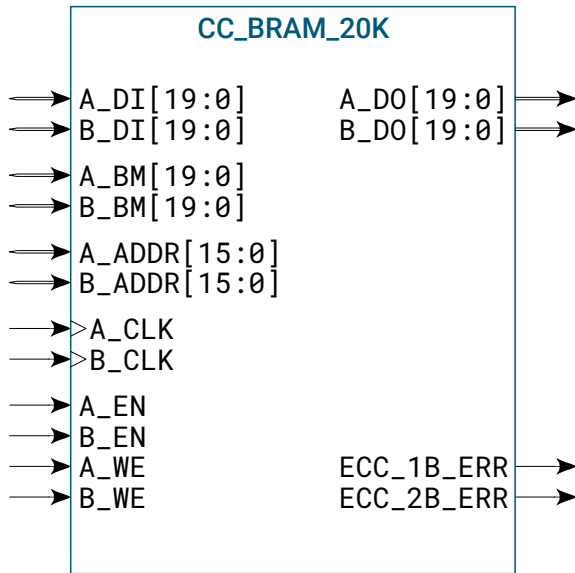


Figure 6.1: CC_BRAM_20K primitive schematic

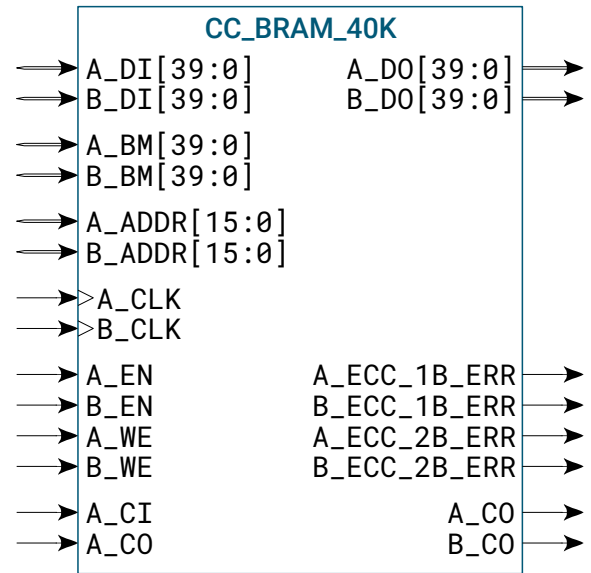


Figure 6.2: CC_BRAM_40K primitive schematic

Each RAM block allows usage of the memory in true dual port (TDP) or simple dual port (SDP) mode and offers the following features:

- Data widths from 1 bit up to 40 bits in TDP mode or 80 bits in SDP mode.
- Bit-wide write enable allows a bit-wise writing of incoming data and can be used when e.g. interfacing with a microprocessor.
- Each port has optional output registers for the {A|B}_DO and error checking and correcting (ECC) status signals. When enabled, validity of the output data gets delayed by one clock cycle.
- Each port provides an ECC module for data protection against unintended changes. The feature is available for data widths of 32 or 64 bits only while the remaining 8 or 16 bits are required for storage of parity bits. The error correction algorithm is able to correct one bit error and detect two bit errors. The error status can be queried via the corresponding output signals.
- A memory cascading feature connects adjacent RAM cells to form a larger memory.
- Contents of the RAM cells can be initialized during configuration, e.g. to store firmware data or to build read-only memories (ROMs).
- All clock, enable and write enable signals can be inverted individually.

Tables 6.1 and 6.2 show the available address and data bus width configurations as well as the ECC availability for both 20K and 40K configurations in SDP and TDP modes.

Table 6.1: 20K configurations

Configuration	TDP	SDP	ECC
(RAM size per 20K block)			
16K x 1 bit	✓	✓	✗
8K x 2 bit	✓	✓	✗
4K x 5 bit	✓	✓	✗
2K x 10 bit	✓	✓	✗
1K x 20 bit	✓	✓	✗
512 x 40 bit	✗	✓	✓

Table 6.2: 40K configurations

Configuration	TDP	SDP	ECC
32K x 1 bit	✓	✓	✗
16K x 2 bit	✓	✓	✗
8K x 5 bit	✓	✓	✗
4K x 10 bit	✓	✓	✗
2K x 20 bit	✓	✓	✗
1K x 40 bit	✓	✓	✓
512 x 80 bit	✗	✓	✓

Port Description

Table 6.3: CC_BRAM_{20K, 40K} port description

Port	Direction	Width		Description
		20K	40K	
{A B}_DI	Input	20	40	Port {A B} data input bus
{A B}_CI	Input	n/a	1	Port {A B} cascade input
{A B}_BM	Input	20	40	Port {A B} write enable bitmask
{A B}_ADDR	Input	16		Port {A B} address bus
{A B}_EN	Input	1		Port {A B} global enable
{A B}_WE	Input	1		Port {A B} global write enable
{A B}_CLK	Input	1		Port {A B} clocks
{A B}_D0	Output	20	40	Port {A B} data output bus
{A B}_C0	Input	n/a	1	Port {A B} cascade output
{A B}_ECC_1B_ERR	Output	1		Port {A B} 1-bit ECC output flags
{A B}_ECC_2B_ERR	Output	1		Port {A B} 2-bit ECC output flags

Parameter Description

Table 6.4: *CC_BRAM_{20K, 40K} parameter description*

Parameter	Type	Default	Description
LOC	STRING	"UNPLACED"	Location in FPGA array: $D(0..N-1)X(0..3)Y(0..7)$ D: FPGA-die number X: x-coordinate (0..3) Y: y-coordinate (0..7)
CAS	STRING	"NONE"	Cascade setting, 40K only: "NONE": no cascade (default) "UPPER": cell is upper half "LOWER": cell is lower half
{A B}_RD_WIDTH	INT	0	Valid data output {A B} widths: 0 (default), 1, 2, 5, 10, 20, 40 and 80 (see Tables 6.1 and 6.2)
{A B}_WR_WIDTH	INT	0	Valid data input {A B} widths: 0 (default), 1, 2, 5, 10, 20, 40 and 80 (see Tables 6.1 and 6.2)
RAM_MODE	STRING	"SDP"	RAM dual-port mode: "SDP": simple dual port (default) "TDP": true dual port
{A B}_WR_MODE	STRING	"NO_CHANGE"	Write {A B} / modes: "NO_CHANGE" (default) "WRITE_THROUGH"
{A B}_CLK_INV	BOOL	0	{A B}_CLK inverting: 0: disable (default) 1: enable
{A B}_EN_INV	BOOL	0	{A B}_EN inverting: 0: disable (default) 1: enable
{A B}_WE_INV	BOOL	0	{A B}_WE inverting: 0: disable (default) 1: enable

Continued on next page

Table 6.3: *CC_BRAM_{20K, 40K} parameter description*

Continued from previous page

Parameter	Type	Default	Description
{A B}_DO_REG	BOOL	0	Output {A B} registers enable: 0: disable (default) 1: enable
{A B}_ECC_EN	INT[1:0]	0	Port {A B} 1-bit ECC enable: 0: disable (default) 1: enable
INIT_XX	INT[319:0]	0	RAM initialization parameters (see Section 'Content Initialization' on page 83)

SDP Mode

Block RAM in SDP mode supports simultaneous read and write operations, but has a single output port for read data. By that, data widths can be increased up to 80 bits.

Port A is always the write port, while port B is always the read port. Both write and read ports can have separate clocks. The write port uses the enable (A_EN) and write enable (A_BM or A_WE) signals for write access and the read port uses only the enable signal (B_EN) for read access.

Tables 6.4 and 6.5 show the data and address bus wiring in SDP 20K and 40K mode.

Table 6.4: *Pin wiring in SDP 20K mode*

Width	Depth	Address-In Bus	Data-In Bus ¹	Data-Out Bus ¹
40	512	{A B}_ADDR[15:7]	B_DI[19:0] ◦ A_DI[19:0]	B_DO[19:0] ◦ A_DO[19:0]

¹ Symbol ◦ is concatenation.

Table 6.5: *Pin wiring in SDP 40K mode*

Width	Depth	Address-In Bus	Data-In Bus ¹	Data-Out Bus ¹
80	512	{A B}_ADDR[15:7]	B_DI[39:0] ◦ A_DI[39:0]	B_DO[39:0] ◦ A_DO[39:0]

¹ Symbol ◦ is concatenation.

TDP Mode

Block RAM in TDP mode supports simultaneous read and write operations and has two independent access ports. All combinations of operations at these two ports are possible: two reads, two writes or one read and one write. Both ports can have separate clocks. Thus, the clocks can be synchronous or asynchronous. Both ports have access to the entire memory at any time. There is no internal conflict handling when accessing the same address at the same time. Input and output data widths may be different.

Tables 6.6 and 6.7 show the data and address bus wiring in TDP 20K and 40K mode.

Table 6.6: Pin wiring in TDP 20K mode

Width	Depth	Address-In Bus ¹	Data-In Bus	Data-Out Bus
1	16,384	{A B}_ADDR[15:7] ◦ {A B}_ADDR[5:1]	{A B}_DI[0]	{A B}_DO[0]
2	8,192	{A B}_ADDR[15:7] ◦ {A B}_ADDR[5:2]	{A B}_DI[1:0]	{A B}_DO[1:0]
5	4,096	{A B}_ADDR[15:7] ◦ {A B}_ADDR[5:3]	{A B}_DI[4:0]	{A B}_DO[4:0]
10	2,048	{A B}_ADDR[15:7] ◦ {A B}_ADDR[5:4]	{A B}_DI[9:0]	{A B}_DO[9:0]
20	1,024	{A B}_ADDR[15:7] ◦ {A B}_ADDR[5]	{A B}_DI[19:0]	{A B}_DO[19:0]

¹ Symbol ◦ is concatenation.

Table 6.7: Pin wiring in TDP 40K mode

Width	Depth	Address-In Bus	Data-In Bus	Data-Out Bus
1	32,768	{A B}_ADDR[15:1]	{A B}_DI[0]	{A B}_DO[0]
2	16,384	{A B}_ADDR[15:2]	{A B}_DI[1:0]	{A B}_DO[1:0]
5	8,192	{A B}_ADDR[15:3]	{A B}_DI[4:0]	{A B}_DO[4:0]
10	4,096	{A B}_ADDR[15:4]	{A B}_DI[9:0]	{A B}_DO[9:0]
20	2,048	{A B}_ADDR[15:5]	{A B}_DI[19:0]	{A B}_DO[19:0]
40	1,024	{A B}_ADDR[15:6]	{A B}_DI[39:0]	{A B}_DO[39:0]

Access Modes and Enable

A block RAM cell has three signals to control read and write access. {A|B}_EN is a global enable signal. It is required to be active during any read or write access. {A|B}_WE is a global write enable signal that operates in association with the bit-wise write enable vector {A|B}_BM.

Data is only written to the RAM if {A|B}_EN, {A|B}_WE and the corresponding bitmask {A|B}_BM are active. Reading of data depends on the selected access mode and the values of {A|B}_WE and {A|B}_BM. The address modes are listed in Table 6.8 and 6.9. They show

all combinations of $\{A|B\}_{EN}$, $\{A|B\}_{WE}$ and $\{A|B\}_{BM}$ in both modes NO CHANGE and WRITE THROUGH as well as the resulting actions.

