



NG-MEDIUM

NX1H35S

Library Guide

Ver 1.5

May 2019

NANOXPLORE CONFIDENTIAL

Table of Content

TABLE OF CONTENT	3
1 INTRODUCTION.....	6
2 CLOCKS DISTRIBUTION AND MANAGEMENT	7
2.1 NX_CKB.....	7
2.1.1 Description	7
2.1.2 Ports.....	7
2.1.3 Example	7
2.2 NX_CKS.....	7
2.2.1 Description	7
2.2.2 Ports.....	8
2.2.3 Example	8
2.3 NX_PLL.....	8
2.3.1 Description	8
2.3.2 Generics.....	11
2.3.3 Ports.....	14
2.3.4 Instantiation Example	14
2.3.5 Simulation	15
2.4 NX_WFG.....	15
2.4.1 Description	15
2.4.2 Generics.....	15
2.4.3 Ports.....	17
2.4.4 Example	17
2.4.5 Simulation	18
3 CORE LOGIC.....	19
3.1 NX_FE.....	19
3.1.1 Description	19
3.1.2 Generics.....	19
3.1.3 Ports.....	20
3.1.4 Example	21
3.2 NX_CY	21
3.2.1 Description	21
3.2.2 Generics.....	22
3.2.3 Ports.....	24
3.2.4 Example	24
3.3 NX_RFB.....	24
3.3.1 Description	24
3.3.2 Generics.....	26
3.3.3 Ports.....	27
3.3.4 Instantiation Example	27
3.4 NX_DSP.....	28
3.4.1 Description	28
3.4.2 Generics.....	29
3.4.3 Ports.....	33
3.4.4 Instantiation Example	34
3.4.1 Simulation	35
3.5 NX_DSP_SPLIT.....	35
3.6 NX_ECC.....	39
3.6.1 Description	39
3.6.2 Ports.....	39
3.6.3 Instantiation Example	39
3.7 NX_RAM	41
3.7.1 Description	41
3.7.2 Memory ports configurations	42
3.7.3 Generics.....	46

3.7.4	Ports.....	49
3.7.5	Instantiation Example	51
3.7.6	Simulation	51
3.8	NX_RAM_WRAP	52
3.8.1	Description	52
3.8.2	Generics.....	52
3.8.3	Ports.....	52
4	I/O ELEMENTS	54
4.1	NX_IOB	54
4.1.1	Description	54
4.1.2	Generics.....	54
4.1.3	Ports.....	57
4.1.4	Example	57
4.2	NX_IOB_I	57
4.2.1	Description	57
4.2.2	Generics.....	58
4.2.3	Ports.....	60
4.2.4	Example	60
4.3	NX_IOB_O	61
4.3.1	Description	61
4.3.2	Generics.....	61
4.3.3	Ports.....	63
4.3.4	Example	63
4.4	SERIALIZERS AND DESERIALIZERS	63
4.4.1	Introduction	63
4.4.2	SERDES architecture overview	64
4.4.3	DPA : Dynamic Phase Adjustment	65
4.5	NX_DES	69
4.5.1	Description	69
4.5.2	Generics.....	70
4.5.3	Ports.....	72
4.6	NX_SER	73
4.6.1	Description	73
4.6.2	Generics.....	73
4.6.3	Ports.....	75
5	RESERVED.....	76

Figure 1: CKS chronograms	8
Figure 2: Simplified PLL block diagram.....	9
Figure 3: PLL block diagram and settings.....	10
Figure 4: WFG diagram	15
Figure 5: WFG chronograms.....	16
Figure 6: FE simplified diagram.....	19
Figure 7: CARRY logic diagram	22
Figure 8: RFB diagram.....	25
Figure 9: DSP simplified block diagram	28
Figure 10: RAM diagram	41
Figure 11: RAM organization (No ECC).....	43
Figure 12: Address and data connections (No ECC).....	44
Figure 13: RAM organization (ECC FAST or SLOW).....	45
Figure 14: Address and data connections (ECC FAST or SLOW).....	45
Figure 15: IOB diagram	54
Figure 16: IOB_I diagram	58
Figure 17: IOB_O diagram.....	61
Figure 18: SERDES data path simplified diagram	65

Figure 19: SERDES delay lines control block simplified diagram 65

Figure 20: FLD activation..... 66

Figure 21: FLG activation..... 66

Figure 22: Writing and reading delay registers..... 67

Figure 23: SER_DES IP Core simplified diagram 69

Figure 24 NX_DES primitive..... 70

Figure 25: NX_SER primitive 73

NANOXPLORE CONFIDENTIAL

1 Introduction

This document aims at giving guidelines on how to use the provided NX components in VHDL source code. Its purpose is to explain how to correctly instantiate the different supported NX components provided by NanoXplore.

For each NX component, the reader will find a quick introduction and a description of both the generics and ports. He will also find a diagram of the component with an example in VHDL.

NANOXPLORE CONFIDENTIAL

2 Clocks distribution and management

2.1 NX_CKB

2.1.1 Description

The NX_CKB component describes a Clock Buffer circuit that allows user to force internally generated clock to be reinjected in low skew routing network.

2.1.2 Ports

Ports	Direction	Type	Description
I	input	std_logic	Input clock (generated)
O	output	std_logic	Output clock (low skew routing)

2.1.3 Example

This documentation only provides the instantiation of the component. For a full example, please refer to NX_CKB example provided in the example/nxLibrary installation directory.

```
CKB_0 : NX_CKB
port map (
    I => CK_GEN
    , O => CK_LS
);
```

2.2 NX_CKS

2.2.1 Description

The NX_CKS component describes a Clock Switch circuit that allows glitch free clock generation. It can be used to enable/disable the clock to part of the user's logic – providing that the output signal will be glitch free – and the delay from the main clock to the generated one is un-significant. See Figure 1 for a detailed chronogram.

The NX_CKS can be used exclusively by instantiation. The current version of NXmap does not yet support inference for this device.

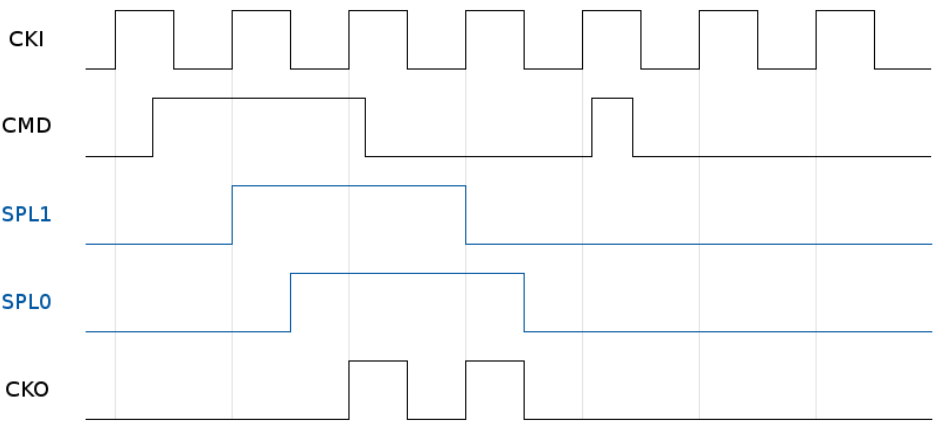


Figure 1: CKS chronograms

The blue internal signals are CMD signal sampled on rising edge (SPL1) and then sampled on falling edge (SPL0). SPL0 is the final enable.

2.2.2 Ports

Ports	Direction	Type	Description
CKI	input	std_logic	Input clock
CMD	input	std_logic	Command
CKO	output	std_logic	Output clock

2.2.3 Example

This documentation only provides the instantiation of the component. For a full example, please refer to NX_CKS example provided in the example/nxLibrary installation directory.

```
CKS_0 : NX_CKS
port map (
    CKI => CK
    , CMD => ENABLE
    , CKO => CKG
);
```

2.3 NX_PLL

2.3.1 Description

The NX_PLL component describes a Phase Locked Loop circuit available in NG-MEDIUM. The PLL just as the WaveForm Generators (WFG) is part of the Clock Generator block (also called CKG). There are 4 CKG blocks, on in each corner of the FPGA die. Each CKG is composed of one PLL and eight WFG.

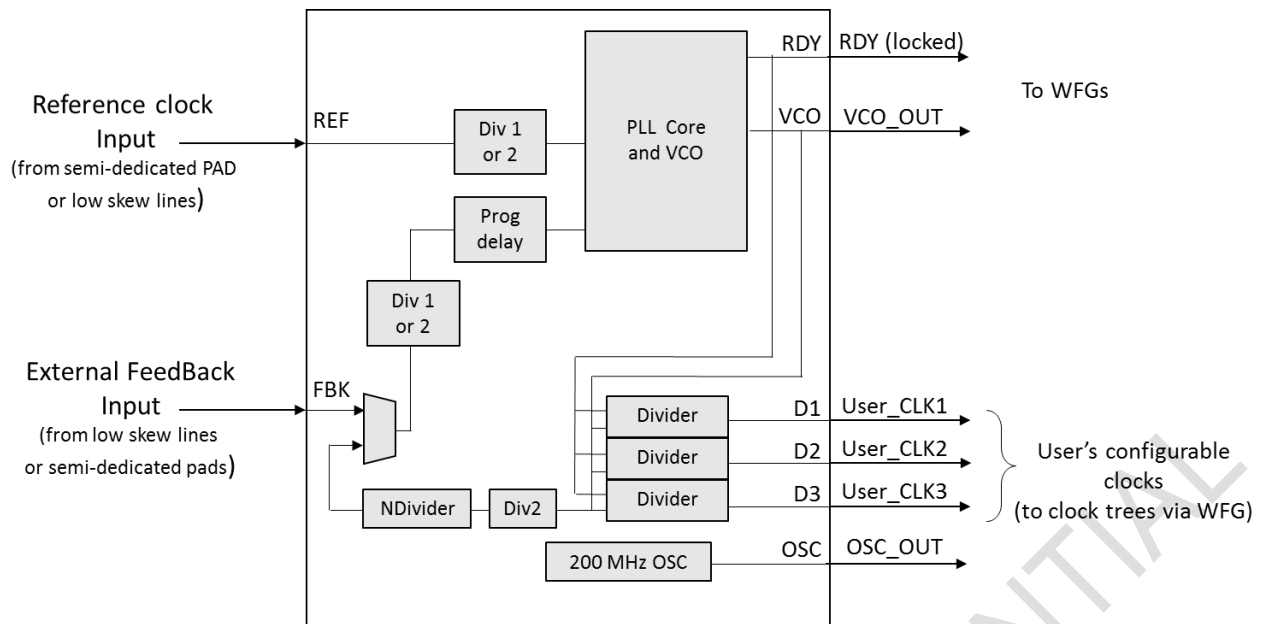


Figure 2: Simplified PLL block diagram

PLL inputs:

- REF: input reference clock. The input reference clock enters in the REF pin.
- FBK: The feedback can be external (via clock tree connected to the FBK pin) for phase controlled outputs, or internal to the PLL (no phase control or adjustment of the generated clocks with the REF pin).

PLL outputs:

- VCO: the output of the VCO
- D1, D2 and D3 : three outputs generated by frequency division of the VCO output
- OSC: Internal 200 MHz oscillator output (used for delays calibration on the PLL feedback path, WFG internal delays and input/output delays). OSC output can also be used as auxiliary clock.
- RDY: status pin. Goes high when the PLL is locked

PLL detailed description and settings:

The next figure shows a more detailed view of the PLL, and the attributes used to configure its functionality.

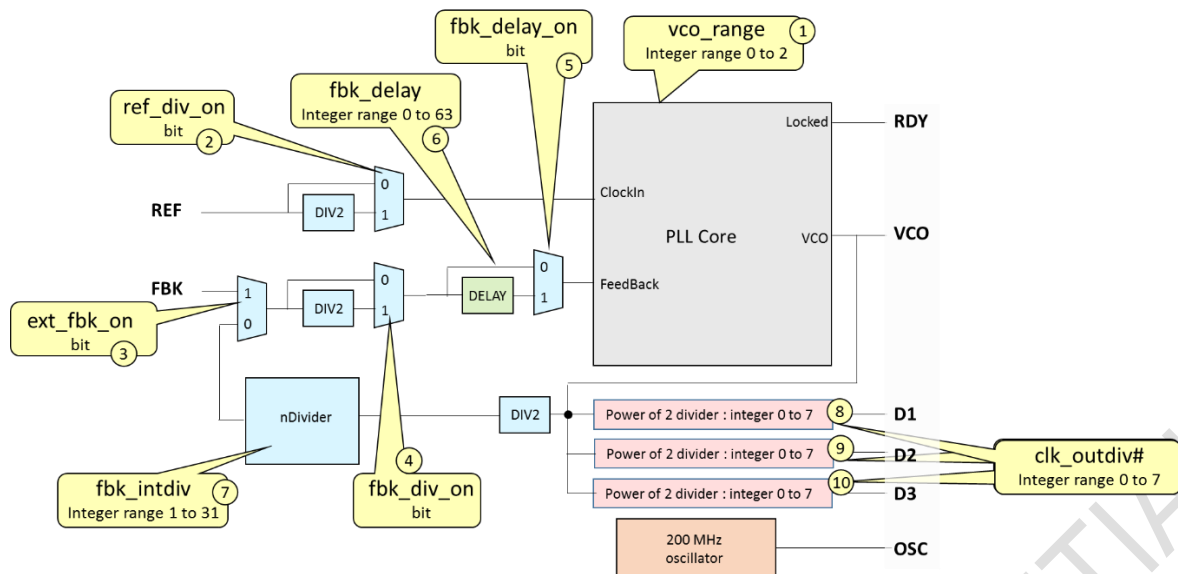


Figure 3: PLL block diagram and settings

The PLL can generate a set of user's defined clocks which frequencies are based on the REFerence input clock, with multiply and/or divide factors.

The PLL outputs connect directly with the WaveForm Generators (WFG) of the same Clock Generator (CKG) for clock buffering and added clock generation flexibility.

- The REFerence clock can be optionally divided by 2. This divider can be used for example to maintain the VCO input frequency in the allowed range (20 to 100 MHz): Assuming that $F(\text{ref}) = 150$ MHz, setting the divider by 2 allows the VCO to see a 75 MHz frequency.
- FeedBack (FBK pin): The feedback can be internal or external.
 - o When external, the feedback must be done by using a clock tree (low skew network). The internally generated clocks can be rising edge aligned with the reference clock input pad (highly recommended to optimize the communications with external components like memories, ADCs, DACs). An optional divider by 2 is included in the external feedback path (user's selected by setting the "vco_fbkdiv" generic to '1')
 - o If internal feedback is chosen, the FBK input pin must be left open. The internal feedback path includes three dividers.
 - Divider by 2 (cannot be bypassed)
 - User's programmable integer divider (nDiv – can divide in the range 2 to 31). See **fbk_intdiv** generic explanations.
 - Additional optional divider by 2 (user's selected by setting the "vco_fbkdiv" generic to '1'). This divider is also included in the external feedback path.
 - o A user's programmable delay chain allows to delay the feedback to the VCO feedback. This feature can be used to fine tune the rising edge alignment of the generated clocks with the REFerence input pad, when external feedback is used – with clock tree. Each delay step is 159 ps. The user can select 0 to 63 steps.
- VCO (PLL core)
 - o The VCO generates a frequency in the range 200 MHz to 1200 MHz.
 - o There are 3 frequency ranges, 200 MHz to 425 MHz, 400 MHz to 850 MHz and 800 to 1200 MHz.

- The VCO range is defined by the “**vco_range**” attribute. NanoXplore recommends to choose preferably the lowest possible range when the VCO frequency value is in the overlap between two ranges.
- Divided outputs
 - There are 3 additional outputs (D1, D2 and D3). Each output has a programmable divider by powers of 2, in the range 1 to 128 (1, 2, 4, 8, 16, 32, 64 or 128).
- Internal 200 MHz oscillator (precision and stability over PVT around 10%)
 - Can be used as auxiliary clock
 - In addition, this oscillator is used by NXmap to calibrate the programmable delays available in :
 - PLL feedback path
 - WFG (to delay the clocks)
 - IOs input, output and tri-state command paths (complex IO banks only)

2.3.2 Generics

vco_range (1)

type integer (range 0 to 2)
 default value 0

This generic configures the VCO frequency range. The value must be in range 0 to 2 according to the following ranges:

vco_range	VCO frequency		Unit
	Min	Max	
0	200	425	MHz
1	400	850	MHz
2	800	1200	MHz

ref_div_on (2)

type bit
 default value '0'

This generic configures whether the input reference frequency is divided by 2 (**vco_refdiv** = '1') or not (**vco_refdiv** = '0'). It can be useful to maintain the input reference clock, and the VCO frequencies into their respective ranges.

ref_div_on	Reference frequency range		Unit
	Min	Max	
'0'	20	100	MHz
'1'	40	200	MHz

ext_fbk_on (3)

type bit
 default value '0'

When '0', the internal feedback path is selected. The nDivider whose value is defined by **fbk_intdiv** and associated pre-divider by 2 are then used.

When '1', the external feedback path is selected. This can be useful to ensure rising edges alignment of the REFERENCE input clock pad and internal clock trees for fully synchronous behavior with external source and destination components.

fbk_div_on (4)

type bit
default value '0'

This generic configures whether the VCO feedback frequency is divided by 2 ('1') or not ('0'). See also ***fbk_intdiv*** to set the global division factor on the internal feedback path.

fbk_delay_on (5)

type bit
default value '0'

This generic configures whether the delay of the feedback path is active ('1') or not ('0').

fbk_delay (6)

type integer (range 0 to 63)
default value 0

The number of delay taps on the feedback path (internal or external) can be adjusted to meet the required phase on the VCO outputs. When using external feedback, it can be used to compensate the delay on the reference clock input to the REF pin of the PLL via the semi-dedicated clock input pin and associated direct routing.

The delay can be selected or not (see ***fbk_delay_on***). When selected, it can be adjusted from 340 ps (***fbk_delay*** = 0) to 10 400 ps (***fbk_delay*** = 63) by steps of 160 ps.

fbk_intdiv (7)

type integer (range 1 to 15 or 2 to 31)
default value 0

This generic allows to define (together with ***fbk_div_on***) the division factor of the VCO frequency on the internal feedback path.

<i>fbk_intdiv</i> (nDivider)	<i>fbk_div_on</i>	Division factor on feedback path
0	'0'	Not allowed
1	'0'	Not allowed
2	'0'	4
3	'0'	6
...	'0'	...
30	'0'	60
31	'0'	62

0	'1'	Not allowed
1	'1'	4
2	'1'	8
3	'1'	12
...	'1'	...
14	'1'	56
15	'1'	60
16 to 31	'1'	Not allowed

clk_outdiv1 (8)

type integer (range 0 to 7)
 default value 0

This generic allows to define the divider value of the D1 output. There are 8 possible values, 1, 2, 4, 8, 16, 32, 64 and 128 ($2^{**}\text{clk_outdiv}(1)$)

If $\text{clk_outdiv}(1) = 0$ (default value)
 $\text{D1_output_frequency} = \text{Fvco}/(2^{**}0) = \text{Fvco}$

If $\text{clk_outdiv}(1) = 7$
 $\text{D1_output_frequency} = \text{Fvco}/(2^{**}7) = \text{Fvco}/128$

clk_outdiv2 (9)

type integer (range 0 to 7)
 default value 0

This generic allows to define the divider value of the D2 output. There are 8 possible values, 1, 2, 4, 8, 16, 32, 64 and 128 ($2^{**}\text{clk_outdiv}(2)$)

If $\text{clk_outdiv}(2) = 0$ (default value)
 $\text{D2_output_frequency} = \text{Fvco}/(2^{**}0) = \text{Fvco}$

If $\text{clk_outdiv}(2) = 7$
 $\text{D2_output_frequency} = \text{Fvco}/(2^{**}7) = \text{Fvco}/128$

clk_outdiv3 (10)

type integer (range 0 to 7)
 default value 0

This generic allows to define the divider value of the D3 output. There are 8 possible values, 1, 2, 4, 8, 16, 32, 64 and 128 ($2^{**}\text{clk_outdiv}(3)$)

If $\text{clk_outdiv}(3) = 0$ (default value)
 $\text{D3_output_frequency} = \text{Fvco}/(2^{**}0) = \text{Fvco}$

If $\text{clk_outdiv}(3) = 7$
 $\text{D3_output_frequency} = \text{Fvco}/(2^{**}7) = \text{Fvco}/128$

Notes about user's adjustable delays on NG-MEDIUM:

The PLL have a user's selectable and adjustable (no delay or 0 to 63 x 160 ps +/- 5% delay taps) on the feedback path. A similar delay chain is available in each WFGs. Finally the IO banks have input, output and tri-state command 64-tap delay chains.

All the delay chain taps are calibrated with the same procedure and hardware resources.

The procedure is transparent to the user.

The delays calibration system uses the PLL 200 MHz oscillator output as reference clock to calibrate all delays: feedback path in the PLL itself, WFG delays in same CKG), and IO delays in the two neighboring complex and simple IO banks:

- CKG1 oscillator calibrates the delays in CKG1 (PLL + WFGs) and IO banks 0, 12 and 11
- CKG2 oscillator calibrates the delays in CKG2 (PLL + WFGs) and IO banks 1, 2 and 3
- CKG3 oscillator calibrates the delays in CKG3 (PLL + WFGs) and IO banks 4, 5, 6 and 7
- CKG4 oscillator calibrates the delays in CKG4 (PLL + WFGs) and IO banks 8, 9 and 10

The calibration procedure takes about 10 μ s at startup. No status is available on NG-MEDIUM.

2.3.3 Ports

Ports	Direction	Type	Description
REF	In	std_logic	Reference clock input Connectivity: semi-dedicated clock inputs, clock trees (low skew network)
FBK	In	std_logic	External FeedBack input Connectivity: semi-dedicated clock inputs, clock trees (low skew network)
VCO	Out	std_logic	VCO output Connectivity: WFG inputs
D1...D3	Out	std_logic	Divided clocks. Fvco frequency divided by 1, 2, 4, 8, 16, 32, 64 or 128 Important note: D1, D2 and D3 outputs are reset while PLL RDY is not asserted. Connectivity: WFG inputs
OSC	Out	std_logic	Internal 200 MHz oscilator Connectivity :WFG inputs, delay calibration system
RDY	Out	std_logic	High when PLL is locked Connectivity: RDY inputs of WFGs, fabric...

2.3.4 Instantiation Example

This documentation only provides the instantiation of the component. For a full example, please refer to NX_PLL example provided in the example/nxLibrary installation directory.

```
-- targetFreq = (refFreq * (2 * fbk_intdiv)) / (2^clk_outdiv1))
-- 12.5 MHz = (25 MHz * (2 * 4) / (2^4))
-- 50 MHz = (25 MHz * (2 * 4) / (2^2))
--
-- Please note that (refFreq * (2 * fbk_intdiv)) must be above 200 MHz and
below 1200 MHz

PLL_0 : NX_PLL
  generic map (
```

```

        fbk_intdiv => 4
        , clk_outdiv1 => 4  -- Divide by 2**4 = 16
        , clk_outdiv2 => 2  -- Divide by 2**2 = 4
    )
port map (
    REF => ck25MHz
    , FBK => OPEN
    , VCO => OPEN,
    , D1  => ck12_5MHz
    , D2  => ck50MHz
    , D3  => OPEN
    , OSC => OPEN
    , RDY => OPEN
);

```

2.3.5 Simulation

The NX_PLL VHDL simulation model is included in the NxLibrary (NxPackage.vhd). It allows to simulate any one of the possible NX_PLL configurations.

2.4 NX_WFG

2.4.1 Description

The NX_WFG component is used to access the low skew lines and clock trees. Among the main WFG features:

- User's selectable clock inversion
- Programmable delay line (0 to 64 taps)
- Waveform generation by using a 2 to 16-tap user's defined pattern
- Includes synchronization with other WFG using pattern, in the same Clock Generator

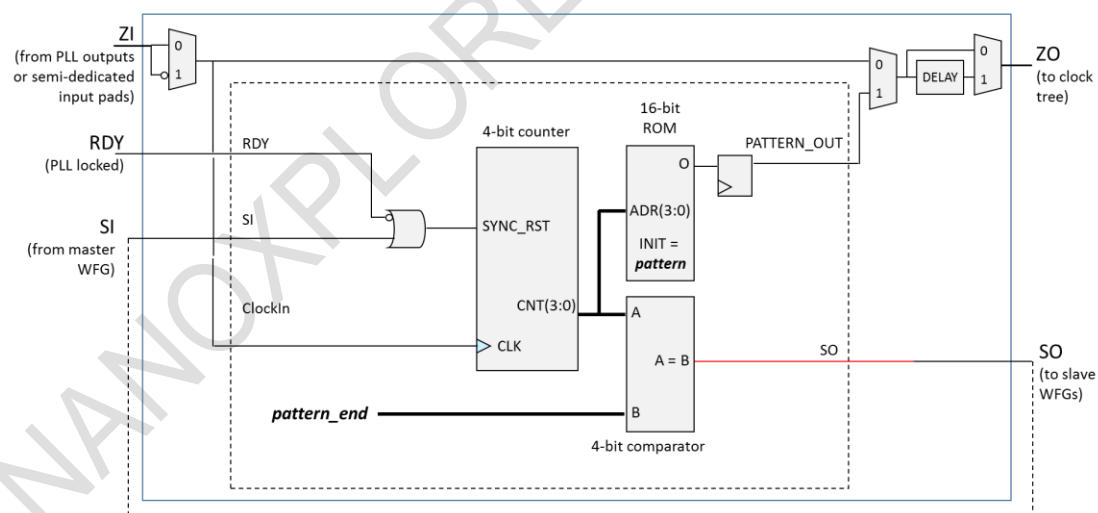


Figure 4: WFG diagram

2.4.2 Generics

delay

type	integer
default value	0

This generic represents the delay line tap count. The value must be in range [0:63] for a tap count in range [1:64].

delay_on

type bit
default value '0'

This generic configures whether the generated clock is delayed ('1') or not ('0').

pattern_end

type integer
default value 0

This generic configures the last useful index of the sampling pattern. The value must be in range [0:15]. When set to 1 only the 2 first bits of the pattern are used to sample the input clock.

mode

type bit
default value '0'

This generic configures whether the generated clock is using the WFG pattern ('1') or not ('0').

pattern

type bit_vector(0 to 15)
default value b"0000000000000000"

This generic configures the sampling pattern. The pattern is temporal which means the first bit considered is left most one.

For example, with a pattern set to b"1000000000000000" and a pattern_end set to 2, only the first 3 bits of the pattern are considered ("100") and the input and output clocks chronograms are:

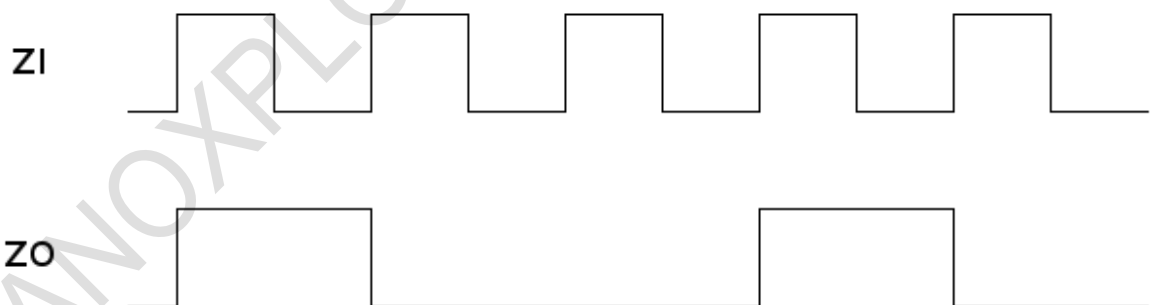


Figure 5: WFG chronograms

wfg_edge

type bit
default value '0'

This generic configures whether the input clock is inverted ('1') or not ('0'). When sampling the input clock, this generic configures whether the sampling is done on rising edge ('0') or falling edge ('1').

2.4.3 Ports

Ports	Direction	Type	Description
SI	input	std_logic	Synchronization input (connected to the synchronization output of the master WFG)
ZI	input	std_logic	Input clock (connected to PLL VCO or D1, D2 or D3 output)
RDY	input	std_logic	Usually connected to the PLL RDY pin. Must be left unconnected for the WFG that generates the clock feedback for the PLL using external feedback. RDY input is an active low reset. When low, it disables the WFG behavior. When high or open, the WFG works as specified.
SO	output	std_logic	Synchronization output (Master WFG SO output is connected to all slave WFGs SI inputs)
ZO	output	std_logic	Generated clock (connected to clock tree)

When sampling the input clock, the synchronization input must be connected either to another WFG (using pattern) synchronization output or the synchronization output of the WFG itself.

2.4.4 Example

This documentation only provides the instantiation of the component. For a full example, please refer to NX_WFG example provided in the example/nxLibrary installation directory.

```
-- CK50MHz at 50 MHz
-- NOTCK    = ~CK50MHz
-- CK25MHz = CK50MHz / 2

WFG_0 : NX_WFG
generic map (
    wfg_edge => '1'
)
port map (
    SI => OPEN, SO => OPEN, RDY => OPEN
    , ZI => CK50MHz, ZO => NOTCK
);

WFG_1 : NX_WFG
Generic map (
    mode      => '1'
    , pattern_end => 1
    , pattern  => b"100000000000000000"
)
port map (
```

```
    SI => SYNC, SO => SYNC, RDY => OPEN  
    , ZI => CK50MHz, ZO => CK25MHz  
);
```

2.4.5 Simulation

The NX_WFG VHDL simulation model is included in the NxLibrary (NxPackage.vhd). It allows to simulate any one of the possible NX_WFG configurations.

NANOXPLORE CONFIDENTIAL

3 Core logic

3.1 NX_FE

3.1.1 Description

The NX_FE component describes a Functional Element circuit which is composed of a LUT associated to a DFF as shown in the following diagram:

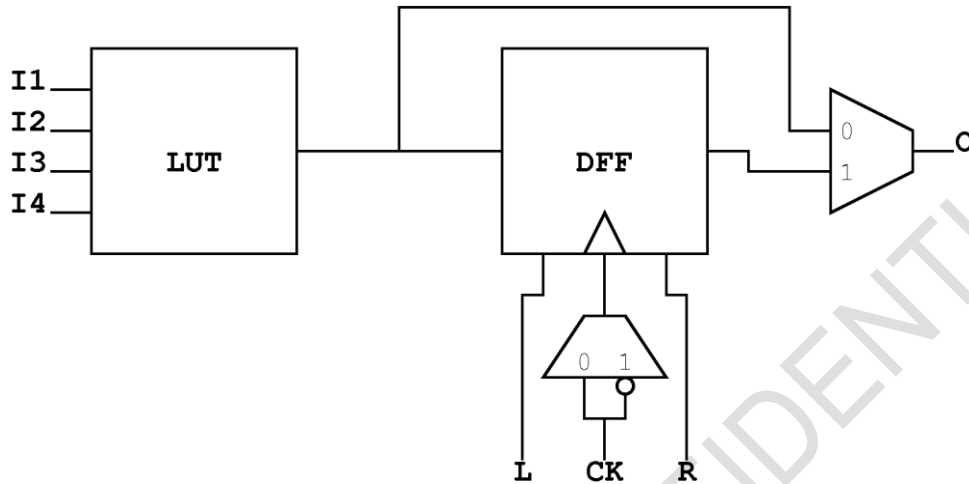


Figure 6: FE simplified diagram

3.1.2 Generics

lut_table

type bit_vector(15 downto 0)

default value b"0000000000000000"

This generic represents the truth table of the associated LUT. The string representing the truth table is MSB ordered ("b(15), b(14),...b(1), b(0)") and b(15) to b(0) are defined as in the following table:

I4	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
I3	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0
I2	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0
I1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
O	b(15)	b(14)	b(13)	b(12)	b(11)	b(10)	b(9)	b(8)	b(7)	b(6)	b(5)	b(4)	b(3)	b(2)	b(1)	b(0)

Lut_table examples for common 4-input functions:

- I4 and I3 and I2 and I1 => lut_table = b"1000000000000000" (or x"8000")
- I4 or I3 or I2 or I1 => lut_table = b"1111111111111110" (or x"FFFE")
- (I4 and I3) xor (I2 and I1) => lut_table = x"0111 1000 1000 1000" (or x"7888")

dff_ctxt

type std_logic
 default value 'U'

This generic represents the initial value of the associated DFF. The initial value is set by bitstream. The available values are: 'U' for undefined (no value set in bitstream), '0' for low and '1' for high.

dff_edge

type bit
 default value '0'

This generic represents the front polarity of the clock of the associated DFF. '0' is for rising edge and '1' for falling edge.

dff_init

type bit
 default value '0'

This generic represents whether the DFF consider the R (reset) input. '0' is for ignore and '1' for using connected net.

dff_load

type bit
 default value '0'

This generic represents whether the DFF consider the L (load) input. '0' is for ignore and '1' for using connected net.

dff_mode

type bit
 default value '0'

This generic represents whether the Functional Element use the associated DFF. '0' is for bypass DFF and '1' for use DFF.

dff_sync

type bit
 default value '0'

This generic represents whether the DFF reset is synchronous or asynchronous. '0' is for asynchronous and '1' for synchronous.

3.1.3 Ports

Ports	Direction	Type	Description
I[1:4]	input	std_logic	LUT inputs
CK	input	std_logic	Clock

L	input	std_logic	Load
R	input	std_logic	Reset
O	output	std_logic	Output

3.1.4 Example

This documentation only provides the instantiation of the component. For a full example, please refer to NX_FE example provided in the example/nxLibrary installation directory.

```

LUT_0 : NX_FE
generic map (
    lut_table => b"100000000000000000" -- O <= A & B & C & D
)
port map (
    I1 => A
    , I2 => B
    , I3 => C
    , I4 => D
    , O  => OUT
);

DFF_0 : NX_FE
generic map (
    lut_table => b"1111111111111110" -- O <= A | B | C | D
    , dff_mode  => '1'                -- sequential mode
    , dff_edge  => '0'                -- rising edge
    , dff_load  => '0'                -- always load
    , dff_init  => '1'                -- use connected reset net
    , dff_sync  => '1'                -- synchronous reset
    , dff_ctxt  => '0'                -- initial value is 0
)
port map (
    I1 => A
    , I2 => B
    , I3 => C
    , I4 => D
    , O  => OUT
    , CK => CLK
    , R  => RST
    , L  => OPEN
);

```

3.2 NX_CY

3.2.1 Description

The NX_CY component describes a 4-bits carry look ahead circuit combined with LUT and DFF as shown in **Erreur ! Source du renvoi introuvable..** The NX_CY is composed of 4 stages numbered from 1 to 4 where 1 represents the LSB.