Table 6.8: Access combinations in NO CHANGE mode with $\{A|B\}_{EN} = 1$ (SDP and TDP)

$\{A B\}_{WE}^1$	$\{A B\}_{BM}[i]$	Action on memory and $\{A B\}_{DO}$
0	0	Single read, no update on $mem[addr][i]$
1	0	Last read, no update on $mem[addr][i]$
0	1	Single read, no update on $mem[addr][i]$
1	1	Last read, with update on $mem[addr][i]$

¹ Only A_WE in SDP mode.

Table 6.9: Access combinations in WRITE THROUGH mode with $\{A|B\}_{EN} = 1$ (TDP only)

$\{A B\}_{WE}$	$\{A B\}_{BM}[i]$	Action on memory and $\{A B\}_{DO}$
0	0	Single read, no update on $mem[addr][i]$
1	0	Single read, no update on $mem[addr][i]$
0	1	Single read, no update on $mem[addr][i]$
1	1	Write through, with update on $mem[addr][i]$

A **single read** access without any write operation occurs always if the global enable is active ($\{A|B\}_{EN} = 1$) and all write enable signals are inactive ($\{A|B\}_{BM} = 0$, $\{A|B\}_{WE} = 0$). A single read access is supported in both modes SDP and TDP.



Figure 6.3: Timing diagram of a single read access with optional (*) output register

All RAM cells can be initialized during configuration and used as ROM. To use the **ROM mode** the write enable has to be set to zero. The memory mapping for initialization is described on page 83.

The NO CHANGE mode is a plain write access with active global enable ($\{A|B\}_{EN} = 1$) and write enable signals ($\{A|B\}_{BM} \geq 1$ or $\{A|B\}_{WE} = 1$). The output remains the last read data, i.e. after a single read, and is not affected by a write access. A NO CHANGE access is supported in both modes SDP and TDP.

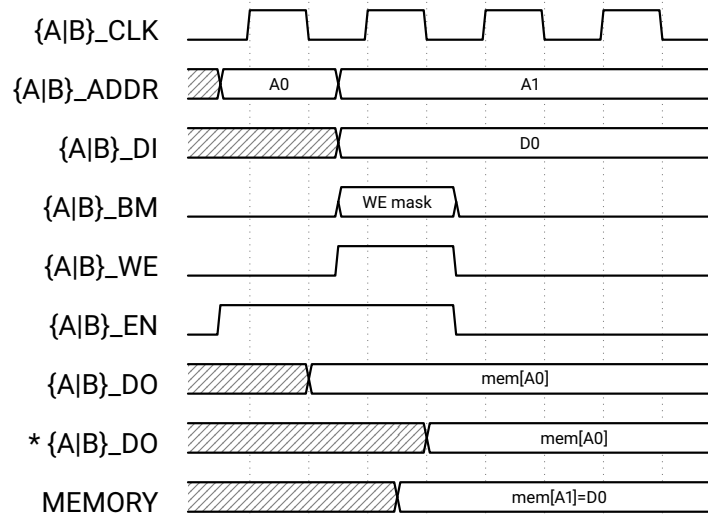


Figure 6.4: Timing diagram of a NO CHANGE access with optional (*) output register

A **read first** access can be carried out in two clock cycles by first reading a word and then performing a write access in the following cycle.

A WRITE THROUGH access is a simultaneous write and read access with active global enable ($\{A|B\}_{EN} = 1$) and write enable signals ($\{A|B\}_{BM} \geq 1$ or $\{A|B\}_{WE} = 1$). Data is written into the memory and simultaneously propagated to the outputs. WRITE THROUGH is only supported in TDP mode.

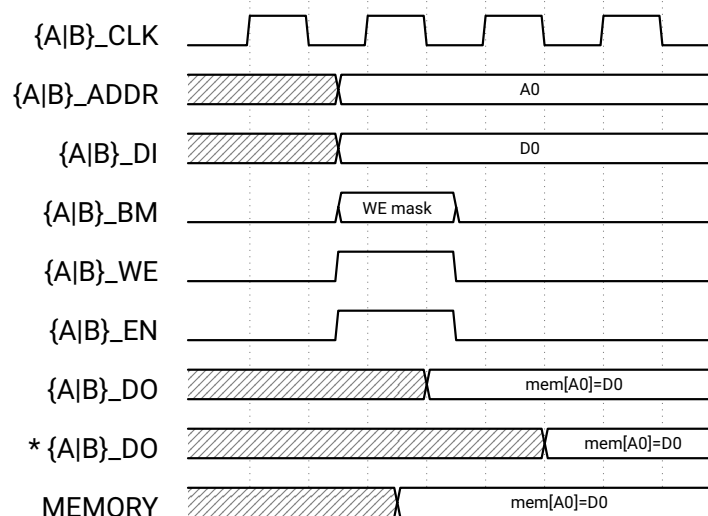


Figure 6.5: Timing diagram of a WRITE THROUGH access with optional (*) output register

Memory Mapping

The addressing scheme depends on the configuration mode. Words of 5, 10, 20, 40 and 80¹ bits are distributed equally over all RAM cells. However, every 5th bit is not accessible in 1 and 2 bit wide word configuration. This leads to a logical mapping of the memory bits as shown in Figure 6.6. The scheme shown here can be applied to both SDP and TDP modes in 20K as well as in 40K configurations.

Content Initialization

The memory contents can be initialized during configuration and is represented by the primitive's initialization attributes. After synthesis, the primitives memory contents can be written into the configuration bitstream during implementation.

The INIT_XX in Table 6.4 attributes define the initial memory contents, where each INIT_XX parameter is a 320 bit vector. For CC_BRAM_20K, there are 64 initialization attributes from INIT_00 to INIT_3F representing all 20K bits. Accordingly, the CC_BRAM_40K has 128 initialization attributes from INIT_00 to INIT_7F.

¹ Data width of 80 bits only supported in 40K SDP mode

Port Width	Logical Mapping																																							
1 Bit	x	63	62	61	60	x	59	58	57	56	x	55	54	53	52	x	51	50	49	48	x	47	46	45	44	x	43	42	41	40	x	39	38	37	36	x	35	34	33	32
2 Bits	x	31	30		x	29	28		x	27	26		x	25	24		x	23	22		x	21	20		x	19	18		x	17	16									
5 Bits	15				14				13				12				11				10				9				8											
10 Bits	7								6								5								4															
20 Bits	3																2																							
40 Bits	1																																							
80 Bits	0																																							

Bit Location	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Dual Port SRAM #1 – 512 x 40 bit																																							

1 Bit	x	31	30	29	28	x	27	26	25	24	x	23	22	21	20	x	19	18	17	16	x	15	14	13	12	x	11	10	9	8	x	7	6	5	4	x	3	2	1	0
2 Bits	x	15	14		x	13	12		x	11	10		x	9	8		x	7	6		x	5	4		x	3	2		x	1										
5 Bits	7				6				5				4				3				2				1				0											
10 Bits	3								2								1								0															
20 Bits	1																0																							
40 Bits	0																																							
80 Bits	0																																							

Bit Location	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Dual Port SRAM #0 – 512 x 40 bit																																							

x Bit not accessible

Figure 6.6: Logical data mapping of memory at physical address 0

Table 6.10: Block RAM initialization parameters

Primitive	Start	End	Parameters	Total Bits
CC_BRAM_20K	INIT_00	INIT_3F	64	20,480
CC_BRAM_40K	INIT_00	INIT_7F	128	40,960

By default, block RAM initialization parameters are initialized with zeros. If no parameters are given, RAM content will not be initialized during configuration and may be in an unpredictable state.

Verilog Inference Examples

```
// Single Port RAM (NO_CHANGE)
module bram_sp_no_change #(
    parameter DATA_WIDTH=32,
    parameter ADDR_WIDTH=9
)(
    input wire we,
    input wire clk,
    input wire [DATA_WIDTH-1:0] di,
    input wire [ADDR_WIDTH-1:0] addr,
    output reg [DATA_WIDTH-1:0] do
);

    localparam WORD = (DATA_WIDTH-1);
    localparam DEPTH = (2**ADDR_WIDTH-1);

    reg [WORD:0] memory [0:DEPTH];

    initial $readmemb("mem/mem_a9d32.hex", memory); // optional

    always @(posedge clk) begin
        if (we)
            memory[addr] <= di;
        else do <= memory[addr];
    end
endmodule
```

```
// Single Port RAM (WRITE_THROUGH)
module bram_sp_write_through #(
    parameter DATA_WIDTH=32,
    parameter ADDR_WIDTH=9
)(
    input wire we,
    input wire clk,
    input wire [DATA_WIDTH-1:0] di,
    input wire [ADDR_WIDTH-1:0] addr,
    output reg [DATA_WIDTH-1:0] do
);

    localparam WORD = (DATA_WIDTH-1);
    localparam DEPTH = (2**ADDR_WIDTH-1);

    reg [WORD:0] memory [0:DEPTH];

    initial $readmemb("mem/mem_a9d32.hex", memory); // optional

    always @(posedge clk) begin
        if (we) begin
            memory[addr] <= di;
            do <= di;
        end else
            do <= memory[addr];
        end
    end

endmodule
```

```
// Dual Port RAM (NO_CHANGE)
module bram_dp_no_change #(
    parameter DATA_WIDTH=18,
    parameter ADDR_WIDTH=9
)(
    input  wire          wea,
    input  wire          web,
    input  wire          clka,
    input  wire          clkb,
    input  wire [DATA_WIDTH-1:0] dia,
    input  wire [DATA_WIDTH-1:0] dib,
    input  wire [ADDR_WIDTH-1:0] addra,
    input  wire [ADDR_WIDTH-1:0] addrb,
    output reg [DATA_WIDTH-1:0] doa,
    output reg [DATA_WIDTH-1:0] dob
);

    localparam WORD  = (DATA_WIDTH-1);
    localparam DEPTH = (2**ADDR_WIDTH-1);

    reg [WORD:0] memory [0:DEPTH];

    initial $readmemb("mem/mem_a9d18.hex", memory); // optional

    always @(posedge clka) begin
        if (wea) begin
            memory[addra] <= dia;
        end else
            doa <= memory[addra];
        end

    always @(posedge clkb) begin
        if (web) begin
            memory[addrb] <= dib;
        end else
            dob <= memory[addrb];
        end

endmodule
```

```
// Dual Port RAM (WRITE_THROUGH)
module bram_dp_write_through #(
    parameter DATA_WIDTH=18,
    parameter ADDR_WIDTH=9
)(
    input  wire          wea,
    input  wire          web,
    input  wire          clka,
    input  wire          clkb,
    input  wire [DATA_WIDTH-1:0] dia,
    input  wire [DATA_WIDTH-1:0] dib,
    input  wire [ADDR_WIDTH-1:0] addra,
    input  wire [ADDR_WIDTH-1:0] addrb,
    output reg [DATA_WIDTH-1:0] doa,
    output reg [DATA_WIDTH-1:0] dob
);

    localparam WORD  = (DATA_WIDTH-1);
    localparam DEPTH = (2**ADDR_WIDTH-1);

    reg [WORD:0] memory [0:DEPTH];

    initial $readmemb("mem/mem_a9d18.hex", memory); // optional

    always @(posedge clka) begin
        if (wea) begin
            memory[addra] <= dia;
            doa <= dia;
        end else
            doa <= memory[addra];
        end

    always @(posedge clkb) begin
        if (web) begin
            memory[addrb] <= dib;
            dob <= dib;
        end else
            dob <= memory[addrb];
        end

endmodule
```

VHDL Inference Examples

```
library ieee;
USE ieee.std_logic_1164.ALL;
use ieee.numeric_std.all;

-- Single Port RAM (NO_CHANGE)
entity bram_sp_no_change is
  generic (
    DATA_WIDTH : integer := 32;
    ADDR_WIDTH  : integer := 9
  );
  port (
    we : in  std_logic;
    clk : in  std_logic;
    di : in  std_logic_vector(DATA_WIDTH-1 downto 0);
    addr : in  std_logic_vector(ADDR_WIDTH-1 downto 0);
    do : out std_logic_vector(DATA_WIDTH-1 downto 0)
  );
end entity;

architecture rtl of bram_sp_no_change is
  type ram is array (0 to (2**ADDR_WIDTH)-1) of std_logic_vector(
    ↪ DATA_WIDTH-1 downto 0);
  signal memory : ram;
begin

  process(clk)
  begin
    if rising_edge(clk) then
      if (we = '1') then
        memory(to_integer(unsigned(addr))) <= di;
      else
        do <= memory(to_integer(unsigned(addr)));
      end if;
    end if;
  end process;
end architecture;
```

```

library ieee;
USE ieee.std_logic_1164.ALL;
use ieee.numeric_std.all;

-- Single Port RAM (WRITE_THROUGH)
entity bram_sp_write_through is
  generic (
    DATA_WIDTH : integer := 32;
    ADDR_WIDTH  : integer := 9
  );
  port (
    we  : in  std_logic;
    clk : in  std_logic;
    di  : in  std_logic_vector(DATA_WIDTH-1 downto 0);
    addr : in  std_logic_vector(ADDR_WIDTH-1 downto 0);
    do  : out std_logic_vector(DATA_WIDTH-1 downto 0)
  );
end entity;

architecture rtl of bram_sp_write_through is
  type ram is array (0 to (2**ADDR_WIDTH)-1) of std_logic_vector(
    ↪ DATA_WIDTH-1 downto 0);
  signal memory : ram;
begin

  process(clk)
  begin
    if rising_edge(clk) then
      if (we = '1') then
        memory(to_integer(unsigned(addr))) <= di;
        do <= di;
      else
        do <= memory(to_integer(unsigned(addr)));
      end if;
    end if;
  end process;

end architecture;

```

```

library ieee;
USE ieee.std_logic_1164.ALL;
use ieee.numeric_std.all;

-- Dual Port RAM (NO_CHANGE)
entity bram_dp_no_change is
  generic (
    DATA_WIDTH : integer := 18;
    ADDR_WIDTH  : integer := 6
  );
  port (
    wea  : in  std_logic;
    web  : in  std_logic;
    clka : in  std_logic;
    clkb : in  std_logic;
    dia  : in  std_logic_vector(DATA_WIDTH-1 downto 0);
    dib  : in  std_logic_vector(DATA_WIDTH-1 downto 0);
    addra : in  std_logic_vector(ADDR_WIDTH-1 downto 0);
    addrb : in  std_logic_vector(ADDR_WIDTH-1 downto 0);
    doa  : out std_logic_vector(DATA_WIDTH-1 downto 0);
    dob  : out std_logic_vector(DATA_WIDTH-1 downto 0)
  );
end entity;

architecture rtl of bram_dp_no_change is
  type ram is array (0 to (2**ADDR_WIDTH)-1) of std_logic_vector(
    ↪ DATA_WIDTH-1 downto 0);
  shared variable memory : ram;
begin

  port_a: process(clka)
  begin
    if rising_edge(clka) then
      if (wea = '1') then
        memory(to_integer(unsigned(addra))) := dia;
      else
        doa <= memory(to_integer(unsigned(addra)));
      end if;
    end if;
  end process port_a;

  port_b: process(clkb)
  begin
    if rising_edge(clkb) then
      if (web = '1') then
        memory(to_integer(unsigned(addrb))) := dib;
      else
        dob <= memory(to_integer(unsigned(addrb)));
      end if;
    end if;
  end process port_b;

end architecture;

```

```

library ieee;
USE ieee.std_logic_1164.ALL;
use ieee.numeric_std.all;

-- Dual Port RAM (WRITE_THROUGH)
entity bram_dp_write_through is
  generic (
    DATA_WIDTH : integer := 18;
    ADDR_WIDTH  : integer := 6
  );
  port (
    wea  : in  std_logic;
    web  : in  std_logic;
    clka : in  std_logic;
    clkb : in  std_logic;
    dia  : in  std_logic_vector(DATA_WIDTH-1 downto 0);
    dib  : in  std_logic_vector(DATA_WIDTH-1 downto 0);
    addra : in  std_logic_vector(ADDR_WIDTH-1 downto 0);
    addrb : in  std_logic_vector(ADDR_WIDTH-1 downto 0);
    doa  : out std_logic_vector(DATA_WIDTH-1 downto 0);
    dob  : out std_logic_vector(DATA_WIDTH-1 downto 0)
  );
end entity;

architecture rtl of bram_dp_write_through is
  type ram is array (0 to (2**ADDR_WIDTH)-1) of std_logic_vector(
    ↪ DATA_WIDTH-1 downto 0);
  shared variable memory : ram;
begin

  port_a: process(clka)
  begin
    if rising_edge(clka) then
      if (wea = '1') then
        memory(to_integer(unsigned(addra))) := dia;
        doa <= dia;
      else
        doa <= memory(to_integer(unsigned(addra)));
      end if;
    end if;
  end process port_a;

  port_b: process(clkb)
  begin
    if rising_edge(clkb) then
      if (web = '1') then
        memory(to_integer(unsigned(addrb))) := dib;
        dob <= dib;
      else
        dob <= memory(to_integer(unsigned(addrb)));
      end if;
    end if;
  end process port_b;
end architecture;

```



```
end process port_b;  
end architecture;
```

6.2 CC_FIFO_40K

Each GateMate™ block RAM cell has an integrated synchronous and asynchronous first-in, first-out memory (FIFO) controller, allowing usage of a block RAM cell as FIFO memory in the 40K configurations only.

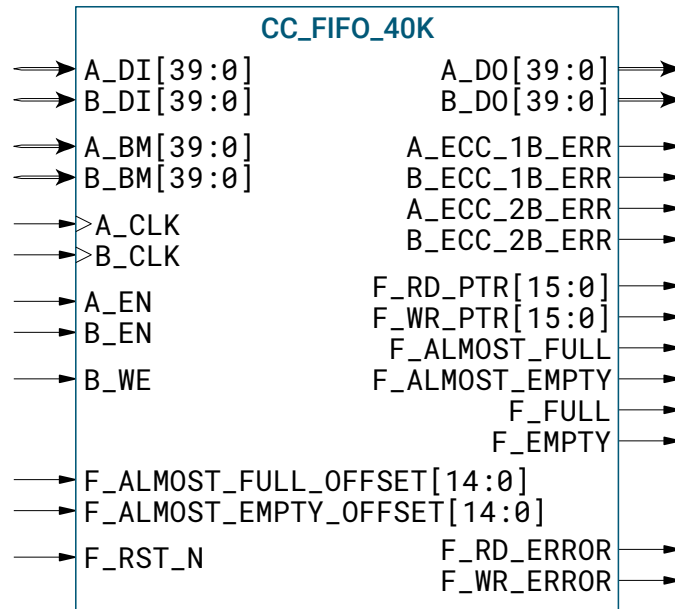


Figure 6.7: *CC_FIFO primitive schematic*

The port B is the write / push port of the FIFO and port A is the read / pop port. In case of synchronous FIFO, A_CLK is used as clock for both write / push and read / pop.

Since the FIFO mode is an extension to the TDP / SDP 40K mode, it supports the same bitwidth configurations as shown in Table 6.2. Widths of the input and output buses must be equal.

- TDP 40K, with arbitrary but equal input and output bit width
- SDP 40K, with fixed 80 bit input and output bit width

The CC_FIFO_40K has additional outputs which are solely used for FIFO monitoring and described in Table 6.11. The F_ALMOST_FULL and F_ALMOST_EMPTY status flags give an early warning when the FIFO is approaching its limits. Its offset can be configured using 15 bit registers during configuration or set dynamically using the inputs F_ALMOST_FULL_OFFSET and F_ALMOST_EMPTY_OFFSET.

Please note, that the utilization of dynamic offset inputs is exclusively accessible within the TDP configuration. In SDP mode, the static offset configuration remains the sole viable option.

Moreover, the FIFO controller has a dedicated active low reset input signal F_RST_N which is synchronized into clock domain internally.

Table 6.11: *FIFO status flags*

Flag	Width	Description
F_FULL	1	All entries in the FIFO are filled, is set on the rising edge of the write clock (asynchronous)
F_EMPTY	1	The FIFO is empty, is set on the rising edge of the read clock (asynchronous)
F_ALMOST_FULL	1	Almost all entries in the FIFO are filled, is set on the rising edge of the write clock (asynchronous)
F_ALMOST_EMPTY	1	Almost all entries in FIFO have been read, is set on the rising edge of the read clock (asynchronous)
F_RD_PTR	16	Current FIFO read pointer
F_WR_PTR	16	Current FIFO write pointer
F_RD_ERR	1	Is set if FIFO is empty and a read access takes place
F_WR_ERR	1	Is set if FIFO is full and data is pushed, new data will be lost

Table 6.12 illustrates the valid FIFO data concatenations for the variable bitwidth data input TDP mode. B_EN and B_WE are the write / push enable and A_EN is the read / pop enable signal.

Table 6.12: *FIFO 40 bit data input concatenations*

Function	Width	Concatenation
Push data	40	B_DI[39:0]
Push bitmask	40	B_BM[39:0]
Pop data	40	A_D0[39:0]

Table 6.13 illustrates the valid FIFO data concatenations for the 80 bit data input SDP mode. B_EN and B_WE are the write / push enable and A_EN is the read / pop enable signal.

Table 6.13: *FIFO 80 bit data input concatenations*

Function	Width	Concatenation
Push data	80	B_DI[39:0] ◦ A_DI[39:0]
Push bitmask	80	B_BM[39:0] ◦ A_BM[39:0]
Pop data	80	B_D0[39:0] ◦ A_D0[39:0]

¹ Symbol ◦ is concatenation.