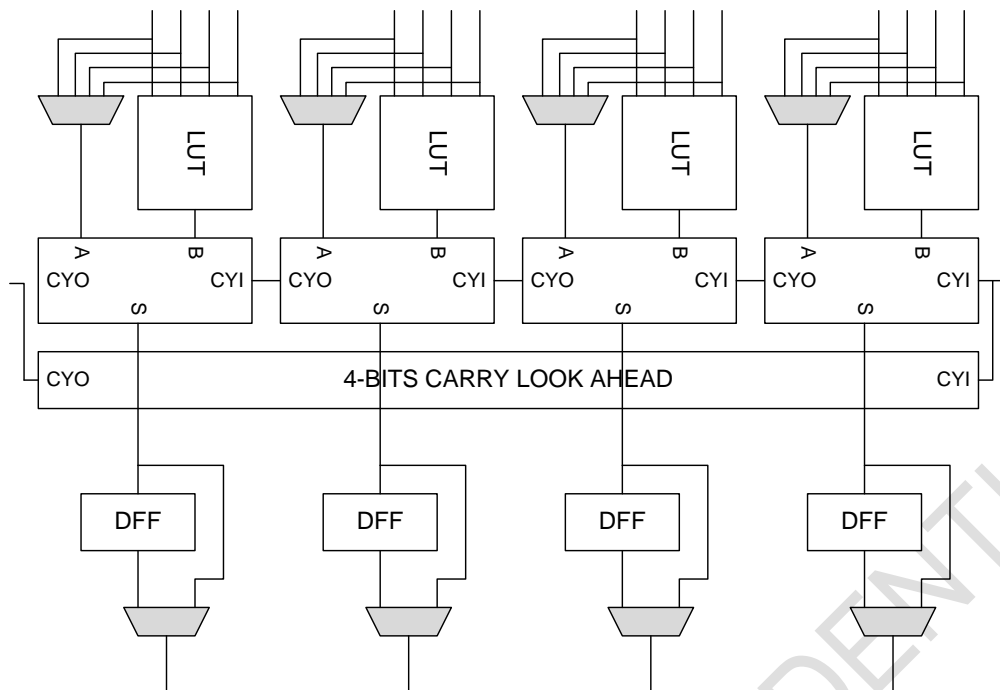


Figure 7: CARRY logic diagram

3.2.2 Generics

add_carry

type	integer
------	---------

default value 0

This generic represents the way the CI (carry in) port is connected: 0 is for low, 1 for high and 2 for propagate which means it is connected to the previous NX_CY CO (carry out) port.

add_ctxt[1-4]

```
type          std_logic
```

default value 'U'

This generic represents the initial value of the DFF of the corresponding stage [1-4]. The initial value is set by bitstream. The available values are: 'U' for undefined (no value set in bitstream), '0' for low and '1' for high.

add_dff[1-4]

type	bit
------	-----

default value '0'

This generic represents whether the output DFF of the corresponding stage [1-4] is bypassed ('0') or used ('1').

add_table[1-4]

```
type          bit_vector(15 downto 0)
```

default value b"000000000000000000"

This generic represents the truth table of the LUT associated to B input of corresponding stage [1-4].

dff_edge

type bit
default value '0'

This generic represents the front polarity of the clock of all sequential stages. '0' is for rising edge and '1' for falling edge. Please note that all sequential stages have the same front polarity.

dff_init

type bit
default value '0'

This generic represents whether the sequential stages consider the R (reset) input. '0' is for ignore and '1' for using connected net. Please note that this generic affect all sequential stages.

dff_load

type bit
default value '0'

This generic represents whether the sequential stages consider the L (load) input. '0' is for ignore and '1' for using connected net. Please note that this generic affect all sequential stages.

dff_sync

type bit
default value '0'

This generic represents whether the sequential stages reset is synchronous or asynchronous. '0' is for asynchronous and '1' for synchronous. Please note that this generic affect all sequential stages.

3.2.3 Ports

Ports	Direction	Type	Description
A[1:4]	input	std_logic	A input of each stage
B1I[1:4]	input	std_logic	LUT inputs for stage 1
B2I[1:4]	input	std_logic	LUT inputs for stage 2
B3I[1:4]	input	std_logic	LUT inputs for stage 3
B4I[1:4]	input	std_logic	LUT inputs for stage 4
CK[1:4]	input	std_logic	Clock input of each stage
L[1:4]	input	std_logic	Load input of each stage
R[1:4]	input	std_logic	Reset input of each stage
CIB	input	std_logic	Carry input
COB	output	std_logic	Carry output
O[1:4]	output	std_logic	Output of each stage

3.2.4 Example

This documentation only provides the instantiation of the component. For a full example, please refer to NX_CY example provided in the example/nxLibrary installation directory.

```
-- SUM[3:0] <= A[3:0] + "10"&B[1:0]

ADD_0 : NX_CY
generic map (
    add_carry    => 0                -- low
    , add_table1 => b"1100110011001100" -- O <= I2
    , add_table2 => b"1010101010101010" -- O <= I1
    , add_table3 => b"0000000000000000" -- O <= '0'
    , add_table4 => b"1111111111111111" -- O <= '1'
)
port map (
    A1  => A(0), B1I1 => OPEN, B1I2 => B(0), B1I3 => OPEN, B1I4 => OPEN
    , A2  => A(1), B2I1 => B(1), B2I2 => OPEN, B2I3 => OPEN, B2I4 => OPEN
    , A3  => A(2), B3I1 => OPEN, B3I2 => OPEN, B3I3 => OPEN, B3I4 => OPEN
    , A4  => A(3), B4I1 => OPEN, B4I2 => OPEN, B4I3 => OPEN, B4I4 => OPEN
    , CIB => OPEN, COB => OPEN
    , O1  => SUM(0), O2 => SUM(1), O3 => SUM(2), O4 => SUM(3)
);
```

3.3 NX_RFB

3.3.1 Description

The NX_RFB component describes a Register File Block circuit that is a Simple Dual Port memory of 64 words of 16-bit (one is port dedicated to write, the second port is dedicated to read). The circuit includes Error Code Correction (EDAC).

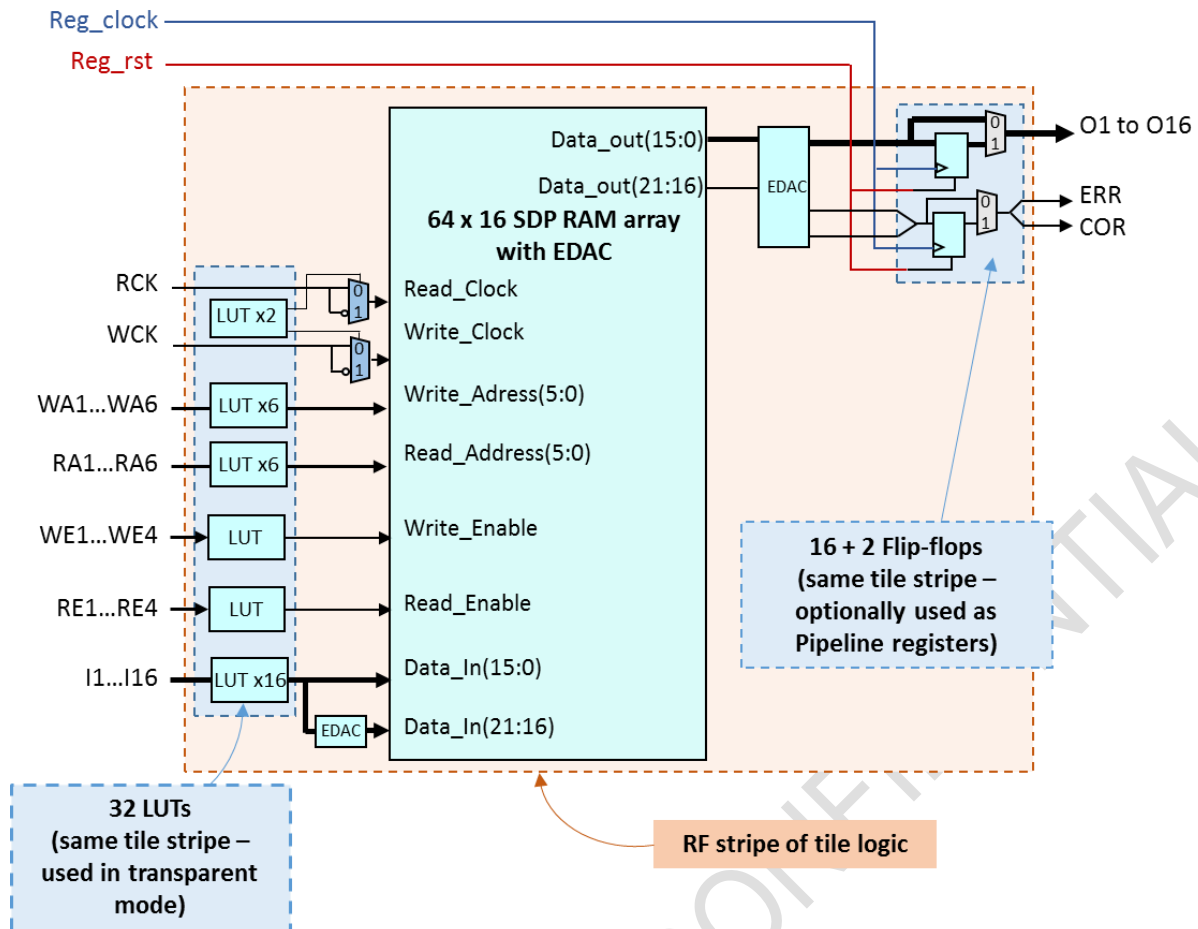


Figure 8: RFB diagram

The Register_File block is made of:

- 32 FEs
- One 64 x 16-bit Simple Dual Port RAM with user's transparent EDAC

Associated FE usage :

- If the Register_File is not used, all the 32 FEs are free to be used to implement user's logic
- If the Register_File is used, up to 30 inputs will reach the 64 x 16 RAM array by crossing LUTs, and 2 additional LUTs will be used for internal configuration purpose
 - o 1 LUT for Read_Enable (with potential 4-input decoder)
 - o 1 LUT for Write_Enable (with potential 4-input decoder)
 - o 6 LUTs for Write_Address
 - o 6 LUTs for Read_Address
 - o 16 LUTs for Data_In
 - o 2 additional LUTs for RF internal configuration

Using the whole RAM array (64 x 16) requires using 32 LUTs of the same tile section.

The LUTs can implement optional customer logic, for example

- o Implement simple decoding functions for Write_Enable or Read_Enable
 - o Address/Data multiplexers to implement time multiplexed two write ports and/or two read ports (not yet supported by NXmap)
- (see NXmap related notes for more details)

- If the Register_File is used and configured with optional output pipeline registers, those registers can be implemented with FE Flip-Flops of the same tile section (Up to 18 FFs, 16 for Data_out + 2 for ERR and COR outputs)
- If the Register_File is partially used (for example as 64 x 8 SDP RAM), the remaining 8 FEs will stay free to implement other unrelated logic functions)

NXmap support:

- The current version of NXmap supports the implementation of simple decoders on the Write_Enable and Read_Enable commands paths (the Register_File must be instantiated).
- LUTs are used as transparent for data inputs as well as read and write addresses (both inference and instantiation).

Future versions of NXmap will support higher flexibility such as multiplexers and other simple combinatorial function on the data and address input paths.

3.3.2 Generics

mem_context (1)

type string
default value ""

This generic represents the initial value of the RFB. The initial value is set by bitstream. The string contains a list of all complete bit words separated by coma.

When a word size is less than 16 bits or when number of words is less than 64, an error occurs.

When a word size exceeds 16 bits or when the number of words exceeds 64, an error occurs.

ren_table (2)

type bit_vector(15 downto 0)
default value b"0000000000000000"

This generic represents the truth table of the LUT associated to RE input.

rck_edge (3)

type bit
default value '0'

This generic represents the front polarity of the RCK clock. '0' is for rising edge and '1' for falling edge.

wen_table (4)

type bit_vector(15 downto 0)
default value b"0000000000000000"

This generic represents the truth table of the LUT associated to WE input.

wck_edge (3)

type bit
default value '0'

This generic represents the front polarity of the WCK clock. '0' is for rising edge and '1' for falling edge.

3.3.3 Ports

Ports	Direction	Type	Description
RCK	input	std_logic	Read clock
WCK	input	std_logic	Write clock
I[1:16]	input	std_logic	Data input
COR	output	std_logic	Correction output flag
ERR	output	std_logic	Error output
O1 to O16	output	std_logic	Data output
RA1 to RA6	input	std_logic	Read address
RE1 to RE4	input	std_logic	Read enable
WA1 to WA6	input	std_logic	Write address
WE1 to WE4	input	std_logic	Write enable

3.3.4 Instantiation Example

This documentation only provides the instantiation of the component. For a full example, please refer to NX_RFB example provided in the example/nxLibrary installation directory.

```
-- RFB 64 words of 16 bits

RFB_0 : NX_RFB
generic map (
    ren_table => b"1111111111111111" - O <= '1'
    , wen_table => b"1010101010101010" - O <= I1
    , mem_context => "1111111111111111,0011001100110011," &
                    "1100110011001100,1111111111111111," &
                    "... "
    -- other 64 words must be also initialized
)
port map (
    RCK => CLK, WCK => CLK
    , I1 => DI(0), ... , I16 => DI(15)
    , COR => COR, ERR => ERR
    , O1 => DO(0), ... , O16 => DO(15)
    , RA1 => RA(0), ... , RA6 => RA(5)
    , WA1 => WA(0), ... , WA6 => WA(5)
    , RE1 => OPEN, ... , RE4 => OPEN
    , WE1 => WE
    , WE2 => OPEN, ... , WE4 => OPEN
)
```

3.4 NX_DSP

3.4.1 Description

The NX_DSP component describes a Digital Signal Processor circuit that allows implementation of arithmetic computations such as multiply, add/subtract.

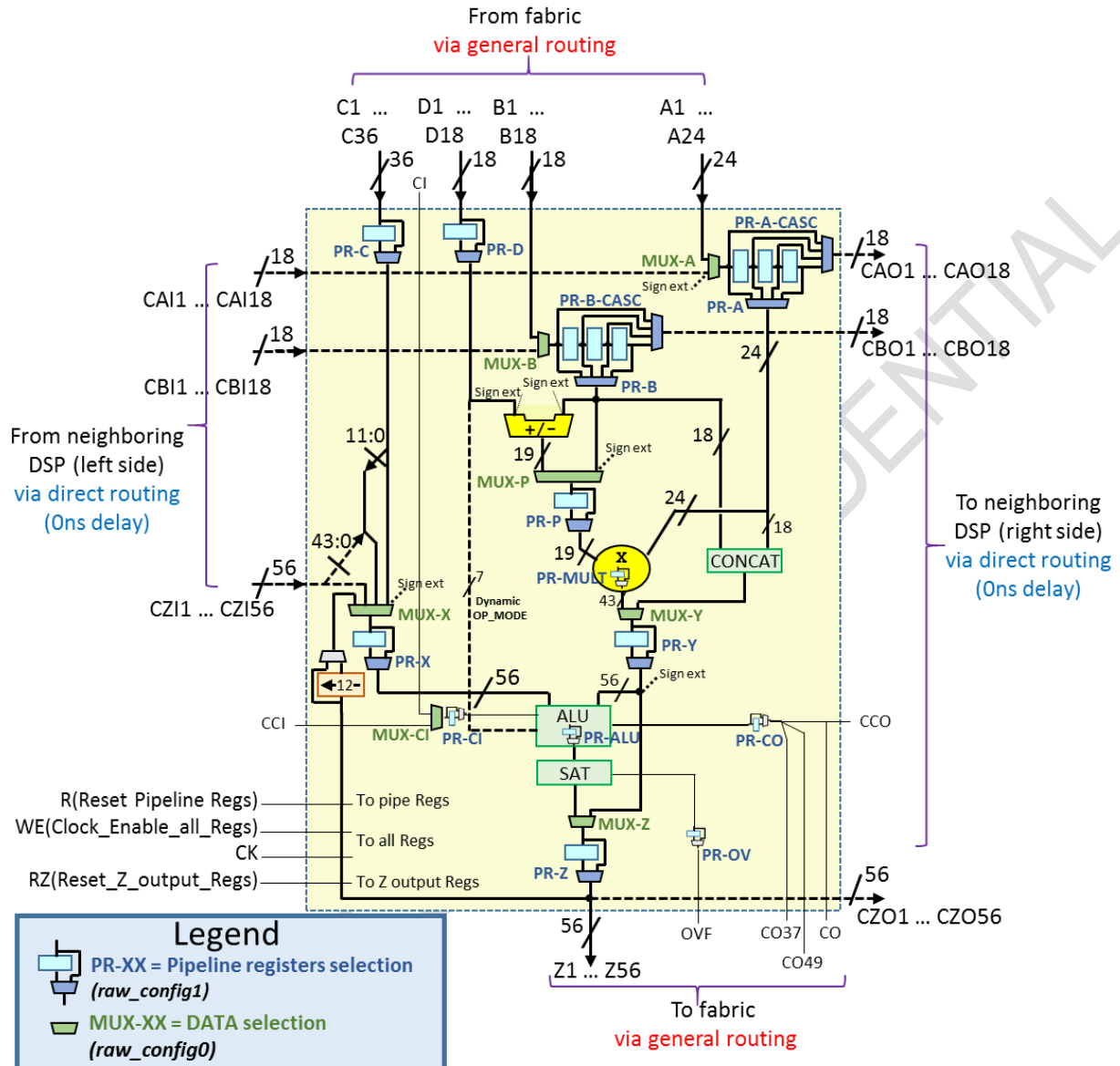


Figure 9: DSP simplified block diagram

3.4.2 Generics

std_mode

type string

default value ""

This generic represents the predefined operating mode of the DSP. When empty the operating mode is defined by the 4 raw_config generics.