Port Description

Table 6.14: *CC_FIFO_40K port description*

Port	Direction	Width	Description
{A B}_DI	Input	40	Port {A B} data input bus
{A B}_BM	Input	40	Port {A B} write enable bitmask
{A B}_CLK	Input	1	Port {A B} clocks
{A B}_EN	Input	1	Port {A B} global enable
B_WE	Input	1	Port B global write enable
F_ALMOST_FULL_OFFSET	Input	15	Offset value for F_ALMOST_FULL status flag, only available in TDP configuration
F_ALMOST_EMPTY_OFFSET	Input	15	Offset value for F_EMPTY_FULL status flag, only available in TDP configuration
F_RST_N	Input	1	asynchronous FIFO reset, active low
{A B}_DO	Output	40	Port {A B} data output bus
{A B}_ECC_1B_ERR	Output	1	Port {A B} 1-bit ECC output flags
{A B}_ECC_2B_ERR	Output	1	Port {A B} 2-bit ECC output flags
F_RD_PTR	Output	16	FIFO read pointer
F_WR_PTR	Output	16	FIFO write pointer
F_ALMOST_FULL	Output	1	FIFO almost full status flag
F_ALMOST_EMPTY	Output	1	FIFO almost empty status flag
F_FULL	Output	1	FIFO full status flag
F_EMPTY	Output	1	FIFO empty status flag
F_RD_ERR	Output	1	FIFO read error flag, see Table 6.11
F_WR_ERR	Output	1	FIFO write error flag, see Table 6.11

Parameter Description

Table 6.15: *CC_FIFO_40K parameter description*

Parameter	Type	Default	Description
LOC	STRING	"UNPLACED"	Location in FPGA array: $D(0..N-1)X(0..3)Y(0..7)$ D: FPGA-die number X: x-coordinate (0..3) Y: y-coordinate (0..7)
ALMOST_FULL_OFFSET	INT[14:0]	0	Static early FULL limit warning
ALMOST_EMPTY_OFFSET	INT[14:0]	0	Static early EMPTY limit warning
DYN_STAT_SELECT	INT	0	Select between dynamic or static almost full or empty offset: 0: dynamic offset via port inputs (default) 1: static offset via parameters
{A B}_RD_WIDTH	INT	0	Valid data output {A B} widths: 0 (default), 1, 2, 5, 10, 20, 40 and 80 (see Tables 6.1 and 6.2)
{A B}_WR_WIDTH	INT	0	Valid data input {A B} widths: 0 (default), 1, 2, 5, 10, 20, 40 and 80 (see Tables 6.1 and 6.2)
RAM_MODE	STRING	"SDP"	RAM dual-port mode: "SDP": simple dual port (default) "TDP": true dual port
FIFO_MODE	STRING	"SYNC"	FIFO mode: "SYNC": synchronous push / pop (default) "ASYN": asynchronous push / pop
{A B}_CLK_INV	BOOL	0	{A B}_CLK inverting: 0: disable (default) 1: enable
{A B}_EN_INV	BOOL	0	{A B}_EN inverting: 0: disable (default) 1: enable

Continued on next page

Table 6.14: *CC_FIFO_40K parameter description*

Continued from previous page

Parameter	Type	Default	Description
{A B}_WE_INV	BOOL	0	{A B}_WE inverting: 0: disable (default) 1: enable
{A B}_DO_REG	BOOL	0	Output {A B} registers enable: 0: disable (default) 1: enable
{A B}_ECC_EN	INT[1:0]	0	Port {A B} 1-bit ECC enable: 0: disable (default) 1: enable

Synchronous FIFO Access

Write/push and read/pop pointers are both registered with the rising clock edge of A_CLK. During the write/push operation, the data word available at {A,B}_DI / {A,B}_BM is written into the FIFO whenever the B_EN and B_WE signals are active one setup time before the rising clock edge of A_CLK. The write / push operation presents the data word at {A0,A1}_D0 whenever the A_EN signal is active one setup time the rising clock edge of A_CLK.

The signals empty, full, almost empty, almost full, read and write error are combinatorial computed out of read and write pointer. The error flags are not sticky.

The timing diagram in Figure 6.8 illustrates the writing to an empty synchronous FIFO.

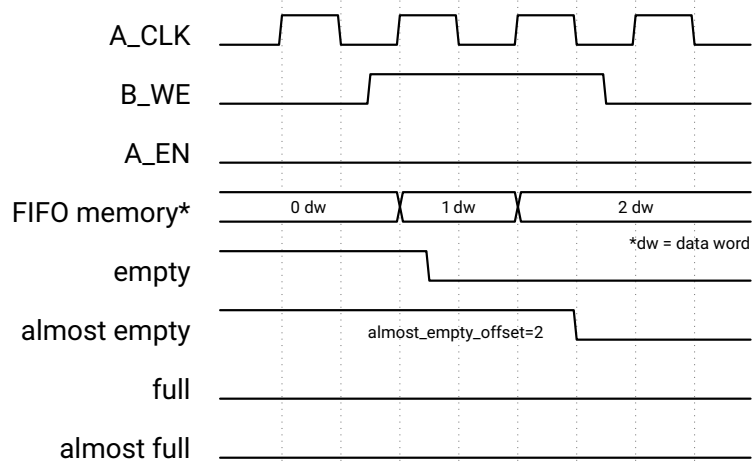


Figure 6.8: Writing to an empty synchronous FIFO

The timing diagram in Figure 6.9 illustrates the writing to an almost full synchronous FIFO.

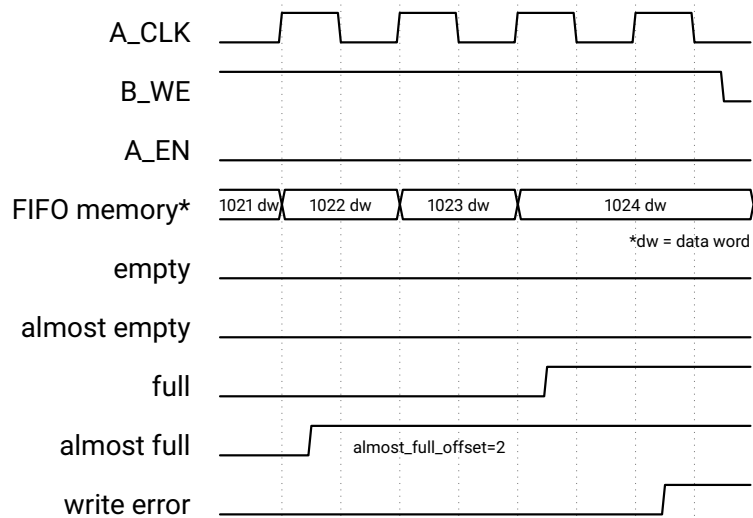


Figure 6.9: Writing to an almost full synchronous FIFO

The timing diagram in Figure 6.10 illustrates the reading from a full synchronous FIFO.

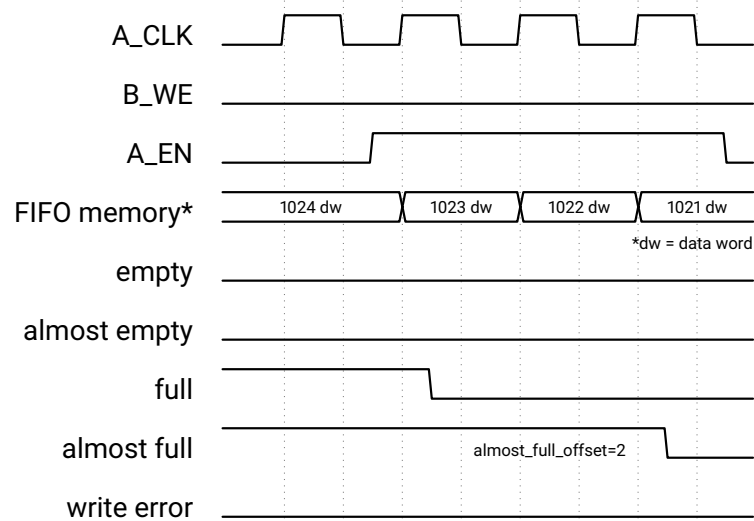


Figure 6.10: Reading from a full synchronous FIFO

The timing diagram in Figure 6.11 illustrates the reading from an almost empty synchronous FIFO.

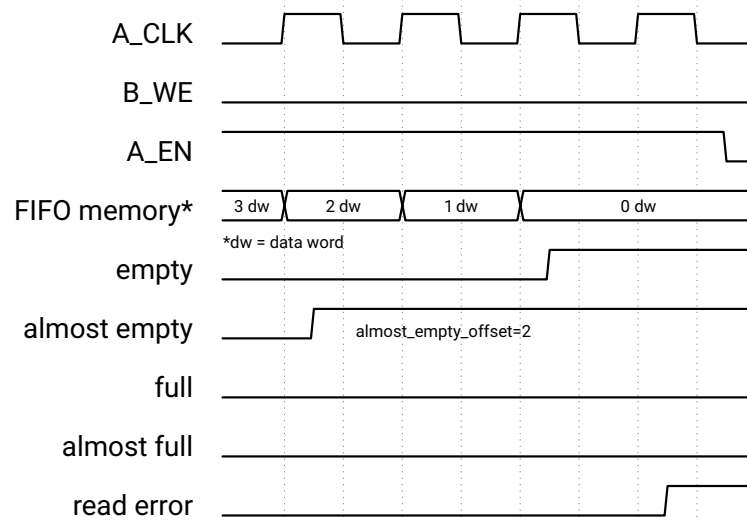


Figure 6.11: Reading from an almost empty synchronous FIFO

Asynchronous FIFO Access

During the write / push operation, the data word available at {A, B}_DI / {A, B}_BM is written into the FIFO whenever the B_EN and B_WE signals are active one setup time before the rising clock edge of B_CLK. The write / push operation presents the data word at {A, B}_DO whenever the A_EN signal is active one setup time the rising clock

edge of A_CLK. The read / pop pointer is registered with the read / pop clock A_CLK, the write / push pointer is registered with the write / push clock B_CLK.

For full, empty, almost full, almost empty, read and write error signal generation the read pointer is synchronized into the write clock domain via Gray encoding and 2-stage synchronization, the write pointer is synchronized into the read clock domain via Gray encoding and 2-stage synchronization.

The empty, almost empty and read error signals are combinatorial computed out of the read / pop pointer and the synchronized write / push pointer. By that, the empty, almost empty and read error signals will be always cycle accurate with the read clock without any delay and a read from an empty FIFO can always be prevented.

In the following figures, these signals are highlighted in red related to the read side of the FIFO since their computation bases on flip-flops driven by the read / pop clock. The full, almost full and write error signals are combinatorial computed out of the write / push pointer and the synchronized read / pop pointer. By that, the full, almost full and write error signals will be always cycle accurate with the write clock without any delay and a write to a full FIFO can always be prevented.

Moreover, the signals highlighted in green relate to the write side of the FIFO since their computation bases on flip-flops driven by the write / push clock.

The timing diagram in Figure 6.12 illustrates the writing to an empty asynchronous FIFO.

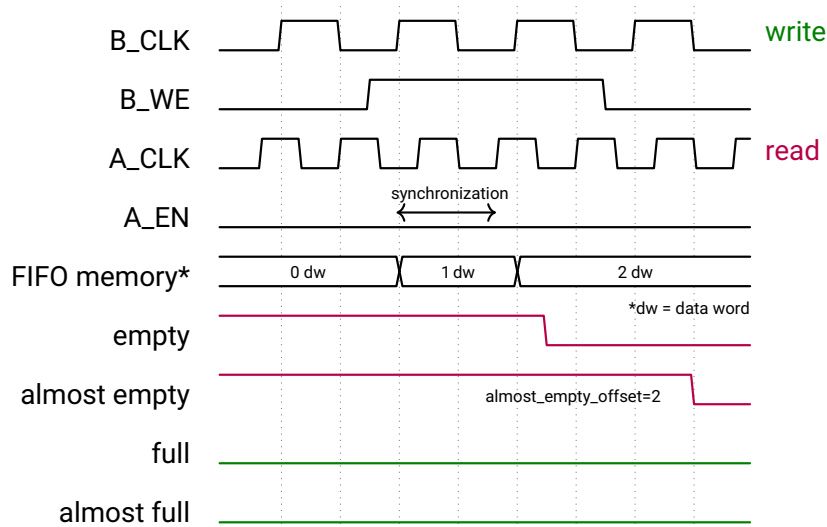


Figure 6.12: Writing to an empty asynchronous FIFO

The timing diagram in Figure 6.13 illustrates the writing to an almost full asynchronous FIFO.

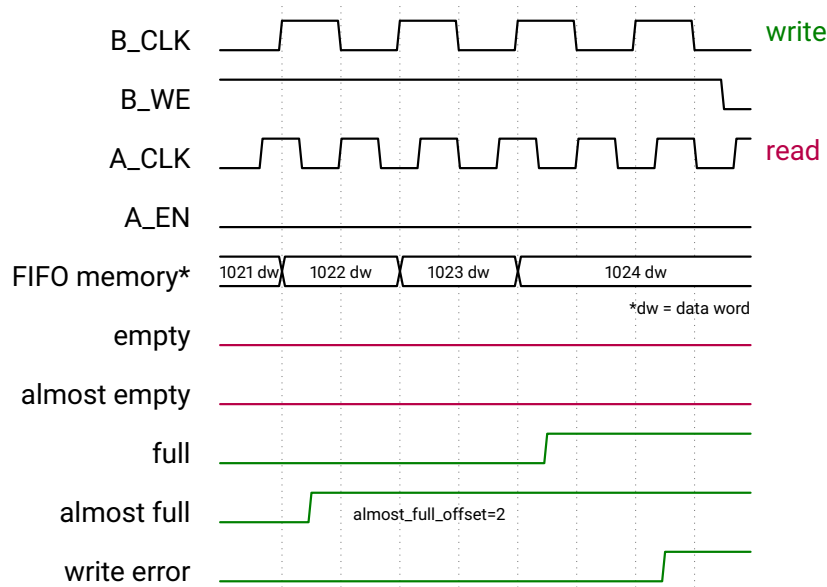


Figure 6.13: Writing to an almost full asynchronous FIFO

The timing diagram in Figure 6.14 illustrates the reading from a full asynchronous FIFO.

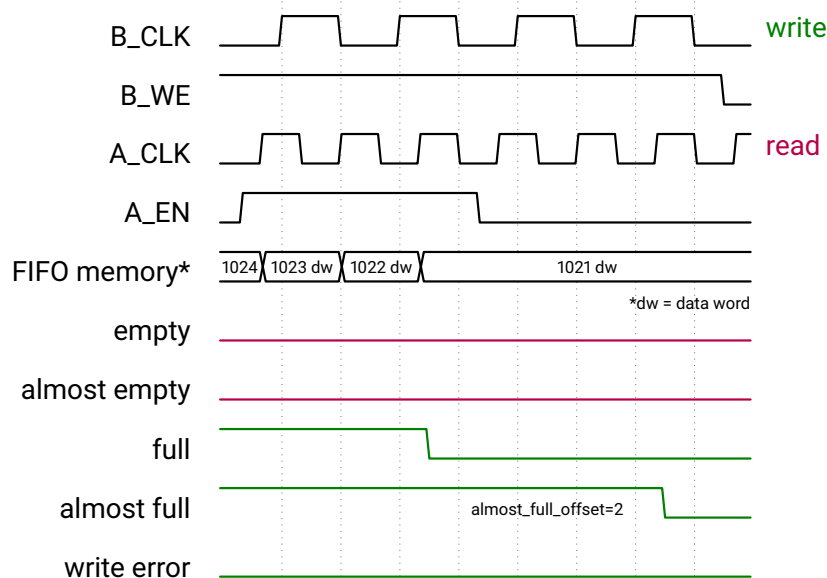


Figure 6.14: Reading from a full asynchronous FIFO

The timing diagram in Figure 6.15 illustrates the reading from a full asynchronous FIFO.

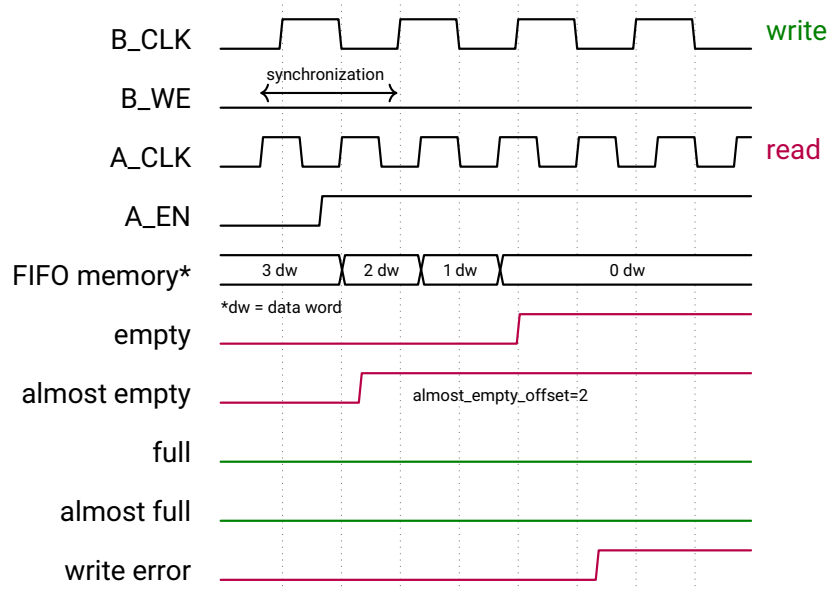


Figure 6.15: Reading from an almost empty asynchronous FIFO

Chapter 7

Special Function Blocks

Content Overview

7.1	CC_BUFG	106
7.2	CC_USR_RSTN	107
7.3	CC_PLL	109
7.4	CC_PLL_ADV	114
7.5	CC_SERDES	117
7.6	CC_CFG_CTRL	125

7.1 CC_BUFG

The CC_BUFG primitive is a buffer that connects the input signal to the global routing resources. Typically, it is automatically inferred by the synthesis tool to feed a clock signal into the clock net. The low skew distribution of the signal makes the primitive particularly efficient when dealing with high-fanout signals.

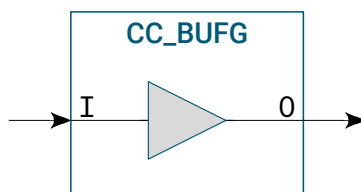


Figure 7.1: CC_BUFG primitive schematic

Port Description

Table 7.1: CC_BUFG port description

Port	Direction	Width	Description
I	Input	1	Input from user logic or input buffer
O	Output	1	Output to global routing resource

Verilog Instantiation

```
CC_BUFG bufg_inst (
    .I(I), // Input from CPE array or input buffer
    .O(O) // Output to global routing resource
);
```

VHDL Instantiation

```
bufg_inst: CC_BUFG
port map (
    I => I, -- Input from CPE array or input buffer
    O => O  -- Output to global routing resource
);
```

7.2 CC_USR_RSTN

The CC_USR_RSTN primitive is used to generate the signal USR_RSTN which shows the end of configuration. It can be used to generate an internal asynchronous set / reset or start signal. No set / reset signal needs to be fed-in from any GPIO.



Figure 7.2: CC_USR_RSTN primitive schematic

Port Description

Table 7.2: CC_USR_RSTN port description

Port	Direction	Width	Description
USR_RSTN	Output	1	Reset signal to the Cologne Programmable Element (CPE) array

Verilog Instantiation

```
CC_USR_RSTN usr_rstn_inst (  
    .USR_RSTN(USR_RSTN) // reset signal to CPE array  
);
```

VHDL Instantiation

```
usr_rstn_inst: CC_USR_RSTN  
port map (  
    USR_RSTN => USR_RSTN -- reset signal to CPE array  
);
```

Verilog Inference Examples

```
module usrrstn(  
    input  clk_i,  
    output rst_o  
);  
  
wire usr_rstn_w;  
reg  usr_rst_sync_r, rst_r;  
  
assign rst_o = rst_r;  
  
CC_USR_RSTN usr_rstn_inst (  
    .USR_RSTN(usr_rstn_w)  
);  
  
always @(posedge clk_i or negedge usr_rstn_w)  
begin  
    if (!usr_rstn_w) begin  
        usr_rst_sync_r <= 1'b0;  
        rst_r <= 1'b0;  
    end  
    else begin  
        usr_rst_sync_r <= 1'b1;  
        rst_r <= usr_rst_sync_r; // use 'rst_r' for sync. user reset  
    end  
end  
  
always @(posedge clk_i)  
if (rst_r) begin  
    // ...  
end  
else begin  
    // ...  
end  
endmodule
```


7.3 CC_PLL

The CC_PLL primitive is a blackbox element of the built-in clock generators and must be instantiated manually. It abstracts the hardware to an extent that it can be operated without in-depth knowledge of the phase-locked loop (PLL). Only few parameter settings are necessary to configure the PLL without the need of any additional tools.

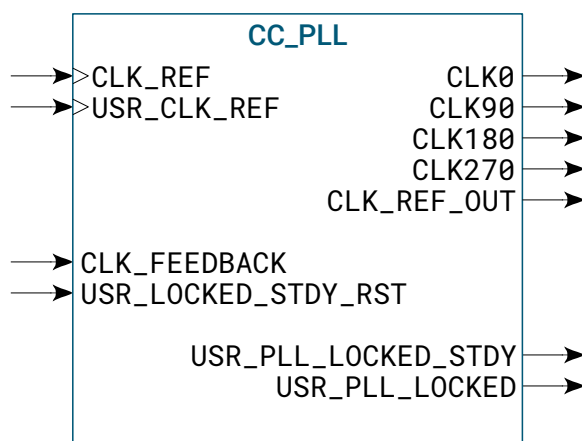


Figure 7.3: CC_PLL primitive schematic

When instantiating the CC_PLL primitive, either CLK_REF or USR_CLK_REF must be driven from an input buffer CC_IBUF to the primitive input port. CLK_REF can be connected to any signal listed in Table 7.4. USR_CLK_REF can be connected to any general purpose input/output (GPIO) input. This includes the dedicated clock inputs IO_SB_A5..IO_SB_A8 (CLK3..CLK0) as well.

Each clock output of the PLL that is connected to the FPGA-internal circuitry must also be fed into the clock network via a global buffer primitive CC_BUFG.

Figure 7.4 shows the CC_PLL output signals. The last two waves show that the ports CLK180 and CLK270 can be configured to double the output frequency.

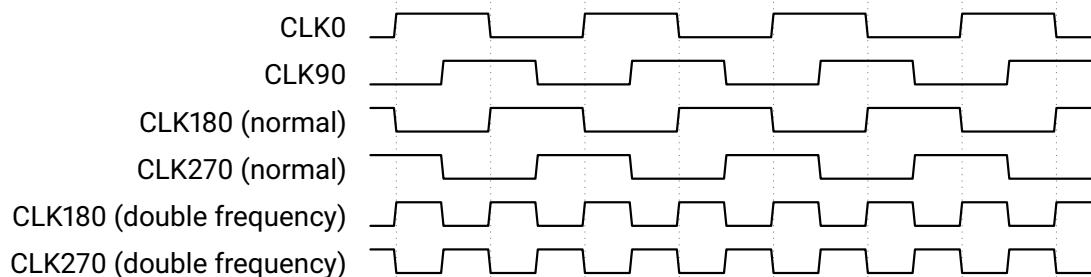


Figure 7.4: CC_PLL output signals including frequency doubling on ports CLK180 and CLK270

Verilog Instantiation

```
CC_PLL #(
    .REF_CLK(10.0),
    .OUT_CLK(50.0),
    .PERF_MD("ECONOMY"), // LOWPOWER, ECONOMY, SPEED (optional, global
        ↪ setting of Place&Route can be used instead)
    .LOW_JITTER(1),      // 0: disable, 1: enable low jitter mode
    .LOCK_REQ(1),
    .CLK270_DOUB(0),
    .CLK180_DOUB(0),
    .CI_FILTER_CONST(2), // optional CI filter constant
    .CP_FILTER_CONST(4)  // optional CP filter constant
) pll_inst (
    .CLK_REF(),
    .USR_CLK_REF(),
    .CLK_FEEDBACK(),
    .USR_LOCKED_STDY_RST(),
    .USR_PLL_LOCKED_STDY(),
    .USR_PLL_LOCKED(),
    .CLK0(),
    .CLK90(),
    .CLK180(),
    .CLK270(),
    .CLK_REF_OUT()
);
```