The available predefined modes are:

- "ADD_36" → 36 bits addition
- "SUB_36" → 36 bits subtraction
- "SMUL_18" → 18 bits signed multiplication
- "UMUL_18" → 18 bits unsigned multiplication
- "SMUL_EXT" → extension for 24 bits signed multiplication
- "UMUL_EXT" → extension for 24 bits unsigned multiplication

When using one of these predefined modes, the 4 raw_config generics are defined as follow:

- raw_config0
 - ADD_36 b"00000000000010000000"
 - SUB_36 b"00000000000010000000"
 - SMUL_18 b"00000000001000000001"
 - UMUL_18 b"00000000001000000000"
 - SMUL_EXT b"00000000000001100001"
 - UMUL_EXT b"00000000000001100000"
- raw_config1
 - All modes b"00000000000000000000"
- raw_config2
 - All modes b"0000000000000000"
- raw_config3
 - ADD_36 b"0000001"
 - SUB_36 b"0001011 "
 - SMUL_18 b"0100000"
 - UMUL_18 b"0100000"
 - SMUL_EXT b"0000000"
 - UMUL_EXT b"0000000"

raw_config0

type bit_vector(19 downto 0)

default value b"00000000000000000000"

This generic configures the following fields:

Name	Index	Description
CO_SEL	19	Carry out MUX for CO and CCO outputs '0': Select CO37 '1': Select CO49
ALU_DYNAMIC_OP	18	ALU Dynamic Operation '0': use raw_config3 as ALU operation '1': use D1...D6 as ALU operation
SATURATION_RANK	17:12	MSB position for saturation and overflow Signed : "100000" for range -2^{31} to $(2^{31})-1$ Unsigned : "100000" for range 0 to $(2^{32})-1$ Max value = "110111" (55)
ENABLE_SATURATION	11	'0': disable, '1': enable
Z_FEEDBACK_SHL12	10	Shift of the Z output for feedback '0': No shift '1': 12-bit left shift
MUX_Z	9	Selection for Z output '0': ALU '1': PR_Y
MUX_CI	8	Carry in MUX '0': CI input '1': CCI cascade input
MUX_Y	7	Y operand MUX '0': MULT '1': Concat (B, A)
MUX_X	6:5	X operand MUX "00": C (sign extended to 56-bit) "01": CZI "11": CZI(43:0] & C(11:0] "10": Z (12-bit left shifted or not)
MUX_P	4	Pre-adder/ B MUX (to multiplier) '0': B (sign extended) '1': Pre-adder
MUX_B	3	B input MUX '0': Select B input port '1': Select CBI input
MUX_A	2	A input MUX. '0': Select A input port '1': Select CAI input
PRE_ADDER_OP	1	Pre-adder operation '0': add '1': subtract)
SIGNED_MODE	0	'0': unsigned, '1': signed

raw_config1

type bit_vector(21 downto 0)

default value b"0000000000000000000000"

This generic configures the following fields:

Name	Index	Description
RESERVED	21:19	"000"
PR_OV_MUX	18	ALU overflow pipe register '0' : no pipeline '1' : pipeline
PR_CO_MUX	17	Carry out pipe depth '0' : no pipeline '1' : pipeline
PR_Z_MUX	16	Z output pipe depth '0' : no pipeline '1' : pipeline
PR_ALU_MUX	15	ALU out pipe depth '0' : no pipeline '1' : pipeline
PR_MUL_MUX	14	Multiplier out pipe depth '0' : no pipeline '1' : pipeline
PR_Y_MUX	13	Y operand pipe depth '0' : no pipeline '1' : pipeline
PR_X_MUX	12	X operand pipe depth '0' : no pipeline '1' : pipeline
PR_P_MUX	11	Pre-adder pipe depth '0' : no pipeline '1' : pipeline
PR_CI_MUX	10	Carry in pipe depth '0' : no pipeline '1' : pipeline
PR_D_MUX	9	D input pipe depth '0' : no pipeline '1' : pipeline
PR_C_MUX	8	C input pipe depth '0' : no pipeline '1' : pipeline
PR_B_CAS_MUX	7:6	Cascaded B input pipe depth B input pipe depth
PR_B_MUX	5:4	Cascaded A input pipe depth A input pipe depth
PR_A_CAS_MUX	3:2	"00" : no pipeline "01" : 1 level pipeline register
PR_A_MUX	1:0	"10" : 2 levels pipeline "11" : 3 levels pipeline

raw_config2

type bit_vector(12 downto 0)

default value b"00000000000000"

This generic configures the following fields:

Name	Index	Description
PR_OV_RST	12	ALU overflow pipe reset ('0' : disable, '1' : enable)
PR_CYO_RST	11	Carry out pipe reset ('0' : disable, '1' : enable)
PR_Z_RST	10	Z output pipe reset ('0' : disable, '1' : enable)
PR_ALU_RST	9	ALU out pipe reset ('0' : disable, '1' : enable)
PR_MUL_RST	8	Multiplier out pipe reset ('0' : disable, '1' : enable)
PR_Y_RST	7	Y operand pipe reset ('0' : disable, '1' : enable)
PR_X_RST	6	X operand pipe reset ('0' : disable, '1' : enable)
PR_P_RST	5	Pre-adder pipe reset ('0' : disable, '1' : enable)
PR_CYL_RST	4	Carry in pipe reset ('0' : disable, '1' : enable)
PR_D_RST	3	D input pipe reset ('0' : disable, '1' : enable)
PR_C_RST	2	C input pipe reset ('0' : disable, '1' : enable)
PR_B_RST	1	B input pipe reset ('0' : disable, '1' : enable)
PR_A_RST	0	A input pipe reset ('0' : disable, '1' : enable)

raw_config3

type bit_vector(6 downto 0)

default value b"00000000"

This generic configures the following fields:

Name	Index	Description
ALU_MUX	6	Swap ALU operand ('0' : no swap, '1' : swap)
ALU_OP	5:0	ALU operation (16 valid values over 64 possible combinations)

ALU operation must be set based on the following table:

Operation	Opcode	Equation
Arithmetic operation		
ADD	b"000000"	$Z = A + B$
ADDC	b"000001"	$Z = A + B + CI$
SUB	b"001010"	$Z = A - B$
SUBC	b"001011"	$Z = A - B - CI$
INCA	b"000101"	$Z = A + CI$

DECA	b"000111"	$Z = A - CI$
Logic operation		
A	b"100000"	$Z = A$
notA	b"110000"	$Z = \sim A$
AND	b"100001"	$Z = A \& B$
ANDnotB	b"101001"	$Z = A \& \sim B$
NAND	b"110001"	$Z = \sim(A \& B)$
OR	b"100010"	$Z = A B$
ORnotB	b"101010"	$Z = A \sim B$
NOR	b"110010"	$Z = \sim(A B)$
XOR	b"100011"	$Z = A \wedge B$
XNOR	b"110011"	$Z = \sim(A \wedge B)$
INVALID OP	48 other possible values	$Z = \text{XXXXXXXXXXXXXXXX}$

3.4.3 Ports

Ports	Direction	Type	Description
A1 to A24	input	std_logic	24-bit A input
B1 to B18	input	std_logic	18-bit B input
C1 to C36	input	std_logic	36-bit C input
CA11 to CA18	input	std_logic	18-bit Cascaded A input
CAO1 to CAO18	output	std_logic	18-bit Cascaded A output
CBI1 to CBI18	input	std_logic	18-bit Cascaded B input
CBO1 to CBO18	output	std_logic	18-bit Cascaded B output
CCI	input	std_logic	Cascaded Carry input
CCO	output	std_logic	Cascaded Carry output
CI	input	std_logic	Carry input
CK	input	std_logic	Clock (works on rising edges)
CO	output	std_logic	Carry output
CO37	output	std_logic	Carry output bit 37
CO49	output	std_logic	Carry output bit 49
CZ11 to CZ156	input	std_logic	56-bit Cascaded Z input
CZO1 to CZO56	output	std_logic	56-bit Cascaded Z output
D1 to D18	input	std_logic	18-bit D input
OVF	output	std_logic	Overflow output flag

R	input	std_logic	Reset for pipeline registers except Z output register (active high)
RZ	input	std_logic	Reset for Z output register only(active high)
WE	input	std_logic	Write enable: '0': all DSP internal registers are frozen, '1': normal operation
Z1 to Z56	output	std_logic	56-bit Z output

3.4.4 Instantiation Example

This documentation only provides the instantiation of the component. For a full example, please refer to NX_DSP example provided in the example/nxLibrary installation directory.

```
-- MUL(47:0) <= A(23:0) * B(23:0)      unsigned

signal link : std_logic_vector(35 downto 0)

DSP_0 : NX_DSP
generic map (
    std_mode => "UMUL_18"
)
port map (
    A1    => A(0)      , ... , A24    => A(23)
    , B1    => B(12)     , ... , B12    => B(23)
    , B13    => OPEN     , ... , B18    => OPEN
    , C1    => OPEN     , ... , C36    => OPEN
    , D1    => OPEN     , ... , D18    => OPEN
    , Z1    => OPEN     , ... , Z56    => OPEN
    , CAI1   => OPEN    , ... , CAI18  => OPEN
    , CAO1   => OPEN    , ... , CAO18  => OPEN
    , CBI1   => OPEN    , ... , CBI18  => OPEN
    , CBO1   => OPEN    , ... , CBO18  => OPEN
    , CZI1   => OPEN    , ... , CZO56  => OPEN
    , CZO1   => link(0) , ... , CZO36  => link(35)
    , CZO37  => OPEN    , ... , CZO56  => OPEN
    , CCI    => OPEN    , CCO => OPEN, CK    => OPEN
    , CI     => OPEN    , CO  => OPEN, CO37  => OPEN, CO49  => OPEN
    , OVF    => OPEN    , R   => OPEN, RZ    => OPEN, WE    => OPEN
);

DSP_1 : NX_DSP
generic map (
    std_mode => "UMUL_EXT"
)
port map (
    A1    => A(0)      , ... , A24    => A(23)
    , B1    => B(0)     , ... , B12    => B(11)
    , B13    => OPEN     , ... , B18    => OPEN
    , C1    => OPEN     , ... , C36    => OPEN
    , D1    => OPEN     , ... , D18    => OPEN
    , Z1    => MUL(0)   , ... , Z48    => MUL(47)
    , Z49    => OPEN     , ... , Z56    => OPEN
    , CAI1   => OPEN    , ... , CAI18  => OPEN
    , CAO1   => OPEN    , ... , CAO18  => OPEN
    , CBI1   => OPEN    , ... , CBI18  => OPEN
```

```
, CBO1 => OPEN , ... , CBO18 => OPEN
, CZI1 => link(0), ... , CZO36 => link(35)
, CZI36 => OPEN , ... , CZI56 => OPEN
, CZO1  => OPEN , ... , CZO56 => OPEN
, CCI   => OPEN , CCO => OPEN, CK    => OPEN
, CI    => OPEN , CO  => OPEN, CO37 => OPEN, CO49 => OPEN
, OVF   => OPEN , R   => OPEN, RZ    => OPEN, WE    => OPEN
);
```

3.4.1 Simulation

The NX_DSP VHDL simulation model is included in the NxLibrary (NxPackage). It allows to simulate any one of the possible NX_DSP configurations.

3.5 NX_DSP_SPLIT

The NX_DSP_SPLIT is an alternate primitive for using DSP blocks. It can be instantiated as many times as required in your design.

For user's convenience, the generics are split, and can be modified separately. The input and output busses are grouped.

The following is the declaration of the component NX_DSP_SPLIT_GENERIC, included in the NxPackage.vhd.

```
component DSP_SPLIT
```

```
generic (
```

```
-- Generic declaration to define the "raw_config0" (cfg_mode). Defines :
```

```
-----
SIGNED_MODE : bit           := '1';
PRE_ADDER_OP : bit         := '0'; -- '0' = Addition, '1' = Subtraction
MUX_A : bit                := '0'; -- '0' = A input, '1' = CAI input
MUX_B : bit                := '0'; -- '0' = B input, '1' = CBI input
MUX_P : bit                := '0'; -- '0' for PRE_ADDER, '0' for B input
MUX_X : bit_vector(1 downto 0) := "01"; -- Select X operand "00" = C,
                                         -- "01" = CZI,
                                         -- "10" Select Z feedback
                                         -- "11" = SHFT(CZI) & C(11:0),
MUX_Y : bit                := '0'; -- '0' Select MULT output, '1' for (B & A)
MUX_CI : bit               := '0'; -- Select fabric input (not cascade)
MUX_Z : bit                := '0'; -- Select ALU output (not ALU input operand coming from PR_Y

Z_FEEDBACK_SHL12 : bit     := '0'; -- '0' for No shift, '1' for 12-bit left shift
ENABLE_SATURATION : bit    := '0'; -- '0' for Disable, '1' for Enable
SATURATION_RANK : bit_vector(5 downto 0) := "110110"; -- Weight of useful MSB on Z and CZO result
                                                         -- (to define saturation and overflow)

ALU_DYNAMIC_OP : bit       := '0'; -- '0' for Static, '1' for Dynamic
                                         -- D(5:0) are used for dynamic operation
CO_SEL : bit               := '0'; -- '0' for CO = ALU(36), '1' for CO = ALU(48)
-----
```

```
-- Generic declaration to define the "raw_config1" (cfg_pipe_mux)
```

```
-----
PR_A_MUX : bit_vector(1 downto 0) := "01"; -- Number of pipe reg levels on A input
PR_A_CASCADE_MUX : bit_vector(1 downto 0) := "10"; -- Number of pipe reg levels for CAO output
PR_B_MUX : bit_vector(1 downto 0) := "01"; -- Number of pipe reg levels on B input
PR_B_CASCADE_MUX : bit_vector(1 downto 0) := "10"; -- Number of pipe reg levels for CAO output

PR_C_MUX : bit := '0'; -- '0' for No pipe reg, '1' for 1 pipe reg
PR_D_MUX : bit := '1'; -- '0' for No pipe reg, '1' for 1 pipe reg
PR_CI_MUX : bit := '1'; -- '0' for No pipe reg, '1' for 1 pipe reg
PR_P_MUX : bit := '1'; -- '0' for No pipe reg, '1' for 1 pipe reg (Pre-adder)
PR_X_MUX : bit := '0'; -- '0' for No pipe reg, '1' for 1 pipe reg
-----
```

```

PR_Y_MUX : bit      := '1';      -- '0' for No pipe reg, '1' for 1 pipe reg
PR_MULT_MUX : bit   := '1';      -- No pipe reg -- Register inside MULT
PR_ALU_MUX : bit    := '0';      -- No pipe reg -- Register inside ALU
PR_Z_MUX : bit      := '1';      -- Registered output
PR_CO_MUX : bit     := '0';      -- '0' for No pipe reg, '1' for 1 pipe reg
PR_OV_MUX : bit     := '0';      -- '0' for No pipe reg, '1' for 1 pipe reg

-----
-- Generic declaration to define the "raw_config2" (cfg_pipe_rst)
-----
ENABLE_PR_A_RST : bit      := '1';  -- '0' for Disable, '1' for Enable
ENABLE_PR_B_RST : bit      := '1';  -- '0' for Disable, '1' for Enable
ENABLE_PR_C_RST : bit      := '1';  -- '0' for Disable, '1' for Enable
ENABLE_PR_D_RST : bit      := '1';  -- '0' for Disable, '1' for Enable
ENABLE_PR_CI_RST : bit     := '1';  -- '0' for Disable, '1' for Enable
ENABLE_PR_P_RST : bit      := '1';  -- '0' for Disable, '1' for Enable
ENABLE_PR_X_RST : bit      := '1';  -- '0' for Disable, '1' for Enable
ENABLE_PR_Y_RST : bit      := '1';  -- '0' for Disable, '1' for Enable
ENABLE_PR_MULT_RST : bit   := '1';  -- '0' for Disable, '1' for Enable
ENABLE_PR_ALU_RST : bit    := '1';  -- '0' for Disable, '1' for Enable
ENABLE_PR_Z_RST : bit      := '1';  -- '0' for Disable, '1' for Enable
ENABLE_PR_CO_RST : bit     := '1';  -- '0' for Disable, '1' for Enable
ENABLE_PR_OV_RST : bit     := '1';  -- '0' for Disable, '1' for Enable

-----
-- Constants declaration to define the "cfg_pipe_rst" -- raw_config3(6 downto 0)
-----
ALU_OP : bit_vector(5 downto 0) := "000000"; -- Addition = "000000", Subtract = "001010"
ALU_MUX : bit                  := '0' -- '0' for Don't swap ALU operands, '1' for ALU Swap operands
);
port(
  CK : IN  std_logic;
  R : IN  std_logic;
  RZ : IN  std_logic;
  WE : IN  std_logic;

  CI : IN  std_logic;
  A : IN  std_logic_vector(23 downto 0);
  B : IN  std_logic_vector(17 downto 0);
  C : IN  std_logic_vector(35 downto 0);
  D : IN  std_logic_vector(17 downto 0);

```

```
CAI : IN  std_logic_vector(17 downto 0);
CBI : IN  std_logic_vector(17 downto 0);
CZI : IN  std_logic_vector(55 downto 0);
CCI : IN  std_logic;

Z : out  std_logic_vector(55 downto 0);
CO : OUT  std_logic;
CO36 : OUT  std_logic;
CO48 : OUT  std_logic;
OVF : OUT  std_logic;
CAO : OUT  std_logic_vector(17 downto 0);
CBO : OUT  std_logic_vector(17 downto 0);
CZO : OUT  std_logic_vector(55 downto 0);
CCO : OUT  std_logic
);
end component;
```

3.6 NX_ECC

3.6.1 Description

The NX_ECC component describes an Error Code Correction circuit that can be used with memory declaration to add error correction support.

The NX_ECC can be instantiated with inferred memory blocks. The user must connect the LSB of the output data to the CHK input, and then use the COR and ERR flags.