VHDL Instantiation

```
pll_inst: CC_PLL
generic map (
  REF_CLK => 10.0,      -- reference clk in MHz
  OUT_CLK => 50.0,      -- output clk in MHz
  PERF_MD => "ECONOMY", -- LOWPOWER, ECONOMY, SPEED (optional, global
    ↳ setting of Place&Route can be used instead)
  LOW_JITTER => 1,      -- 0: disable, 1: enable low jitter mode
  LOCK_REQ => 1,
  CLK270_DOUB => 0,
  CLK180_DOUB => 0,
  CI_FILTER_CONST => 2, -- optional CI filter constant
  CP_FILTER_CONST => 4  -- optional CP filter constant
)
port map (
  CLK_REF          => CLK_REF,
  USR_CLK_REF      => USR_CLK_REF,
  CLK_FEEDBACK     => CLK_FEEDBACK,
  USR_LOCKED_STDY_RST => USR_LOCKED_STDY_RST,
  USR_PLL_LOCKED_STDY => USR_PLL_LOCKED_STDY,
  USR_PLL_LOCKED   => USR_PLL_LOCKED,
  CLK0             => CLK0,
  CLK90            => CLK90,
  CLK180           => CLK180,
  CLK270           => CLK270,
  CLK_REF_OUT      => CLK_REF_OUT
);
```

Port Description

Table 7.3: *CC_PLL port description*

Port	Direction	Width	Description
CLK_REF	Input	1	Reference clock signal from dedicated clock pin (see Table 7.4)
USR_CLK_REF	Input	1	Alternative reference clock signal from FPGA-internal circuitry
CLK_FEEDBACK	Input	1	Feedback clock signal
USR_LOCKED_STDY_RST	Input	1	Reset of USR_PLL_LOCKED_STDY, must be set to 1 for a minimum 2 cycles of CLK_REF
USR_PLL_LOCKED_STDY	Output	1	PLL permanent lock status signal
USR_PLL_LOCKED	Output	1	PLL lock status signal
CLK0	Output	1	PLL clock output, no phase shift
CLK90	Output	1	PLL clock output, 90° phase shift
CLK180	Output	1	PLL clock output, 180° phase shift
CLK270	Output	1	PLL clock output, 270° phase shift
CLK_REF_OUT	Output	1	PLL reference clock output

Table 7.4: *Dedicated clock pins*

BGA ball	1st pin function		2nd pin function	
	Name	Description	Name	Description
N14	I0_SB_A8	2nd GPIO south bank signal A8	CLK0	1st clock input
P12	I0_SB_A7	2nd GPIO south bank signal A7	CLK1	2nd clock input
P14	I0_SB_A6	2nd GPIO south bank signal A6	CLK2	3rd clock input
R13	I0_SB_A5	2nd GPIO south bank signal A5	CLK3	4th clock input
T12	SER_CLK	SerDes clock, positive LVDS signal (or single ended)	–	–

Parameter Description

Table 7.5: *CC_PLL parameter description*

Parameter	Type	Default	Description
REF_CLK	REAL	0.0	Input reference clock in MHz, e.g. 13.3
OUT_CLK	REAL	0.0	Output core clock in MHz, e.g. 200.0
PERF_MD	STRING	"UNDEFINED"	FPGA operation mode for VDD_PLL: "LOWPOWER": Low power 0.9 V "ECONOMY": Economy 1.0 V "SPEED": Speed 1.1 V If not specified, the global setting of Place & Route is used.
LOW_JITTER	INT[0:0]	1	Low Jitter mode: 0: disable 1: enable (default)
LOCK_REQ	INT[0:0]	1	Lock status required before PLL output enable: 0: disable 1: enable (default)
CLK270_DOUB	INT[0:0]	0	Frequency doubling of CLOCK_270 PLL output: 0: disable (default) 1: enable
CLK180_DOUB	INT[0:0]	0	Frequency doubling of CLOCK_180 PLL output: 0: disable (default) 1: enable
CI_FILTER_CONST	INT	2	Optional: Integral coefficient of loop filter, should be greater than zero.
CP_FILTER_CONST	INT	4	Optional: Proportional coefficient of loop filter, should be greater than CI. The higher the CP/CI ratio is, the more stable is the loop (phase margin). Higher CP lead to larger period jitter.

7.4 CC_PLL_ADV

The CC_PLL_ADV primitive is a blackbox element of the built-in clock generators and must be instantiated manually. The user has full control over the PLL by specifying a total of two 96-bit configuration vectors that can be selected during runtime using the USR_SET_SEL input.

Note that the PLL can behave unexpectedly when specifying configuration parameters outside the recommended scope. It is strongly recommended to use the CC_PLL primitive or that the configuration of the CC_PLL_ADV primitive is accomplished through the use of the GateMate™ PLL configuration tool.¹

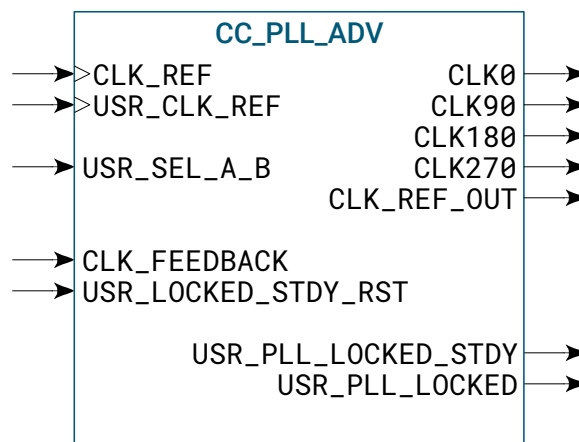


Figure 7.5: CC_PLL_ADV primitive schematic

When instantiating the CC_PLL_ADV primitive, either CLK_REF or USR_CLK_REF must be driven from via an input buffer CC_IBUF to the primitive input port. CLK_REF can be connected to any signal listed in Table 7.4 on page 112. USR_CLK_REF can be connected to any GPIO input. This includes the dedicated clock inputs IO_SB_A5 .. IO_SB_A8 (CLK3 .. CLK0) as well.

Each clock output of the PLL that is connected to the FPGA-internal circuitry must also be fed into the clock network via a global buffer primitive CC_BUF.

¹ Please contact the Cologne Chip support.

Verilog Instantiation

```
CC_PLL_ADV #(
    .PLL_CFG_A(96'bx),
    .PLL_CFG_B(96'bx)
) pll_inst (
    .CLK_REF(),
    .USR_CLK_REF(),
    .USR_SEL_A_B(), // select PLL configuration
    .CLK_FEEDBACK(),
    .USR_LOCKED_STDY_RST(),
    .USR_PLL_LOCKED_STDY(),
    .USR_PLL_LOCKED(),
    .CLK0(),
    .CLK90(),
    .CLK180(),
    .CLK270(),
    .CLK_REF_OUT()
);
```

VHDL Instantiation

```
pll_adv_inst: CC_PLL_ADV
generic map (
    PLL_CFG_A => X"X", -- 96-bit configuration vector A
    PLL_CFG_B => X"X"  -- 96-bit configuration vector A
)
port map (
    CLK_REF          => CLK_REF,
    USR_CLK_REF      => USR_CLK_REF,
    USR_SEL_A_B      => USR_SEL_A_B, -- select PLL configuration
    CLK_FEEDBACK     => CLK_FEEDBACK,
    USR_LOCKED_STDY_RST => USR_LOCKED_STDY_RST,
    USR_PLL_LOCKED_STDY => USR_PLL_LOCKED_STDY,
    USR_PLL_LOCKED    => USR_PLL_LOCKED,
    CLK0              => CLK0,
    CLK90             => CLK90,
    CLK180            => CLK180,
    CLK270            => CLK270,
    CLK_REF_OUT       => CLK_REF_OUT
);
```

Port Description

Table 7.6: *CC_PLL_ADV port description*

Port	Direction	Width	Description
CLK_REF	Input	1	Reference clock signal from dedicated clock pin. See Table 7.4
USR_CLK_REF	Input	1	Alternative reference clock signal from FPGA-internal circuitry
USR_SEL_A_B	Input	1	Input select signal for PLL setup 1 or 2
CLK_FEEDBACK	Input	1	Feedback clock signal
USR_LOCKED_STDY_RST	Input	1	Reset of USR_PLL_LOCKED_STDY, must be set to 1 for a minimum 2 cycles of CLK_REF
USR_PLL_LOCKED_STDY	Output	1	PLL permanent lock status signal
USR_PLL_LOCKED	Output	1	PLL lock status signal
CLK0	Output	1	PLL clock output, no phase shift
CLK90	Output	1	PLL clock output, 90° phase shift
CLK180	Output	1	PLL clock output, 180° phase shift
CLK270	Output	1	PLL clock output, 270° phase shift
CLK_REF_OUT	Output	1	PLL reference clock output

Parameter Description

Table 7.7: *CC_PLL_ADV parameter description*

Parameter	Type	Default	Description
PLL_CFG_A	INT[95:0]	0	First 96-bit configuration vector
PLL_CFG_B	INT[95:0]	0	Second 96-bit configuration vector

7.5 CC_SERDES

The CC_SERDES primitive is a build-in serializer / deserializer (SerDes) for high speed data transfer.

The core of the SerDes primitive is composed of an all-digital phase-locked loop (ADPLL), a transmitter (TX) and a receiver (RX). It is configured via an integrated register file, which can be accessed both from the user side and the configuration controller. The ADPLL of the SerDes generates the bit rate clock for the serial transmission from an external reference clock. Additionally, it generates the clock for the operation of the transmitter and receiver data paths.

File `ug1001-gatamate1-attachment-latest.zip` [↗](#) is published together with this document. It contains the Verilog and VHDL instantiations `cc_serdes_inst.v` and `cc_serdes_inst.vhd`.

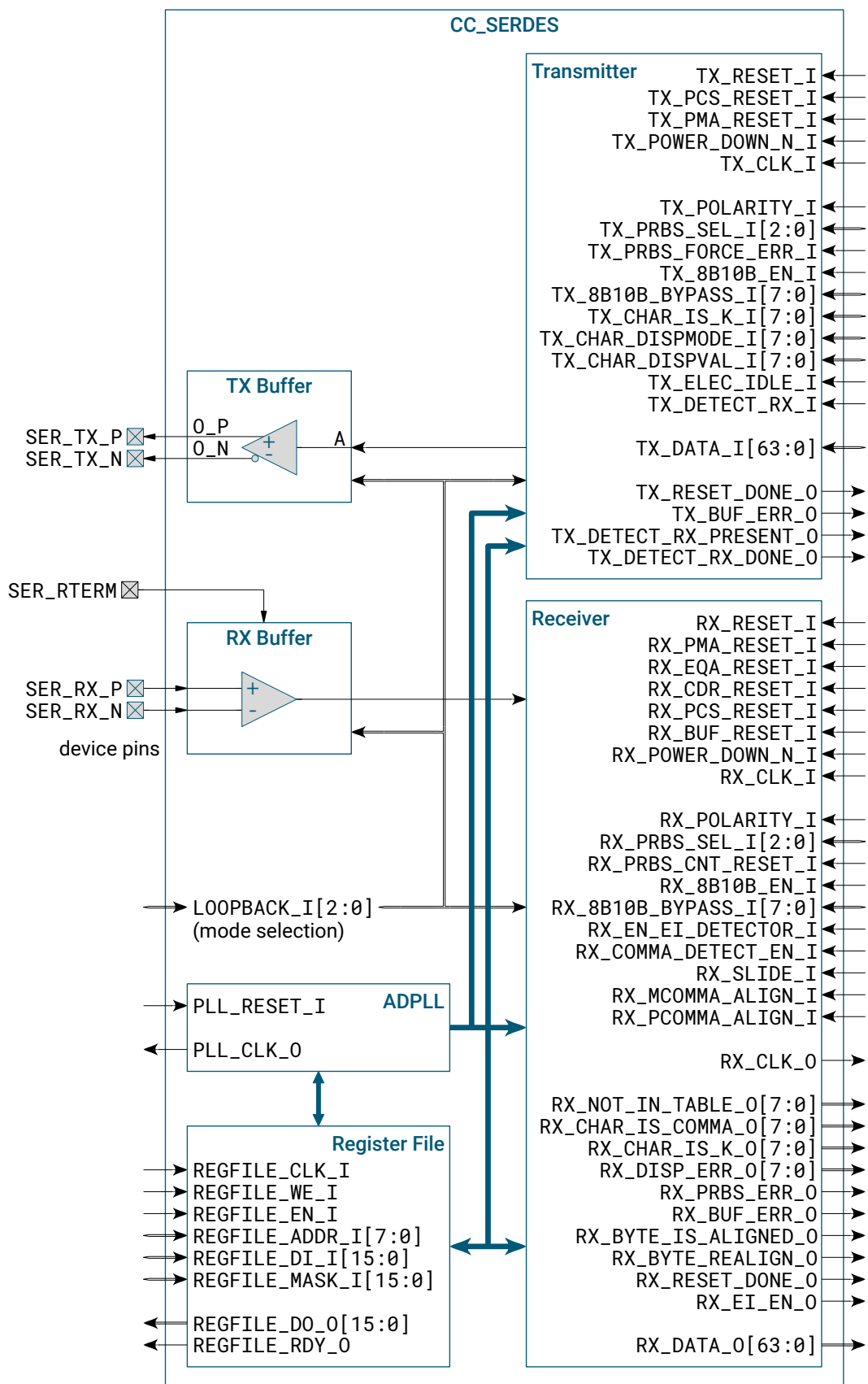


Figure 7.6: CC_SERDES primitive blockdiagram

Port Description

Table 7.8: CC_SERDES transmitter port description

Port	Direction	Width	Description
TX_RESET_I	Input	1	Asynchronous transmitter datapath reset
TX_PCS_RESET_I	Input	1	Asynchronous transmitter PCS reset
TX_PMA_RESET_I	Input	1	Asynchronous transmitter PMA reset
TX_POWER_DOWN_N_I	Input	1	Asynchronous transmitter power down
TX_CLK_I	Input	1	Transmitter datapath clock
TX_POLARITY_I	Input	1	Transmitter polarity inversion control
TX_PRBS_SEL_I	Input	3	Transmitter PRBS mode selection 0: Bypass PRBS generator 1: PRBS-7 2: PRBS-15 3: PRBS-23 4: PRBS-31 5: Reserved 6: 2 UI square wave 7: 20 / 40 / 80 UI square wave
TX_PRBS_FORCE_ERR_I	Input	1	Transmitter PRBS error injection
TX_8B10B_EN_I	Input	1	Transmitter 8B / 10B encoding enable
TX_8B10B_BYPASS_I	Input	8	Transmitter per byte 8B / 10B bypass
TX_CHAR_IS_K_I	Input	8	Transmitter control / data word selection
TX_CHAR_DISPMODE_I	Input	8	Transmitter per byte disparity mode
TX_CHAR_DISPVAL_I	Input	8	Transmitter per byte disparity value
TX_ELEC_IDLE_I	Input	1	Transmitter electrical idle enable
TX_DETECT_RX_I	Input	1	Transmitter detection enable
TX_DATA_I	Input	64	Transmitter datapath
TX_RESET_DONE_0	Output	1	Transmitter datapath reset done indicator
TX_BUF_ERR_0	Output	1	Transmitter buffer error
TX_DETECT_RX_PRESENT_0	Output	1	Transmitter present indicator
TX_DETECT_RX_DONE_0	Output	1	Transmitter detection done

Table 7.9: CC_SERDES receiver port description

Port	Direction	Width	Description
RX_RESET_I	Input	1	Asynchronous receiver datapath reset
RX_PMA_RESET_I	Input	1	Asynchronous receiver PMA reset
RX_EQA_RESET_I	Input	1	Asynchronous receiver DFE reset
RX_CDR_RESET_I	Input	1	Asynchronous receiver CDR reset
RX_PCS_RESET_I	Input	1	Asynchronous receiver PCS reset
RX_BUF_RESET_I	Input	1	Asynchronous receiver buffer reset
RX_POWER_DOWN_N_I	Input	1	Asynchronous receiver power down
RX_CLK_I	Input	1	Receiver datapath clock
RX_POLARITY_I	Input	1	Receiver polarity inversion control
RX_PRBS_SEL_I	Input	3	Receiver PRBS mode selection 0: Bypass PRBS generator 1: PRBS-7 2: PRBS-15 3: PRBS-23 4: PRBS-31 5: Reserved 6: 2 UI square wave 7: 20 / 40 / 80 UI square wave
RX_PRBS_CNT_RESET_I	Input	1	Receiver PRBS error counter reset
RX_8B10B_EN_I	Input	1	Receiver 8B / 10B decoding enable
RX_8B10B_BYPASS_I	Input	8	Receiver per byte 8B / 10B bypass
RX_EN_EI_DETECTOR_I	Input	1	Receiver electrical idle detect enable
RX_COMMA_DETECT_EN_I	Input	1	Receiver comma detect enable
RX_SLIDE_I	Input	1	Receiver manual comma slide
RX_MCOMMA_ALIGN_I	Input	1	Receiver minus comma alignment enable
RX_PCOMMA_ALIGN_I	Input	1	Receiver plus comma alignment enable
RX_CLK_O	Output	1	Receiver recovered clock output
RX_NOT_IN_TABLE_O	Output	8	Receiver 8B / 10B decode error
RX_CHAR_IS_COMMA_O	Output	8	Receiver comma detected indicator
RX_CHAR_IS_K_O	Output	8	Receiver control / data word indicator
RX_DISP_ERR_O	Output	8	Receiver 8B / 10B disparity error
RX_PRBS_ERR_O	Output	1	Receiver PRBS error
RX_BUF_ERR_O	Output	1	Receiver elastic buffer error
RX_BYTE_IS_ALIGNED_O	Output	1	Receiver comma aligned indicator
RX_BYTE_REALIGN_O	Output	1	Receiver comma realignment indicator
RX_RESET_DONE_O	Output	1	Receiver datapath reset done indicator
RX_EI_EN_O	Output	1	Receiver electrical idle status
RX_DATA_O	Output	64	Receiver datapath

Table 7.10: *CC_SERDES register file port description*

Port	Direction	Width	Description
REGFILE_CLK_I	Input	1	Register file interface clock
REGFILE_WE_I	Input	1	Register file write enable signal
REGFILE_EN_I	Input	1	Register file enable signal
REGFILE_ADDR_I	Input	8	Register file address bus
REGFILE_DI_I	Input	16	Register file data input bus
REGFILE_MASK_I	Input	16	Register file data input bus mask
REGFILE_DO_0	Output	16	Register file data output bus
REGFILE_RDY_0	Output	1	Register file ready indicator signal

Table 7.11: *CC_SERDES PLL and miscellaneous port description*

Port	Direction	Width	Description
PLL_RESET_I	Input	1	Asynchronous ADPLL reset
PLL_CLK_0	Output	1	Datapath clock output, frequency divider depending on DIV_VAL_0 which is set by the register file
LOOPBACK_I[2:0]	Input	1	Loopback mode selection 0: Normal operation 1: Near-end PCS loopback 2: Near-end PMA loopback 3: Reserved 4: Far-end PMA loopback 5: Reserved 6: Far-end PCS loopback 7: Reserved

Parameter Description

Table 7.12: CC_SERDES transmitter parameter list

Parameter	Width	Default	Parameter	Width	Default
TX_SEL_PRE	5	0	TX_CM_THRESHOLD_1_RXDET	5	16
TX_SEL_POST	5	0	TX_CALIB_EN	1	0
TX_AMP	5	15	TX_CALIB_OVR	1	0
TX_BRANCH_EN_PRE	5	0	TX_CALIB_VAL	4	0
TX_BRANCH_EN_MAIN	6	0x3F	TX_CM_REG_KI	8	0x80
TX_BRANCH_EN_POST	5	0	TX_CM_SAR_EN	1	0
TX_TAIL_CASCADE	3	4	TX_CM_REG_EN	1	1
TX_DC_ENABLE	7	63	TX_PMA_RESET_TIME	5	3
TX_DC_OFFSET	5	0	TX_PCS_RESET_TIME	5	3
TX_CM_RAISE	5	0	TX_PCS_RESET_OVR	1	0
TX_CM_THRESHOLD_0	5	14	TX_PCS_RESET	1	0
TX_CM_THRESHOLD_1	5	16	TX_PMA_RESET_OVR	1	0
TX_SEL_PRE_EI	5	0	TX_PMA_RESET	1	0
TX_SEL_POST_EI	5	0	TX_RESET_OVR	1	0
TX_AMP_EI	5	15	TX_RESET	1	0
TX_BRANCH_EN_PRE_EI	5	0	TX_PMA_LOOPBACK	2	0
TX_BRANCH_EN_MAIN_EI	6	0x3F	TX_PCS_LOOPBACK	1	0
TX_BRANCH_EN_POST_EI	5	0	TX_DATAPATH_SEL	2	3
TX_TAIL_CASCADE_EI	3	4	TX_PRBS_OVR	1	0
TX_DC_ENABLE_EI	7	63	TX_PRBS_SEL	3	0
TX_DC_OFFSET_EI	5	0	TX_PRBS_FORCE_ERR	1	0
TX_CM_RAISE_EI	5	0	TX_LOOPBACK_OVR	1	0
TX_CM_THRESHOLD_0_EI	5	14	TX_POWERDOWN_OVR	1	0
TX_CM_THRESHOLD_1_EI	5	16	TX_POWERDOWN_N	1	0
TX_SEL_PRE_RXDET	5	0	TX_ELEC_IDLE_OVR	1	0
TX_SEL_POST_RXDET	5	0	TX_ELEC_IDLE	1	0
TX_AMP_RXDET	5	15	TX_DETECT_RX_OVR	1	0
TX_BRANCH_EN_PRE_RXDET	5	0	TX_DETECT_RX	1	0
TX_BRANCH_EN_MAIN_RXDET	6	0x3F	TX_POLARITY_OVR	1	0
TX_BRANCH_EN_POST_RXDET	5	0	TX_POLARITY	1	0
TX_TAIL_CASCADE_RXDET	3	4	TX_8B10B_EN_OVR	1	0
TX_DC_ENABLE_RXDET	7	63	TX_8B10B_EN	1	0
TX_DC_OFFSET_RXDET	5	0	TX_DATA_OVR	1	0
TX_CM_RAISE_RXDET	5	0	TX_DATA_CNT	3	0
TX_CM_THRESHOLD_0_RXDET	5	14	TX_DATA_VALID	1	0