3.6.2 Ports

Ports	Direction	Type	Description
CKD	input	std_logic	Input clock
CHK	input	std_logic	Check link This pin must be connected to the LSB of the output memory block – for each port requiring the ECC function.
COR	output	std_logic	One error found and corrected
ERR	output	std_logic	Errors found and not corrected

3.6.3 Instantiation Example

This documentation only provides the instantiation of the component. For a full example, please refer to NX_ECC example provided in the example/nxLibrary installation directory.

```
entity hdpecc_4Kx32 is
port (
    ckw : in std_logic;
    ckr : in std_logic;
    ckq : in std_logic;
    we : in std_logic;
    adw : in std_logic_vector (11 downto 0);
    adr : in std_logic_vector (11 downto 0);
    di : in std_logic_vector (31 downto 0);
    do : out std_logic_vector (31 downto 0);
    cor : out std_logic;
    err : out std_logic
);
end entity;

architecture rtl of hdpecc_4Kx32 is
    type mem_reg is array (4095 downto 0) of std_logic_vector(31 downto 0);
    signal mem : mem_reg;

begin
    hdpram_ecc: NX_ECC
    port map (
        CKD => ckq
        , CHK => do(0)
    );
end architecture;
```

```
        , COR => cor  
        , ERR => err  
    );  
    ...
```

NANOXPLORE CONFIDENTIAL

3.7 NX_RAM

3.7.1 Description

The NX_RAM component describes a synchronous True Dual Port Random Access Memory circuit of 48 Kbits. The circuit supports Error Code Correction (ECC, also called EDAC – Error Detection and Correction).

The 48K-bit memory array can be simultaneously read or written by two access ports (A and B).

When used without EDAC, the external RAM block configuration can be set independently for the two access ports. As an example, the port A can be configured for 48Kx1, while the port B can be organized as 4Kx12.

Data inputs, addresses, control signals, clock inputs and data outputs are independent for each ports. The clocks can be synchronous or asynchronous.

However, simultaneous write access on both ports at the same physical address, or write access simultaneous with a read access at the same physical address are not allowed.

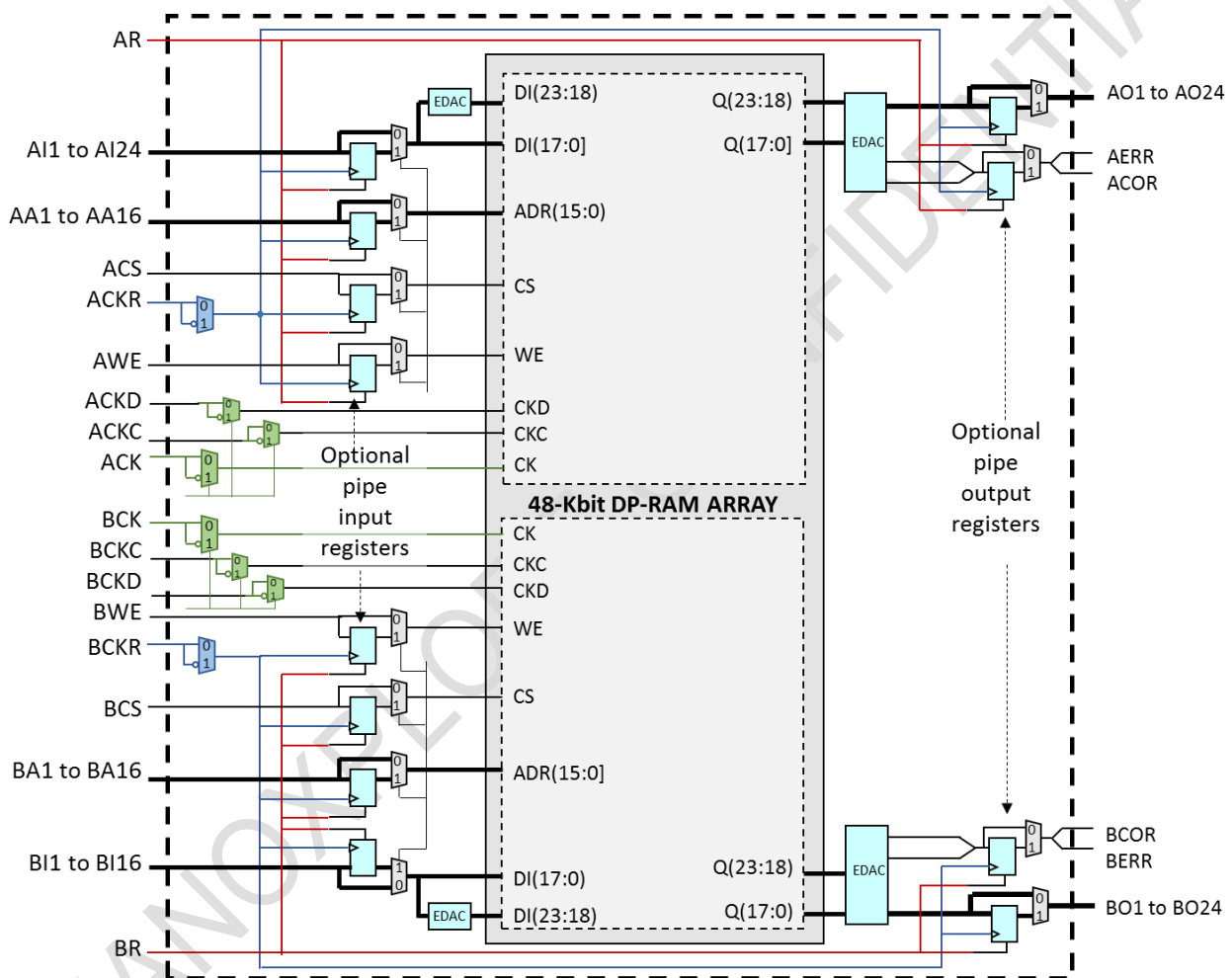


Figure 10: RAM diagram

3.7.2 Memory ports configurations

3.7.2.1 Optional input and output behavior and pipeline registers:

By default, the RAM block do not use pipeline registers. The output delivers a valid data $T_{\text{access_time}}$ after the clock edge that samples the read address (ACS = '1' and AWE = '0') or BCS = '1' and BWE = '0').

Outputs behavior during write :

During write cycles (ACS = '1' and AWE = '1') or (BCS = '1' and BWE = '1'), the RAM block output remains with the anterior value (NO_CHANGE mode)

In addition, reading from one port while simultaneously writing to the other port at the same memory location is not allowed.

During write cycles (ACS = '1' and AWE = '1') or (BCS = '1' and BWE = '1'), the RAM block output remains with the anterior value.

However, to improve the design performance (in terms of clock frequency), the user can optionally insert two levels of pipeline registers.

- The output pipeline allows to support higher frequencies, and reduces the apparent memory access time, at the cost of one clock cycle delay.
- The input pipeline register level also improves the supported frequency, and reduces the apparent memory setup delay, at the cost of one additional clock cycle delay.
- The optional input and output registers can be synchronously reset by activating the AR (A port) and/or BR (B port) inputs (active high).

In addition, the polarity of the block RAM clock as well as the one of the register clocks can be modified by the user (see NX_RAM raw_config0 in the Library Guide).

3.7.2.2 No ECC modes

The NO_ECC configuration mode is set by generics (raw_config1(15:12) = "0000". See the Library Guide for more detailed information.

The memory is internally organized as a 2K x 24-bit array. The memory is True Dual Port. It can be simultaneously access by 2 ports (respectively called port A and port B).

Each port can access the array in several formats. Each port can have an independent configuration (address and data width), with independent data input, addresses, control signals, data output and clock. The two clocks can be synchronous or asynchronous.

The possible configuration ratios on each port are:

- o 2K x 24
- o 4K x 12
- o 6K x 8
- o 12K x 4
- o 24K x 2
- o 48K x 1

The input data width and output data width for both ports A and B can be set by setting some generic values if the RAM block is instantiated (raw_config1(11:0)). More details in the Library Guide.

The figure 8 shows the physical memory organization and the data/address lines to be used to access the array contents (output data is shown for port A only).

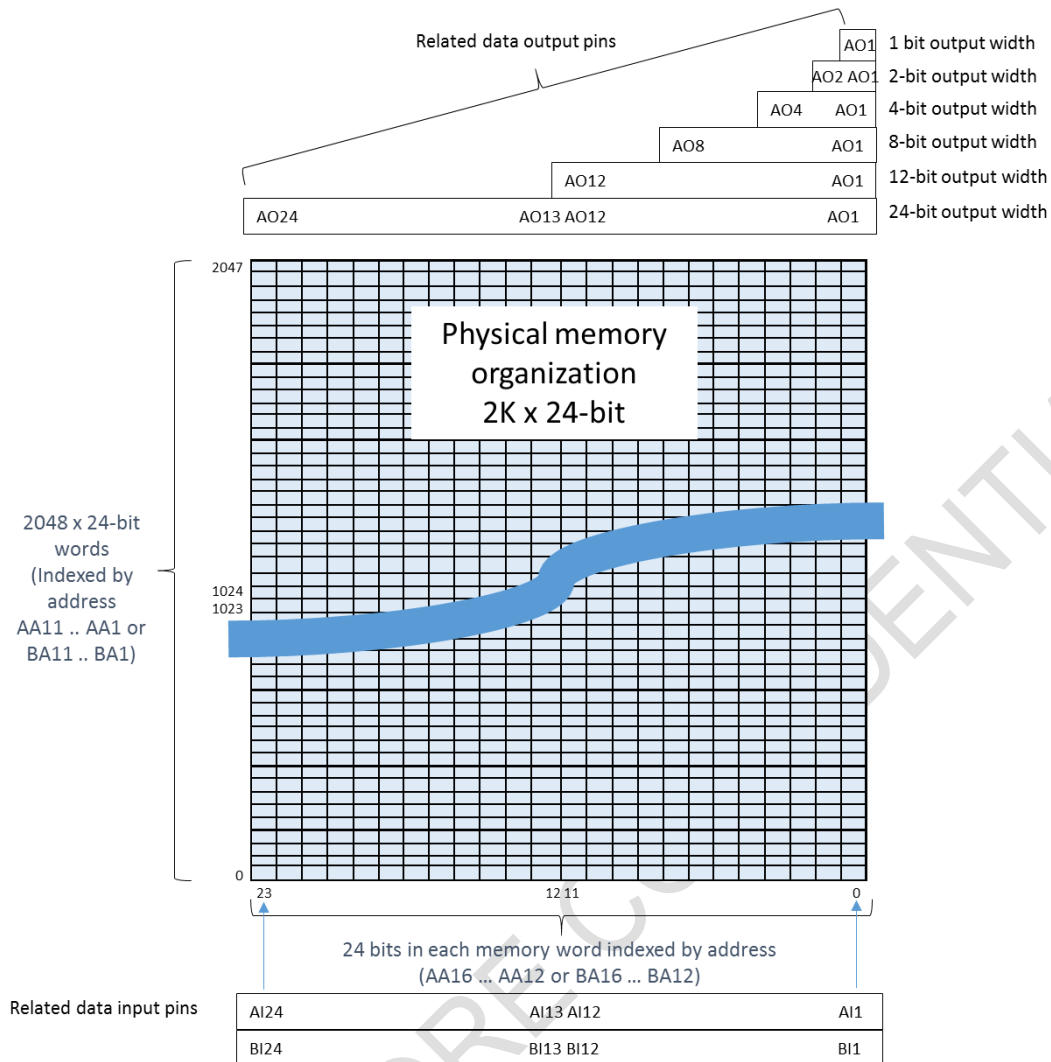


Figure 11: RAM organization (No ECC)

- 2K x 24 :

Internally, the RAM blocks are physically organized as 2K x 24-bit array. The addresses AA11 .. AA1 (or BA11 .. BA1) are used to access the 24-bit data. AI24 .. AI1 (or BI24 .. BI1) data input lines are used for write operations. AO24 .. AO1 (or BO24 .. BO1) are used for data read.

- 4K x 12 and other organizations : 6K x 8, 12K x 4, 24K x 2 and 48K x 1 :

When organized as 4K x 12, the addresses AA11 .. AA1 (or BA11 .. BA1) are used to access the 24-bit data word, an additional address bit (AA12 or BA12) is used to index the lower or higher 12-bit sub words. In addition, during write the data inputs AI12 .. AI1 (or BI12 .. BI1) are used to write the lower 12 bits, and AI24 .. AI13 (or BI24 .. BI13) are used to write the higher 12 bits. For reading, AO12 .. AO1 (or BO12 .. BO1) are used to read both lower and higher 12 bits.

For data write, the data input bus must be replicated one or more times on the RAM block input data pins. The following figure shows a summary for the 6 possible configurations. Only port A is shown. The rules are obviously the same for port B.

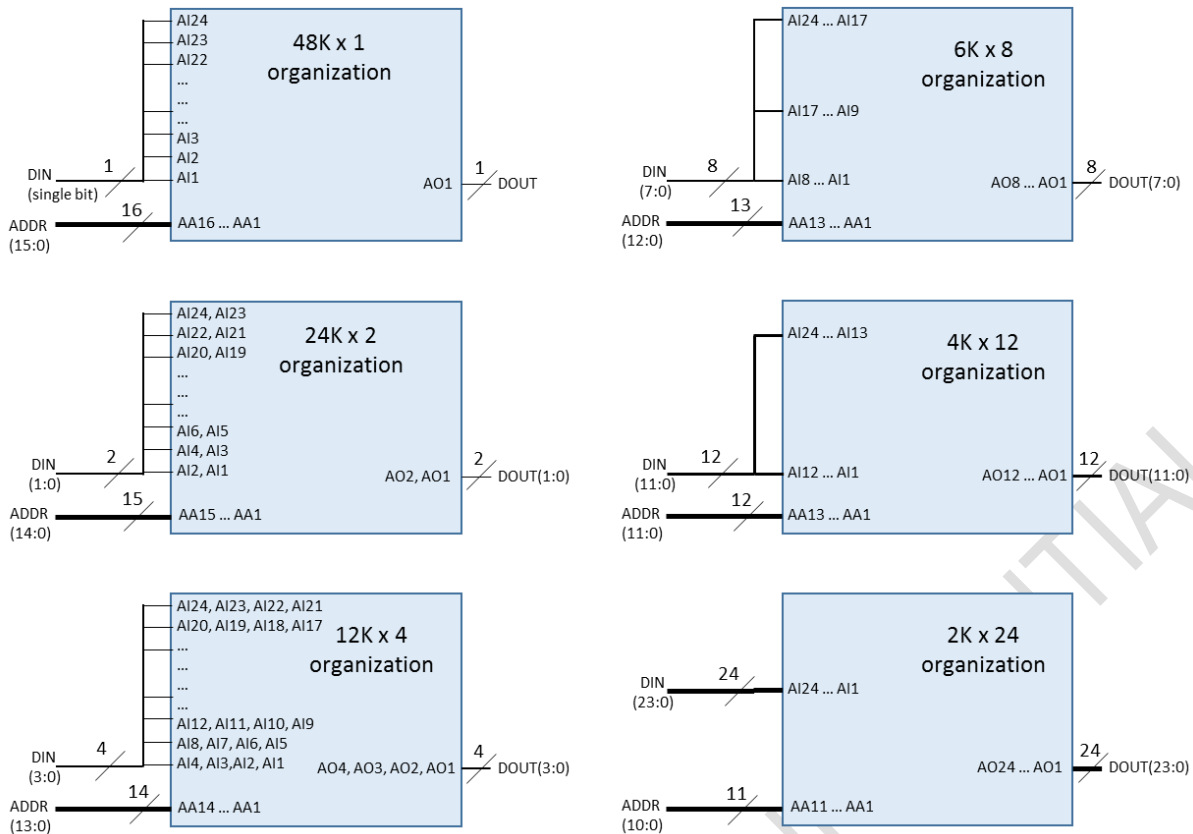


Figure 12: Address and data connections (No ECC)

3.7.2.3 ECC modes

When used with ECC, the user array size is restricted to 2K x 18. The 6 remaining bits of each internal address of 24-bit words are used to store the ECC signature of each 18-bit data.

During the write cycles, the ECC encoder generates a 6-bit signature for each 18-bit data to be written. The resulting 24-bit words is then stored into the specified address.

During read cycles, the ECC decoder can detect and correct any single bit error, or detect any double bit error.

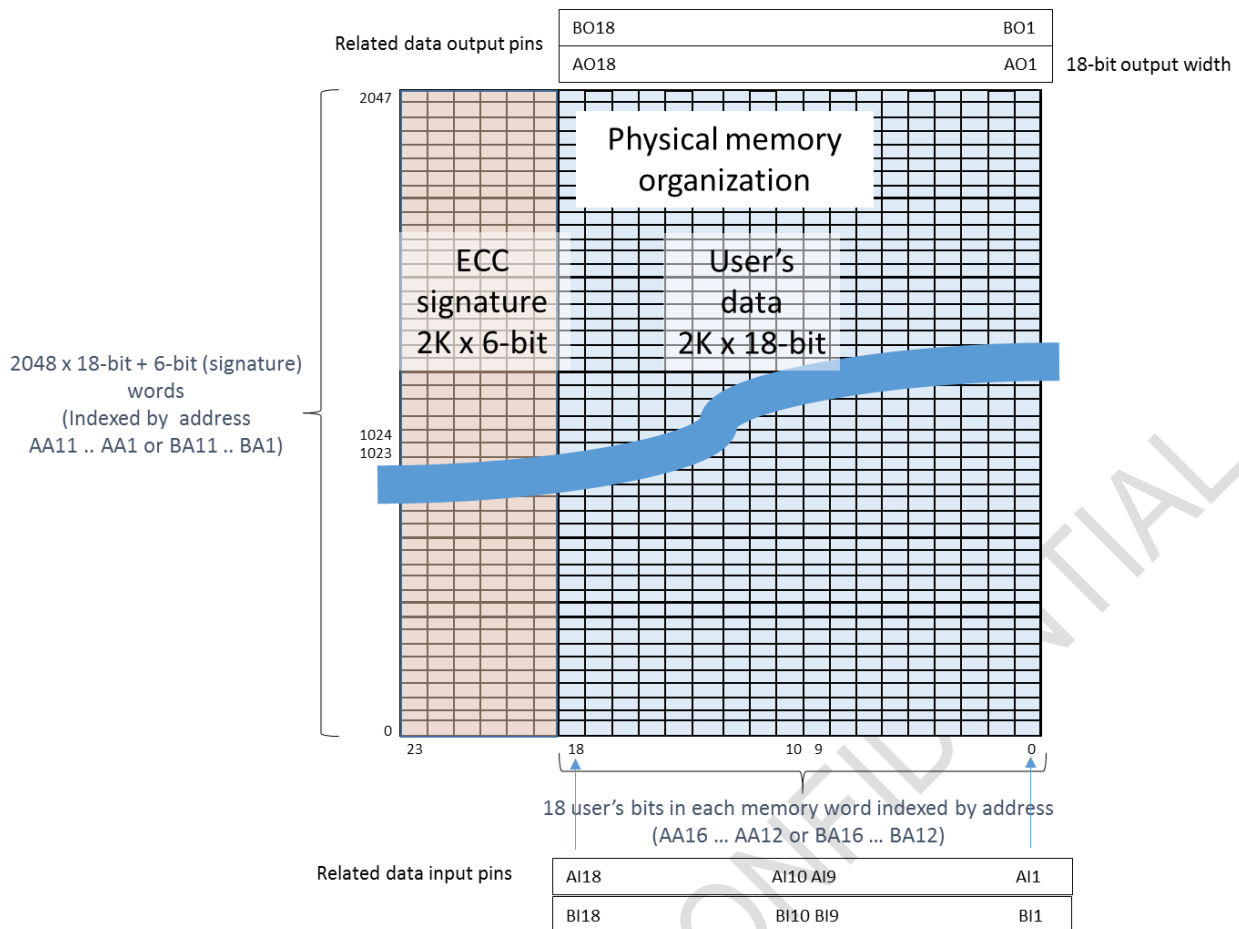


Figure 13: RAM organization (ECC FAST or SLOW)

The physical connections of address and input/output data lines is shown in the next figure.

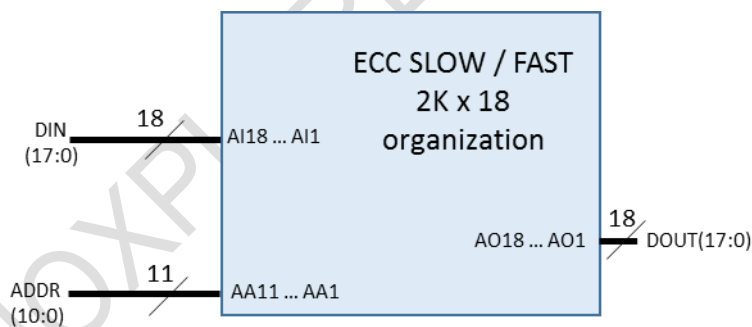


Figure 14: Address and data connections (ECC FAST or SLOW)

3.7.2.4 ECC data correction in FAST mode

- If a single bit error is found, it will be automatically detected and **corrected at the RAM output** port. The flags ACOR or BCOR are set during the read cycle to signal the error detection and correction. However, the internal memory array remains corrupt.
- If a double bit error is detected, it can't be corrected, and the flags AERR or BERR are asserted.

3.7.2.5 ECC data correction in SLOW mode

- This mode is also called Read Repair Mode (RRM).
- If a single bit error is found, it will be automatically detected and corrected at the RAM output port, and the **memory content is automatically updated with the corrected value**. The flags ACOR or BCOR are set during the user's read cycle to signal the error detection and correction.
- If a double bit error is detected, it can't be corrected, but the flags AERR or BERR are asserted.
- In order to correct a possible error during a read access, the read cycle becomes a read modify write, where the write half cycle is transparent to the user. For this, the NG-MEDIUM RAM blocks use a doubled internal frequency. This internal clock is generated by using an exclusive OR, between the main clock (CKA or CKB) and the 90° shifted clock (ACKD or BCKD) required to support the ECC SLOW mode.
- Using ACKD and/or BCKD in ECC SLOW mode is mandatory. It must be a 90° phase shifted version of the main clock input (ACK and/or BCK). ACKD and BCKD can each be generated with ACK and BCK by using PLL and WFGs.
- In this mode, the internal RAM block works a frequency that is double of the user's clock. The maximum user's clock frequency is then reduced by a factor of 2, approximately.

3.7.3 Generics

mcka_edge

type bit

default value '0'

This generic represents the front polarity of the clock associated to the first port of the memory. '0' is for rising edge and '1' for falling edge.

mckb_edge

type bit

default value '0'

This generic represents the front polarity of the clock associated to the second port of the memory. '0' is for rising edge and '1' for falling edge.

pcka_edge

type bit

default value '0'

This generic represents the front polarity of the clock associated to the pipeline registers of the first port. '0' is for rising edge and '1' for falling edge.

pckb_edge

type bit

default value '0'

This generic represents the front polarity of the clock associated to the pipeline registers of the second port. '0' is for rising edge and '1' for falling edge.

mem_context

type string

default value ""

This generic represents the initial value of the RAM. The initial value can be optionally set by bitstream. The string contains a list of all complete bit words separated by coma.

When a word size is less than the RAM data size or when number of words is less than RAM word count, an error occurs.

When a word size exceeds RAM data size or when the number of words exceeds the RAM word count, an error occurs.

std_mode

type string

default value ""

This generic represents the predefined operating mode of the RAM. When empty the operating mode is defined by the 3 raw_config generics.

The available predefined modes are:

- "FAST_2kx18" → 2048 words of 18 bits with fast ECC
- "SLOW_2kx18" → 2048 words of 18 bits with ECC
- "NOECC_2kx24" → 2048 words of 24 bits without ECC
- "NOECC_4kx12" → 4096 words of 12 bits without ECC
- "NOECC_6kx8" → 6144 words of 8 bits without ECC
- "NOECC_12kx4" → 12288 words of 4 bits without ECC
- "NOECC_24kx2" → 24576 words of 2 bits without ECC
- "NOECC_48kx1" → 49152 words of 1 bits without ECC

When using one of these predefined modes, the 3 raw_config generics are defined as follow:

- raw_config0(3 downto 0) → b"0000"
- raw_config1
 - FAST_2kx18 b"0011100100100100"
 - SLOW_2kx18 b"1101100100100100"
 - NOECC_2kx24 b"0000101101101101"
 - NOECC_4kx12 b"0000100100100100"
 - NOECC_6kx8 b"0000011011011011"
 - NOECC_12kx4 b"0000010010010010"
 - NOECC_24kx2 b"0000001001001001"
 - NOECC_48kx1 b"0000000000000000"

raw_config0

type bit_vector(3 downto 0)

default value b"0000"

This generic configures the following fields:

Name	Index	Description
PB_OUT_PR_MUX	3	Port B output optional pipeline register. '0': no register '1': Pipe register is used

PA_OUT_PR_MUX	2	Port A output optional pipeline register. '0': no register '1': Pipe register is used
PB_IN_PR_MUX	1	Port B input optional pipeline register. '0': no register '1': Pipe register is used
PA_IN_PR_MUX	0	Port A input optional pipeline register. '0': no register '1': Pipe register is used

raw_config1

type bit_vector(15 downto 0)

default value b"0000000000000000"

This generic configures the following fields:

Name	Index	Description
PB_ECC_RRM	15	ECC Read Repair Mode on port B
PA_ECC_RRM	14	ECC Read Repair Mode on port A
PX_ECC_FAST	13	Fast mode ECC. Must be low if PB_ECC_RRM and PA_ECC_RRM are set to '1'
PX_ECC	12	Enable ECC
PB_OUT_WIDTH	11:9	B port output width
PA_OUT_WIDTH	8:6	A port output width
PB_IN_WIDTH	5:3	B port input width
PA_IN_WIDTH	2:0	A port input width

Input / output widths values depend on PX_ECC:

- PX_ECC = 0 (ECC deactivated)
 - 000: width is 1 bit
 - 001: width is 2 bits
 - 010: width is 4 bits
 - 011: width is 8 bits
 - 100: width is 12 bits
 - 101: width is 24 bits
 - other values are reserved
- Px_ECC = 1 (all ECC modes)
 - 100: width is 18 bits
 - other values are reserved

The bits raw_config1(15 downto 12) are used to define the NO_ECC, ECC_FAST or ECC_SLOW modes. The following table shows the possible configuration values.

15	14	13	12	Comment
0	0	0	0	Normal mode (NO ECC)
0	0	0	1	Invalid configuration
0	0	1	0	Invalid configuration
0	0	1	1	ECC FAST mode (no read repair)
0	1	0	0	Invalid configuration
0	1	0	1	ECC SLOW (read repair enabled on port A)
0	1	1	0	Invalid configuration
0	1	1	1	Invalid configuration
1	0	0	0	Invalid configuration
1	0	0	1	ECC SLOW (read repair enabled on port B)
1	0	1	0	Invalid configuration
1	0	1	1	Invalid configuration
1	1	0	0	Invalid configuration
1	1	0	1	ECC SLOW (read repair enabled on both ports)
1	1	1	0	Invalid configuration
1	1	1	1	Invalid configuration

3.7.4 Ports

Ports	Direction	Type	Description
ACK	input	std_logic	A port memory main clock
ACKC	input	std_logic	A port memory clock clone. Must be connected to the same clock source as ACK
ACKD	input	std_logic	A port memory 90° shifted clock. ACKD must be used when Read Repair Mode is selected on this port. It allows to internally generate a double frequency for the memory matrix, to allow read modify write during a single user's clock cycle.
ACKR	input	std_logic	A port register clock. ACKR must be fed by a valid clock (typically ACK), if the optional input or output pipeline registers are used.
BCK	input	std_logic	B port memory main clock.

BCKC	input	std_logic	B port memory clock clone. Same comments as for ACKC
BCKD	input	std_logic	B port memory 90° shifted clock. Just as ACKD, BCKD is used when Read Repair Mode is selected on B port
BCKR	input	std_logic	B port register clock. BCKR must be fed by a valid clock (typically BCK), if the optional input or output pipeline registers are used.
AI1 to AI24	input	std_logic	A port input data. See notes on data input width for proper operation.
BI1 to BI24	input	std_logic	B port input data. See notes on data input width for proper operation.
ACOR	output	std_logic	Goes high for one clock cycle when an error has been detected and corrected on port A
AERR	output	std_logic	Goes high for one clock cycle when an uncorrectable error has been found on port A
BCOR	output	std_logic	Goes high for one clock cycle when an error has been detected and corrected on port A
BERR	output	std_logic	Goes high for one clock cycle when an uncorrectable error has been found on port A
AO1 to AO24	output	std_logic	A port output data. See notes on data output width for proper operation
BO1 to BO24	output	std_logic	B port output data. See notes on data output width for proper operation
AA1 to AA16	input	std_logic	A port address. See notes on physical and logical addresses for proper operation
ACS	input	std_logic	A port chip select (active high)
AWE	input	std_logic	A port write enable (active high)
AR	input	std_logic	A port registers reset (active high)
BA1 to BA16	input	std_logic	B port address. See notes on physical and logical addresses for proper operation
BCS	input	std_logic	B port chip select (active high)
BWE	input	std_logic	B port write enable (active high)
BR	input	std_logic	B port registers reset (active high)

The ACKC port must be connected and is a clone of the memory clock (ACK).

When using one of the SLOW modes, ACKD port must be connected to a clock which is a 90° shifted version of ACK. ACKD can be generated with ACK by using a PLL and 2 WFG in the same CKG block.

The ACKR input clock is used only for the optional input and output pipeline registers. AR is the input for synchronous reset of those registers.

The BCKC and BCKD ports must be connected as described for ACKC and ACKD. BR is the input for synchronous reset of the optional input and output registers on port B.

3.7.5 Instantiation Example

This documentation only provides the instantiation of the component. For a full example, please refer to NX_RAM example provided in the example/nxLibrary installation directory.

```
-- RAM with Fast ECC: 1024 words of 18 bits and 1 read/write port

RAM_0 : NX_RAM
generic map (
    std_mode => "FAST_2kx18"
    , mem_context => "1111111111111111,001100110011001100," &
    "110011001100110011,1111111111111111" &
    "... "
    -- other 2048 words must be also initialized
)
port map (
    ACK => CLK, ACKC => CLK, ACKD => OPEN, ACKR => OPEN
    , AI1 => DI(0), ... , AI18 => DI(17)
    , AI19 => OPEN, ... , AI24 => OPEN
    , ACOR => COR, AERR => ERR
    , AO1 => DO(0), ... , AO18 => DO(17)
    , AO19 => OPEN, ... , AO24 => OPEN
    , AA1 => AD(0), ... , AA10 => AD(9)
    , AA11 => OPEN, ... , AA16 => OPEN
    , ACS => '1', AWE => WE, AR => OPEN
    , BCK => OPEN, BCKC => OPEN, BCKD => OPEN, BCKR => OPEN
    , BI1 => OPEN, ... , BI24 => OPEN
    , BCOR => OPEN, BERR => OPEN
    , BO1 => OPEN, ... , BO24 => OPEN
    , BA1 => OPEN, ... , BA16 => OPEN
    , BCS => OPEN, BWE => OPEN, BR => OPEN
);
```

3.7.6 Simulation

The NX_RAM VHDL simulation model is included in the NxLibrary (NxPackage.vhd). It allows to simulate any one of the possible configurations, including with ECC in FAST or SLOW (Read Repair Mode) modes.

3.8 NX_RAM_WRAP

3.8.1 Description

The NX_RAM_WRAP provides an alternate way to instantiate NX_RAM. It uses the same generics as NX_RAM, and the ports are grouped as busses whenever possible.

3.8.2 Generics

```

std_mode : string := "";
mcka_edge : bit := '0';
mckb_edge : bit := '0';
pcka_edge : bit := '0';
pckb_edge : bit := '0';
mem_ctxt : string := "";
raw_config0 : bit_vector( 3 downto 0) := B"0000";
raw_config1 : bit_vector(15 downto 0) := B"0000000000000000"

```

Please, refer to the NX_RAM chapter for more detailed information.

3.8.3 Ports

Ports	Direction	Type	Description
ACK	input	std_logic	A port memory main clock
ACKD	input	std_logic	A port memory 90° shifted clock. ACKD must be used when Read Repair Mode is selected on this port. It allows to internally generate a double frequency for the memory matrix, to allow read modify write during a single user's clock cycle.
ACKR	input	std_logic	A port register clock. ACKR must be fed by a valid clock (typically ACK), if the optional input or output pipeline registers are used.
BCK	input	std_logic	B port memory main clock.
BCKD	input	std_logic	B port memory 90° shifted clock. Just as ACKD, BCKD is used when Read Repair Mode is selected on B port.

BCKR	input	std_logic	B port register clock. BCKR must be fed by a valid clock (typically BCK), if the optional input or output pipeline registers are used.
AI(23:0)	input	std_logic_vector	A port input data. See notes on data input width for proper operation.
BI(23:0)	input	std_logic_vector	B port input data. See notes on data input width for proper operation.
ACOR	output	std_logic	Goes high for one clock cycle when an error has been detected and corrected on port A
AERR	output	std_logic	Goes high for one clock cycle when an uncorrectable error has been found on port A
BCOR	output	std_logic	Goes high for one clock cycle when an error has been detected and corrected on port A
BERR	output	std_logic	Goes high for one clock cycle when an uncorrectable error has been found on port A
AO(23:0)	output	std_logic_vector	A port output data. See notes on data output width for proper operation
BO(23:0)	output	std_logic_vector	B port output data. See notes on data output width for proper operation
AA(15:0)	input	std_logic_vector	A port address. See notes on physical and logical addresses for proper operation
ACS	input	std_logic	A port chip select (active high)
AWE	input	std_logic	A port write enable (active high)
AR	input	std_logic	A port registers reset (active high)
BA(15:0)	input	std_logic_vector	B port address. See notes on physical and logical addresses for proper operation
BCS	input	std_logic	B port chip select (active high)
BWE	input	std_logic	B port write enable (active high)
BR	input	std_logic	B port registers reset (active high)

4 I/O elements

4.1 NX_IOB

4.1.1 Description

The NX_IOB component describes a bidirectional port of the design. The behavior is:

$$O \leq IO$$

$$IO \leq I \text{ when } C = '1'$$

Termination is active when $T = '1'$.

The NX_IOB can be instantiated anywhere in the design hierarchy. It allows to define buried ports (no signal appears in the ports list).