Table 7.13: CC_SERDES receiver parameter list

Parameter	Width	Default	Parameter	Width	Default
RX_BUF_RESET_TIME	5	3	RX_MON_PH_OFFSET	6	0
RX_PCS_RESET_TIME	5	3	RX_EI_BIAS	4	0
RX_RESET_TIMER_PRESC	5	0	RX_EI_BW_SEL	4	4
RX_RESETDONE_GATE	1	0	RX_EN_EI_DETECTOR_OVR	1	0
RX_CDR_RESET_TIME	5	3	RX_EN_EI_DETECTOR	1	0
RX_EQA_RESET_TIME	5	3	RX_DATA_SEL	1	0
RX_PMA_RESET_TIME	5	3	RX_BUF_BYPASS	1	0
RX_WAIT_CDR_LOCK	1	1	RX_CLKCOR_USE	1	0
RX_CALIB_EN	1	0	RX_CLKCOR_MIN_LAT	6	32
RX_CALIB_OVR	1	0	RX_CLKCOR_MAX_LAT	6	39
RX_CALIB_VAL	4	0	RX_CLKCOR_SEQ_1_0	10	0x1 F7
RX_RTERM_VCMSEL	3	4	RX_CLKCOR_SEQ_1_1	10	0x1 F7
RX_RTERM_PD	1	0	RX_CLKCOR_SEQ_1_2	10	0x1 F7
RX_EQA_CKP_LF	8	0x A3	RX_CLKCOR_SEQ_1_3	10	0x1 F7
RX_EQA_CKP_HF	8	0x A3	RX_PMA_LOOPBACK	1	0
RX_EQA_CKP_OFFSET	8	0x 01	RX_PCS_LOOPBACK	1	0
RX_EN_EQA	1	0	RX_DATAPATH_SEL	2	3
RX_EQA_LOCK_CFG	4	0	RX_PRBS_OVR	1	0
RX_TH_MON1	5	8	RX_PRBS_SEL	3	0
RX_EN_EQA_EXT_VALUE	4	0	RX_LOOPBACK_OVR	1	0
RX_TH_MON2	5	8	RX_PRBS_CNT_RESET	1	0
RX_TAPW	5	8	RX_POWERDOWN_OVR	1	0
RX_AFE_OFFSET	5	8	RX_POWERDOWN_N	1	0
RX_EQA_CONFIG	16	0x 01 C0	RX_RESET_OVR	1	0
RX_AFE_PEAK	5	16	RX_RESET	1	0
RX_AFE_GAIN	4	8	RX_PMA_RESET_OVR	1	0
RX_AFE_VCMSEL	3	4	RX_PMA_RESET	1	0
RX_CDR_CKP	8	0x F8	RX_EQA_RESET_OVR	1	0
RX_CDR_CKI	8	0	RX_EQA_RESET	1	0
RX_CDR_TRANS_TH	9	128	RX_CDR_RESET_OVR	1	0
RX_CDR_LOCK_CFG	6	0x 0B	RX_CDR_RESET	1	0
RX_CDR_FREQ_ACC	15	0	RX_PCS_RESET_OVR	1	0
RX_CDR_PHASE_ACC	16	0	RX_PCS_RESET	1	0
RX_CDR_SET_ACC_CONFIG	2	0	RX_BUF_RESET_OVR	1	0
RX_CDR_FORCE_LOCK	1	0	RX_BUF_RESET	1	0
RX_ALIGN_MCOMMA_VALUE	10	0x 2 83	RX_POLARITY_OVR	1	0
RX_MCOMMA_ALIGN_OVR	1	0	RX_POLARITY	1	0
RX_MCOMMA_ALIGN	1	0	RX_8B10B_EN_OVR	1	0

Continued on next page

Table 7.13: CC_SERDES receiver parameter list

Continued from previous page

Parameter	Width	Default	Parameter	Width	Default
RX_ALIGN_PCOMMA_VALUE	10	0x17C	RX_8B10B_EN	1	0
RX_PCOMMA_ALIGN_OVR	1	0	RX_8B10B_BYPASS	8	0
RX_PCOMMA_ALIGN	1	0	RX_BYTE_REALIGN	1	0
RX_ALIGN_COMMA_WORD	2	0	RX_DBG_EN	1	0
RX_ALIGN_COMMA_ENABLE	10	0x3FF	RX_DBG_SEL	2	0
RX_SLIDE_MODE	2	0	RX_DBG_MODE	1	0
RX_COMMA_DETECT_EN_OVR	1	0	RX_DBG_SRAM_DELAY	6	0x05
RX_COMMA_DETECT_EN	1	0	RX_DBG_ADDR	10	0
RX_SLIDE	2	0	RX_DBG_RE	1	0
RX_EYE_MEAS_EN	1	0	RX_DBG_WE	1	0
RX_EYE_MEAS_CFG	15	0	RX_DBG_DATA	20	0

Table 7.14: CC_SERDES PLL and miscellaneous parameter list

Parameter	Width	Default	Parameter	Width	Default
PLL_EN_ADPLL_CTRL	1	0	PLL_FILTER_SHIFT	2	2
PLL_CONFIG_SEL	1	0	PLL_SAR_LIMIT	3	2
PLL_SET_OP_LOCK	1	0	PLL_FT	11	0x200
PLL_ENFORCE_LOCK	1	0	PLL_OPEN_LOOP	1	0
PLL_DISABLE_LOCK	1	0	PLL_SCAP_AUTO_CAL	1	1
PLL_LOCK_WINDOW	1	1	PLL_BISC_MODE	3	4
PLL_FAST_LOCK	1	1	PLL_BISC_TIMER_MAX	4	0xF
PLL_SYNC_BYPASS	1	0	PLL_BISC_OPT_DET_IND	1	0
PLL_PFD_SELECT	1	0	PLL_BISC_PFD_SEL	1	0
PLL_REF_BYPASS	1	0	PLL_BISC_DLY_DIR	1	0
PLL_REF_SEL	1	0	PLL_BISC_COR_DLY	3	1
PLL_REF_RTERM	1	1	PLL_BISC_CAL_SIGN	1	0
PLL_FCNTL	6	0x3A	PLL_BISC_CAL_AUTO	1	1
PLL_MAIN_DIVSEL	6	0x1B	PLL_BISC_CP_MIN	5	4
PLL_OUT_DIVSEL	2	0	PLL_BISC_CP_MAX	5	0x12
PLL_CI	5	3	PLL_BISC_CP_START	5	0xC
PLL_CP	10	0x50	PLL_BISC_DLY_PFD_MON_REF	5	0
PLL_A0	4	0	PLL_BISC_DLY_PFD_MON_DIV	5	2
PLL_SCAP	3	0			
SERDES_ENABLE	1	0			

7.6 CC_CFG_CTRL

The GateMate™ can be partially reconfigured out of the CPE array by using the CC_CFG_CTRL primitive. The user design must provide a clock, valid and enable signal in order to call attention to this source of configuration and then stream the commands into the configuration logic.

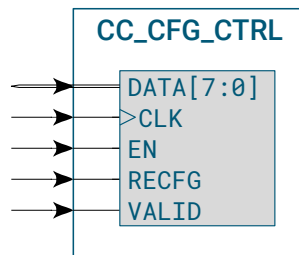


Figure 7.7: CC_CFG_CTRL primitive schematic

The following constraints must be met for successful configuration:

- The enable must be set at least 3 clock cycles before the first data is streamed in, i.e. before the data valid signal is logic high for first time.
- Between two successive bytes there must be at least one idle cycle. There can be an arbitrary number of idle cycles but it must be at least one.
- The enable signal should not be de-asserted until at least eight cycles after the last byte was transmitted.
- The reconfiguration enable signal must be active if the reconfiguration bit has not been set during the previous configuration.

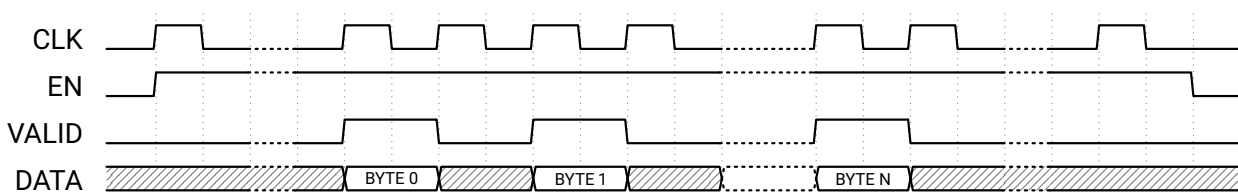


Figure 7.8: Configuration from CPE array

Port Description

Table 7.15: *CC_CFG_CTRL port description*

Port	Direction	Width	Description
DATA	Input	8	Configuration data byte input
CLK	Input	1	Clock to configuration controller
EN	Input	1	Enable signal
RECFG	Input	1	Reconfigure-enable signal
VALID	Input	1	Data valid pulses

Verilog Instantiation

```
CC_CFG_CTRL cfg_ctrl_inst (
    .DATA(DATA),    // Configuration data byte
    .CLK(CLK),      // Configuration clock
    .EN(EN),        // Enable signal
    .RECFG(RECFG),  // Reconfigure-enable signal
    .VALID(VALID)   // Data valid pulse signal
);
```

VHDL Instantiation

```
cfg_ctrl_inst: CC_CFG_CTRL
port map (
    DATA => DATA,  -- Configuration data byte
    CLK   => CLK,    -- Configuration clock
    EN    => EN,     -- Enable signal
    RECFG => RECFG,  -- Reconfigure-enable signal
    VALID => VALID   -- Data valid pulse signal
);
```

Acronyms

ADPLL	all-digital phase-locked loop	117, 121
CDR	Clock and Data Recovery	119
CPE	Cologne Programmable Element	5, 6, 13, 46, 49, 52, 53, 55, 59, 66, 67, 69, 107, 125
DDR	double data rate	40, 42
DFE	decision feedback equalizer	119
DPSRAM	dual port SRAM	76
ECC	error checking and correcting	76–78, 95, 97
FIFO	first-in, first-out memory	6, 8, 13, 94, 95, 97–103
GPIO	general purpose input / output	16, 19, 22, 25, 40, 42, 107, 109, 114
L2T4	4-input tree-structure lookup table	53, 55
L2T5	5-input tree-structure lookup table	55
LSB	least significant bit	69
LUT	lookup table	13, 53, 55, 57, 59
LUT-1	1-input lookup table	8, 57
LUT-2	2-input lookup table	8, 52, 53, 55, 57
LUT-3	3-input lookup table	8, 52, 59
LUT-4	4-input lookup table	8, 52, 59
LUT-tree	tree-structure lookup table	5, 52, 53, 55, 59

LVC MOS	low voltage CMOS	16, 19, 22, 25
LVDS	low-voltage differential signaling	28, 31, 34, 37
MSB	most significant bit	69
MUX	multiplexer	13
PCS	physical coding sublayer	117, 119, 121
PLL	phase-locked loop	9, 13, 109, 111, 112, 114, 115, 121, 124
PMA	physical media attachment	117, 119, 121
PRBS	pseudo-random bit stream	117, 119
RAM	random-access memory	8, 13, 76, 78–82, 84, 94, 97
ROM	read-only memory	76, 81
SDP	simple dual port	8, 76, 78–82, 94, 95, 97
SerDes	serializer / deserializer	11, 117
TDP	true dual port	8, 76, 78–82, 94, 95, 97
VHDL	Very High Speed Integrated Circuit Hardware Description Language	12

GateMate™ FPGA User Guide
Primitives Library
UG1001
July 2024