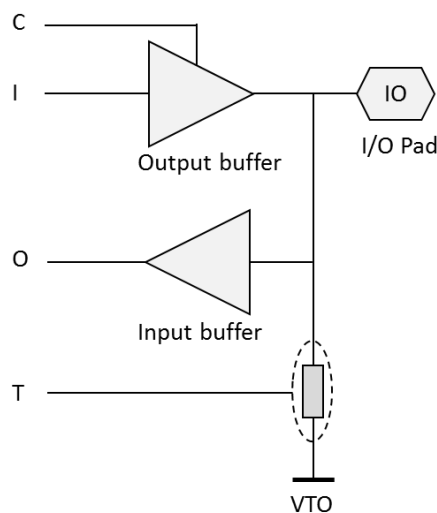


Figure 15: IOB diagram

4.1.2 Generics

Note that the generic assigned to this primitive can be overridden by the addPad or addPads methods in the script file.

location

type string
default value Undefined (no default value)

This generic specifies the position of the physical pad in the IO ring. Example :

`location => "IOB10_D09N"`

padType

type string

default value "LCVMOS_2.5V_2mA"

This generic specifies the electrical standard of the IO, including its power supply and output current drive. The list of the possible values is described in the NanoXplore_NXmap_User_Manual. Example :

padType => "LVCMOS_2.5V_8mA"

differential

type string ("true" or "false")

default value "false"

This generic specifies if the IO uses a differential standard. Example :

differential => "true"

slewRate

type string ("Slow", "Medium" or "Fast")

default value "Medium"

This generic specifies slewrate of the output buffer. Example :

slewRate => "Fast"

termination

type string (value in ohms – range 30 to 80)

default value "" (no termination)

This generic specifies the value of the input impedance resistors. It's specified in Ohms, in a range 30 to 80 Ohms. Example :

termination => "50"

terminationReference

type string => "floating" or "VT"

default value "VT"

This generic specifies if the input termination resistors are floating or connected to the VT voltage reference. Can be useful for some differential input cases. Example :

termination => "floating"

turbo

type string => "true" or "false"

default value "false"

This generic specifies if the input buffer is in turbo mode. Example :

turbo => "true"

weakTermination

type string => "None", "PullUp", "PullDown" or "Keeper"

default value "None"

This generic specifies if the input pad is using weak termination impedance. Note that only 'None' and 'PullUp' are allowed in NG-MEDIUM. Example :

```
weakTermination => "PullUp"
```

inputDelayLine

type string => "0" to "63" (" " – empty string to bypass the delay line)

default value "0"

This generic specifies the number of 160 ps delay taps used on the input path. Example :

```
inputDelayLine => "27"
```

outputDelayLine

type string => "0" to "63" (" " – empty string to bypass the delay line)

default value "0"

This generic specifies the number of 160 ps delay taps used on the output path. Example :

```
outputDelayLine => "35"
```

inputSignalSlope

type string => "decimal value in V/ns" (range 0.5 to 20)

default value ""

This generic has no effect on the implementation process, but it's used by the timing analyzer. The value must be specified in Volts/ns..Example :

```
inputSignalSlope => "8"
```

outputCapacity

type string "integer_value in pF" (range 0 to 40)

default value "0"

This generic has no effect on the implementation process, but it's used by the timing analyzer. The value must be specified in ps..Example :

```
outputCapacity => "15"
```

locked

type bit => '0' or '1'

default value '0'

This generic specifies if the "location" on the instantiated NX_IOB is done in the instantiation (locked => '1') or in the Nxpython script file (locked => '0'). Example :

```
location => "IOB12_D4P",
```

```
locked => '1',
```


4.1.3 Ports

Ports	Direction	Type	Description
I	Input	std_logic	From FPGA fabric
C	Input	std_logic	Tristate control '0': High impedance '1': Enable output
T	Input	std_logic	Termination control '0': No calibration '1': calibration activated
O	output	std_logic	To FPGA fabric
IO	inout	std_logic	External pad

4.1.4 Example

This documentation only provides the instantiation of the component. For a full example, please refer to NX_IOB example provided in the example/nxLibrary installation directory.

```

IOB_0 : NX_IOB
generic map(
  location          => "IOB12_D07P",
  padType           => "LVCMOS_2.5V_8mA",
  slewRate          => "Fast",
  turbo             => "true",
  inputDelayLine    => "10",
  outputDelayLine   => "17",
  outputCapacity    => "15",
  locked            => '1'
)
port map (
  I => fromFPGACore
  , O => toFPGACore
  , C => enable
  , T => '0'
  , IO => open -- A signal name is not required on the external
               -- signal
               -- The pad will take the name of the instance
);

```

4.2 NX_IOB_I

4.2.1 Description

The NX_IOB_I component describes an input port of the design. The behavior is:

$O \leq IO$

Termination is active when T = '1'.

The NX_IOB can be instantiated anywhere in the design hierarchy. It allows to define buried ports (no signal appears in the ports list).

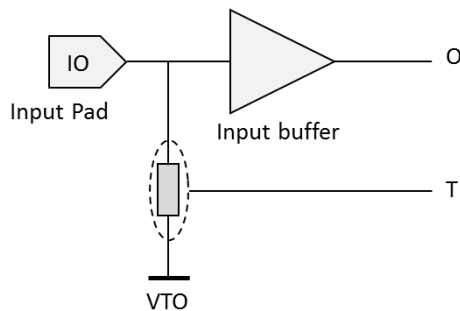


Figure 16: IOB_I diagram

4.2.2 Generics

Note that the generic assigned to this primitive can be overridden by the addPad or addPads methods in the script file.

location

type string
default value Undefined (no default value)

This generic specifies the position of the physical pad in the IO ring. Example :

`location => "IOB10_D09N"`

padType

type string
default value "LCVMOS_2.5V_2mA"

This generic specifies the electrical standard of the IO, including its power supply and output current drive. The list of the possible values is described in the NanoXplore_NXmap_User_Manual. Example :

`padType => "LVCMOS_2.5V_8mA"`

differential

type string => "True" or "False"
default value "False"

This generic specifies if the IO uses a differential standard. Example :

`differential => "True"`

termination

type string => "value in ohms" (range 30 to 80)

default value "" (no termination)

This generic specifies the value of the input impedance resistors. It's specified in Ohms, in a range 30 to 80 Ohms. Example :

termination => "50"

terminationReference

type string => "floating" or "VT"

default value "VT"

This generic specifies if the input termination resistors are floating or connected to the VT voltage reference. Can be useful for some differential input cases. Example :

termination => "floating"

turbo

type string => "true" or "false"

default value "false"

This generic specifies if the input buffer is in turbo mode. Example :

turbo => "true"

weakTermination

type string => "None", "PullUp", "PullDown" or "Keeper"

default value "None"

This generic specifies if the input pad is using weak termination impedance. Note that only 'None' and 'PullUp' are allowed in NG-MEDIUM. Example :

weakTermination => "PullUp"

inputDelayLine

type string => "0" to "63" (" - empty string to bypass the delay line)

default value "0"

This generic specifies the number of 160 ps delay taps used on the input path. Example :

inputDelayLine => "27"

inputSignalSlope

type string => "decimal value in V/ns" (range 0.5 to 20)

default value ""

This generic has no effect on the implementation process, but it's used by the timing analyzer. The value must be specified in Volts/ns..Example :

inputSignalSlope => "8"

locked

type bit => '0' or '1'

default value '0'

This generic specifies if the "location" on the instantiated NX_IOB is done in the instantiation (locked => '1') or in the Nxpython script file (locked => '0'). Example :

location => "IOB12_D4P",

locked => '1',

4.2.3 Ports

Ports	Direction	Type	Description
I	Input	std_logic	Not used. Must be left "open" or unconnected
C	Input	Std_logic	Not used. Must be left "open" or unconnected
T	Input	std_logic	Termination control '0' : No termination '1' : Input termination activated
O	output	std_logic	From FPGA fabric
IO	Input	std_logic	External pad

4.2.4 Example

This documentation only provides the instantiation of the component. For a full example, please refer to NX_IOB_I example provided in the example/nxLibrary installation directory.

```

IOB_0 : NX_IOB_I
generic map(
  location          => "IOB12_D10P",
  padType           => "LVCMOS_2.5V_4mA",
  turbo            => "True",
  inputDelayOn      => '1',
  inputDelayLine    => "13",
  inputSignalSlope  => "8",
  locked            => '1'
)
port map (
  O => toFPGACore
  , T => '1'
  , IO => open    -- A signal name is not required on the external
                  -- signal
                  -- The pad will take the name of the instance
);

```

4.3 NX_IOB_O

4.3.1 Description

The NX_IOB_O component describes an output port of the design. The behavior is:

$IO \leq I$ when $C = '1'$ else 'Z'

Termination is active when $T = '1'$.

The NX_IOB can be instantiated anywhere in the design hierarchy. It allows to define buried ports (no signal appears in the ports list).

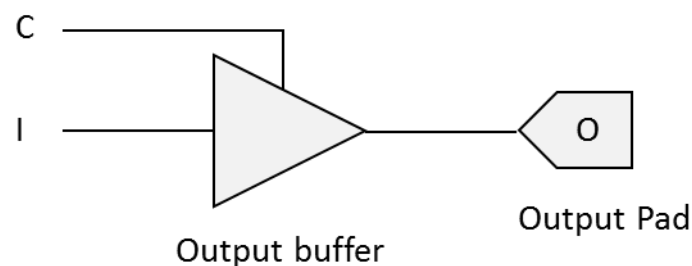


Figure 17: IOB_O diagram

4.3.2 Generics

Note that the generic assigned to this primitive can be overridden by the addPad or addPads methods in the script file.

location

type string
default value Undefined (no default value)

This generic specifies the position of the physical pad in the IO ring. Example :

`location => "IOB10_D09N"`

padType

type string
default value "LCVMOS_2.5V_2mA"

This generic specifies the electrical standard of the IO, including its power supply and output current drive. The list of the possible values is described in the NanoXplore_NXmap_User_Manual. Example :

`padType => "LVCMOS_1.5V_8mA"`

differential

type string => "True" or "False"

default value "False"

This generic specifies if the IO uses a differential standard. Example :

differential => "True"

slewRate

type string => "Slow", "Medium" or "Fast"

default value "Medium"

This generic specifies slewrate of the output buffer. Example :

slewRate => "Fast"

weakTermination

type string => "None", "PullUp", "PullDown" or "Keeper"

default value "None"

This generic specifies if the input pad is using weak termination impedance. Note that only "None" and "PullUp" are allowed in NG-MEDIUM. Example :

weakTermination => "PullUp"

outputDelayLine

type string => "0" to "63" (" " – empty string to bypass the delay line)

default value "0"

This generic specifies the number of 160 ps delay taps used on the output path. Example :

outputDelayLine => "35"

outputCapacity

type string "integer value in pF" (range 0 to 40)

default value ""

This generic has no effect on the implementation process, but it's used by the timing analyzer. The value must be specified in ps..Example :

outputCapacity => "15"

locked

type bit => '0' or '1'

default value '0'

This generic specifies if the "location" on the instantiated NX_IOB is done in the instantiation (locked => '1') or in the Nxpython script fine (locked => '0'). Example :

location => "IOB12_D4P",

locked => '1',

4.3.3 Ports

Ports	Direction	Type	Description
I	input	std_logic	From FPGA fabric
C	input	std_logic	Tristate control ('0' for High Z)
T	input	std_logic	Not used. . Must be left "open" or unconnected
IO	output	std_logic	External pad
O	Output	std_logic	Not used. . Must be left "open" or unconnected

4.3.4 Example

This documentation only provides the instantiation of the component. For a full example, please refer to NX_IOB_O example provided in the example/nxLibrary installation directory.

```

IOB_0 : NX_IOB_O

generic map(
  location          => "IOB12_D10P",
  padType           => "LVCMOS_2.5V_4mA",
  slewRate          => "Fast",
  outputDelayOn     => '1',
  outputDelayLine   => "17",
  outputCapacity    => "15",
  locked            => '1'
)
port map (
  I => fromFPGA
  , C => enable
  , IO => open    -- A signal name is not required on the external
                  -- signal
                  -- The pad will take the name of the instance
);

```

4.4 SERIALizers and DESerializers

4.4.1 Introduction

The NG-MEDIUM complex I/O banks provide serializers and deserializer features. Each serializer or deserializer can use a serialization factor of 2, 3, 4 or 5.

In addition in each I/O pair, the serializers/deserializers associated to the “_P” pad can be chained with its neighbor (associated to the “_N” pad) allowing thus serialization/deserialization factors of 6, 7, 8, 9 and 10.

The serializer/deserializer data path requires two clocks: bit clock (Fast clock – FCK) and word clock (Slow clock – SCK).

Serializers include optional output delay lines for both output and tri-state command. Although output delay lines can be dynamically controlled, serializer delays are usually configured in static mode – most often no delay.

Deserializers require a proper data/clock alignment mechanism for safe sampling, as well as word alignment to recover the original words. Data/clock alignment requires a dynamic control of the delay lines to adjust the phase relationship of the sampled data and the fast clock. This procedure is called Dynamic Phase Alignment (DPA). It requires a training sequence.

NG-MEDIUM complex IO banks provide hardware support for DPA. The dynamic control of the delay lines requires an additional clock to read or write into the delay registers. This clock is called DCK. It can be synchronous or asynchronous with the data path clocks SCK and FCK.

All I/O related delay lines have 0 to 63 x 160 ps steps delays.

4.4.2 SERDES architecture overview

The serializers/deserializers architecture contains two main blocks :

- Data path
- Delay control path

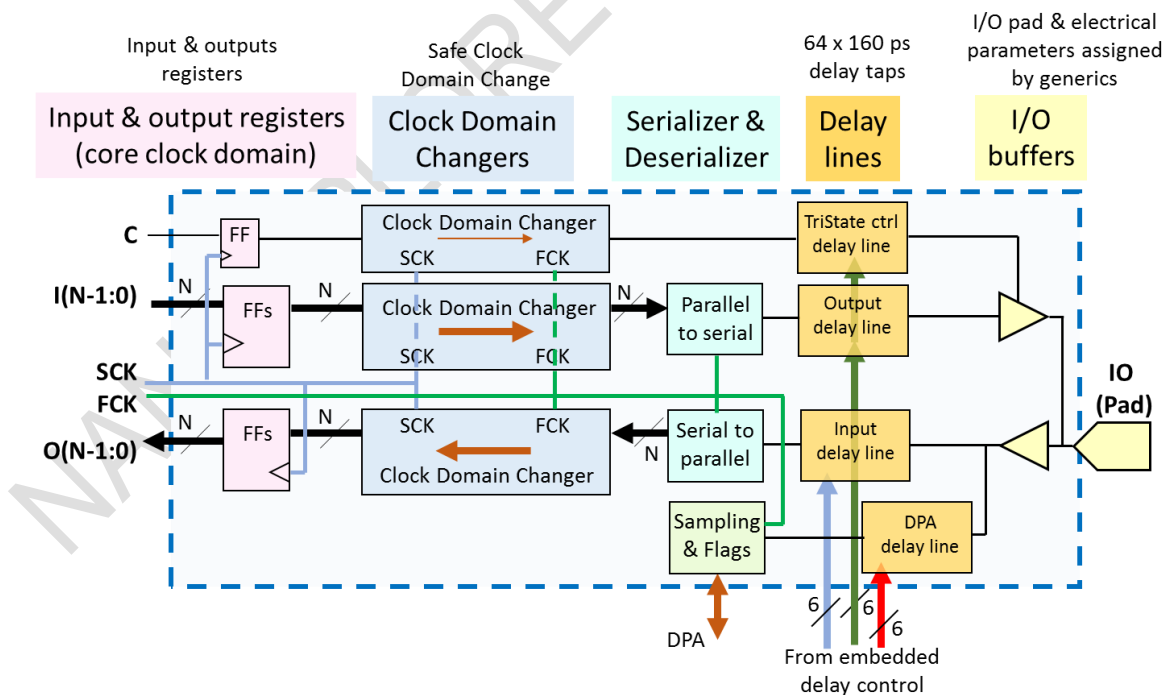
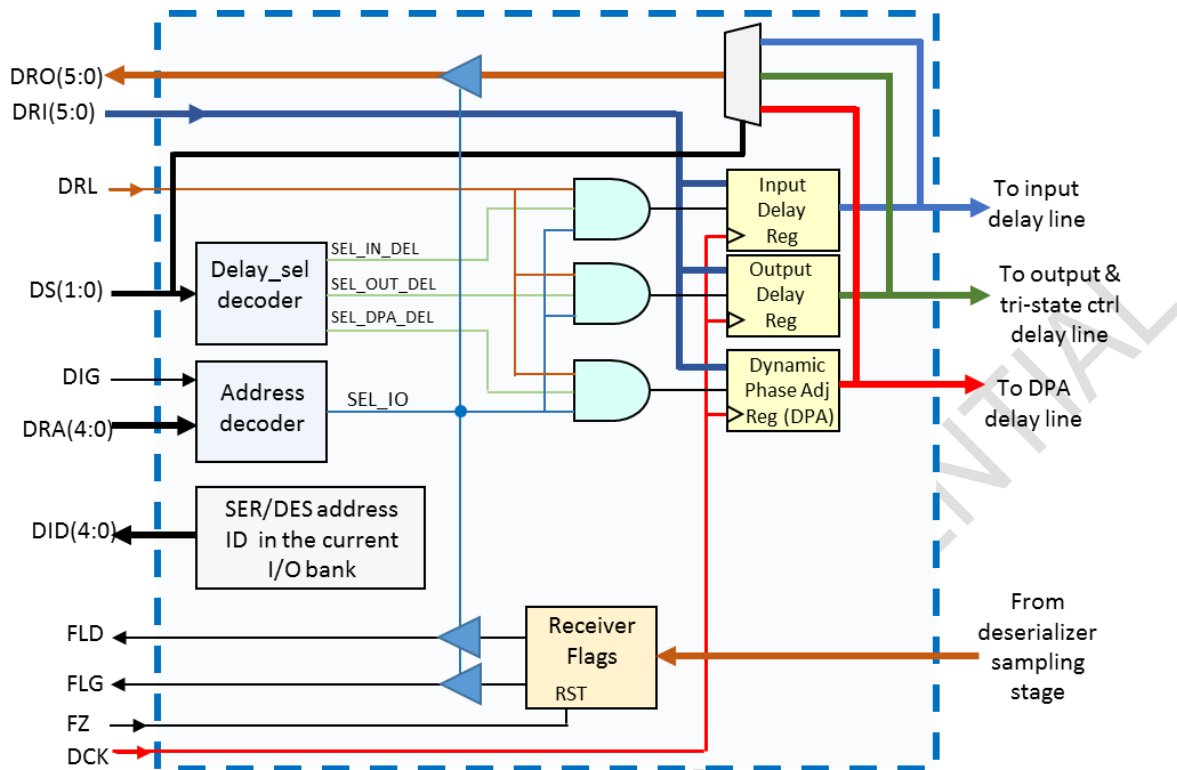


Figure 18: SERDES data path simplified diagram**Figure 19: SERDES delay lines control block simplified diagram**

4.4.3 DPA : Dynamic Phase Adjustment

NG-MEDIUM architecture provides hardware support for Dynamic Phase Adjustment on NX_DES. The following describes how to implement the adjustment procedure.

In the complex banks, all I/Os include three user's selectable and adjustable delay lines that can be selected with "DS(1:0)" sub-address input. Those registers can be read and written with a simple microprocessor-like interface.

Each NX_DES or NX_SER include three delay lines, respectively for output (and tri-state control), input path and DPA path delays. The delay value on each one of those three paths (number of 160 ps delay taps) is defined by the value written into the corresponding delay register.

- The output (and tri-state control) delay register controls the delay inserted on the output data path.
- The input delay register controls the delay inserted between the input pad and the input register.
- The DPA delay register controls the delay inserted between the input pad and the DPA input register.

The DPA logic allows to generate flags (FLD and FLG) to inform about the data input and fast clock relative phase :

- FLD : this flag goes high when a transition on the data line at the output of the DPA delay line, occurs between the falling and the rising edge of the sampling clock (FCK, fast clock).
 - FLG : this flag goes high when a transition on the data line at the output of the DPA delay line, occurs between the rising and the falling edge of the sampling clock (FCK, fast clock).
 - FZ : Active low flags reset
- By modifying the value of the DPA delay line, data/clock relationship can be analyzed by monitoring the FLD and FLG flags – and then deduce the optimal delay value to be written to the input delay register.
 - The following figures show the FLD and FLG flags behavior versus the transition detection on the DPA delay line output.

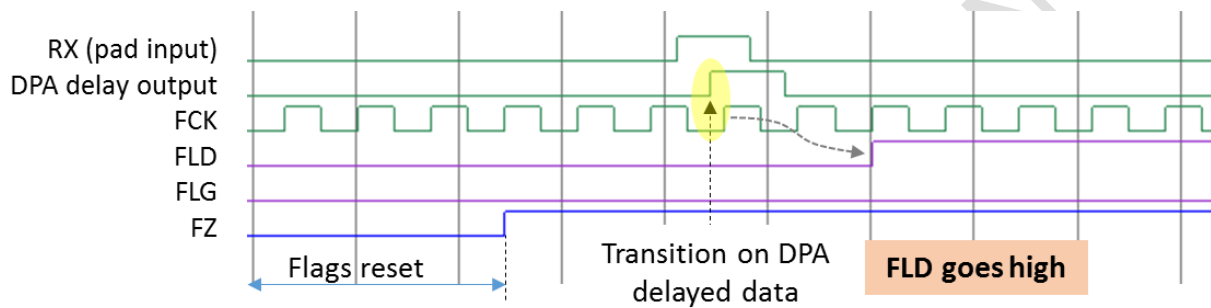


Figure 20: FLD activation

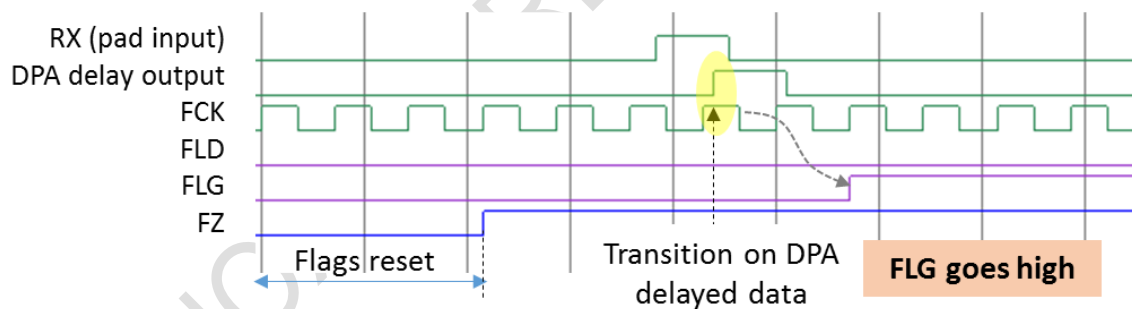


Figure 21: FLG activation

Write and read accesses to the delay registers can be easily managed with the following signals :

- DCK : delay registers clock (can be asynchronous with SCK/FCK). Usually 2 to 20 MHz. Write operations occur on DCK rising edge.

- DID(4:0) : address identifier of the considered I/O in the complex bank (0 to 29).
- DRA(4:0) : address of the I/O in the considered complex bank (0 to 29). Note that when DRA = DID, the DRO outputs, as well as FLD and FLG flags outputs of the considered I/O go to low impedance (allowing thus to be read by the fabric).
- DS(1:0) : allow to select the destination register into the DRA selected I/O. See next table for details.

DS value	Selected delay register
00	Output (and tri-state control) delay register
01	Input delay register
10	DPA delay register
11	Reserved

- DRI(5:0) :value to be written into the selected register.
- DRL : active high load (write enable)

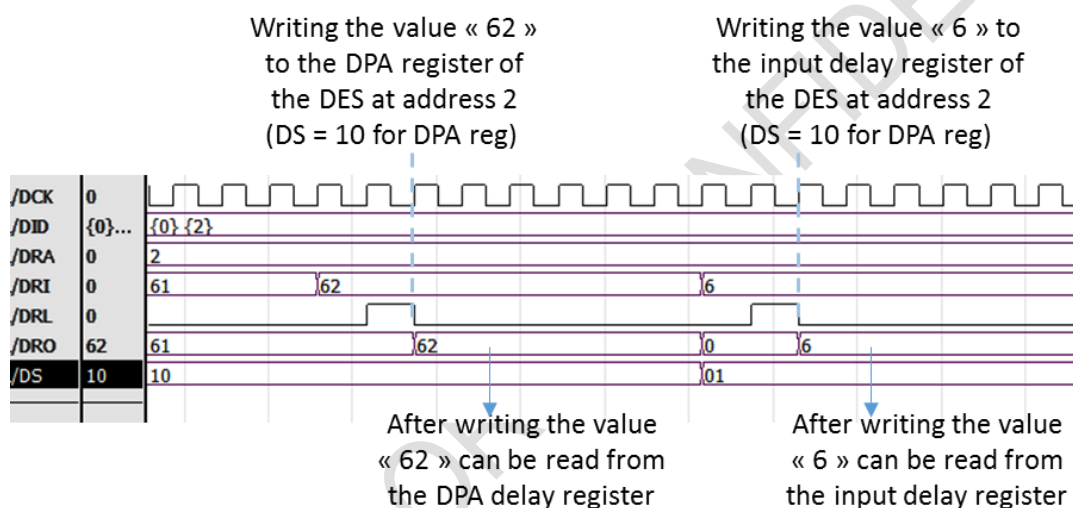


Figure 22: Writing and reading delay registers

Note :

For deserialization, NanoXplore provides an IP Core that automatically adjusts the delay lines in order to properly align the sampled data with the fast clock. It also provides word alignment.

NanoXplore recommends to use NXcore – customizable IP Core generator - to use deserializers with automatic data/clock phase alignment and word alignment.

The IP Core uses an automatic procedure – launched by the user at any time – to proceed to the input delays calibration and word alignment on all DESerializers of the same group.

The IP Core also generates the required clocks (SCK, FCK and DCK) from a word clock input. It requires using the neighboring Clock Generator block (CKG1 for I/O complex banks 11 & 12 for example).

Handshake signals with the transmitter and calibration status are provided to the user application. Among the available signals :

- LAUNCH_CALIB (input) : launches the calibration process (on a rising edge)
- TRAINING_REQ_OUT (output) : The IP Core requests the transmitter to send the serialized "TrainingValue" to the DESerializer(s).
- TRAINING_ACK_IN (input) : The transmitter is ready and sends the serialized "TrainingValue"
- TRAINING_REQ_IN (input) : if the IP Core is used as transmitter, the receiver might require a calibration sequence where the transmitter must send the serialized "TrainingValue". TRAINING_REQ_IN is the request input of the transmitter.
- TRAINING_ACK_OUT (output) : When the transmitter receives a request from the receiver, it sends the serialized "TrainingValue" and activates the TRAINING_ACK output to the receiver.
- CALIB_DONE (output) : goes high to state that the calibration process is done.
- CALIB_ERROR (output) : Active high status bit to state that the calibration was not successful.

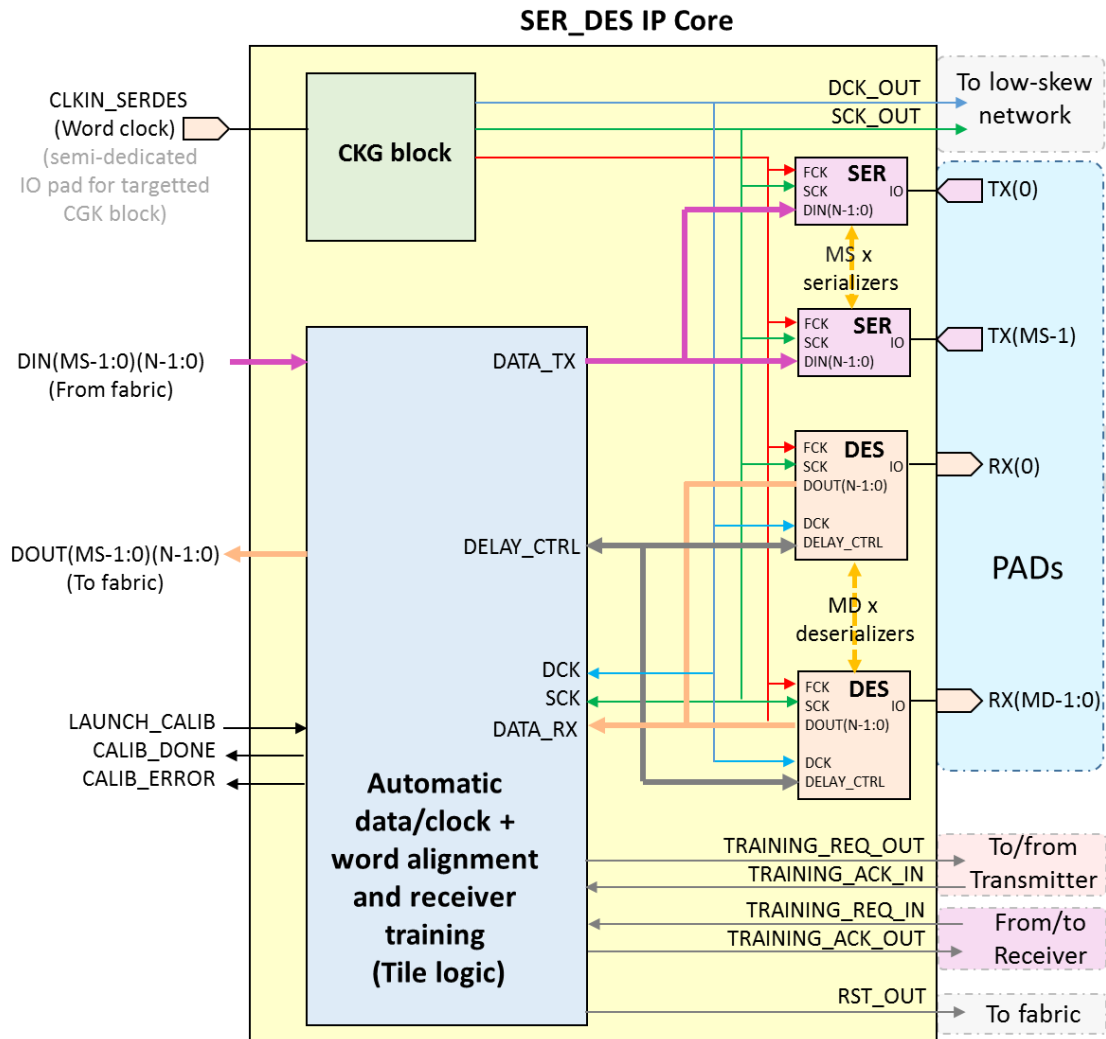


Figure 23: SER_DES IP Core simplified diagram

4.5 NX_DES

4.5.1 Description

The NX_DES is a high performance DESerializer. The complex banks allows to configure the I/Os as DESerializers with deserialization factor from 3 to 5. Higher deserialization factors (6, 7, 8, 9 and 10) can be achieved by combining the two deserializers of a differential IO pair.

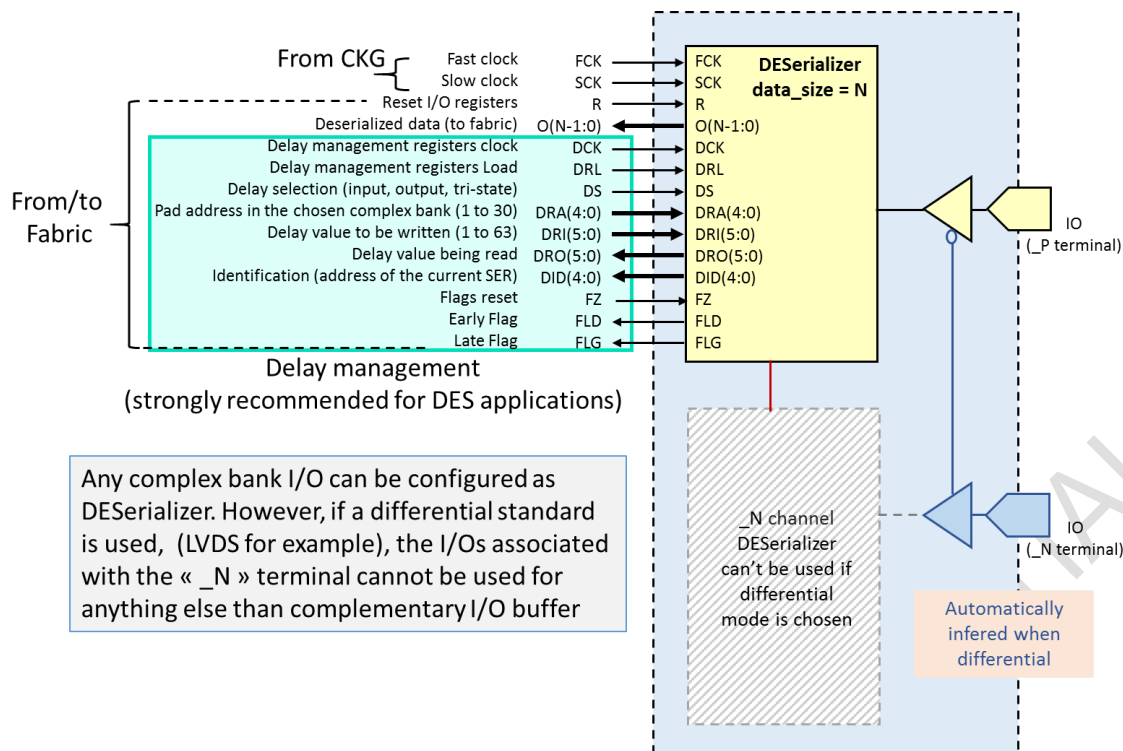


Figure 24 NX_DES primitive

4.5.2 Generics

data_size

type integer

default value Undefined (no default value)

This generic specifies the deserialization factor. Example :

data_size => 8

location

type string

default value Undefined (no default value)

This generic specifies the position of the physical pad in the IO ring. Example :

location => "IOB12_D01P"

padType

type string

default value "LCVMOS_2.5V_2mA"

This generic specifies the electrical standard of the IO, including its power supply and output current drive. The list of the possible values is described in the NanoXplore_NXmap_User_Manual. Example :

```
padType => "LVCMOS_1.5V_8mA"
```

differential

type string => "True" or "False"

default value "False"

This generic specifies if the IO uses a differential standard. Example :

```
differential => "True"
```

termination

type string => "value in ohms" (range 30 to 80)

default value "" (no termination)

This generic specifies the value of the input impedance resistors. It's specified in Ohms, in a range 30 to 80 Ohms. Example :

```
termination => "50"
```

terminationReference

type string => "floating" or "VT"

default value "VT"

This generic specifies if the input termination resistors are floating or connected to the VT voltage reference. Can be useful for some differential input cases. Example :

```
termination => "floating"
```

turbo

type string => "true" or "false"

default value "false"

This generic specifies if the input buffer is in turbo mode. Example :

```
turbo => "true"
```

weakTermination

type string => "None", "PullUp", "PullDown" or "Keeper"

default value "None"

This generic specifies if the input pad is using weak termination impedance. Note that only 'None' and 'PullUp' are allowed in NG-MEDIUM. Example :

```
weakTermination => "PullUp"
```

inputDelayLine

type string => "0" to "63" (" " – empty string for dynamic delay)

default value "0"

This generic specifies the number of 160 ps delay taps used on the input path. Example :

`inputDelayLine => "27"`

inputSignalSlope

type string => "decimal value in V/ns" (range 0.5 to 20)

default value ""

This generic has no effect on the implementation process, but it's used by the timing analyzer. The value must be specified in Volts/ns..Example :

`inputSignalSlope => "8"`

4.5.3 Ports

Ports	Direction	Type	Description
FCK	In	Std_logic	Fast clock (bit clock)
SCK	In	Std_logic	Slow clock (word clock)
R	In	Std_logic	Active high Reset
IO	In	Std_logic	Input pad
O	Out	Std_logic_vector (data_size-1 downto 0)	Sampled word to FPGA fabric
DCK	In	Std_logic	Delay lines management registers clock
DRL	In	Std_logic	Delay Registers Load
DIG	In	Std_logic	Active low multicast write
DS	In	Std_logic_vector (1 downto 0)	Delay Select : 00 => out & tri-state
DRA	In	Std_logic_vector (4 downto 0)	Delay address (0 to 29)
DRI	In	Std_logic_vector (5 downto 0)	Data input to delay registers
DRO	Out (with tri-state)	Std_logic_vector (5 downto 0)	Delay value being read Active when DRA = DID else high impedance
DID	Out	Std_logic_vector (4 downto 0)	Pad address identification
FZ	In	Std_logic	Active low Flags Reset
FLD	Out (with tri-state)	Std_logic	Early capture flag Active when DRA = DID else high impedance
FLG	Out (with tri-state)	Std_logic	Late capture flag Active when DRA = DID else high impedance

4.6 NX_SER

4.6.1 Description

The NX_SER is a high performance SERIALizer. The complex banks allows to configure the I/Os as SERIALizers with serialization factor from 3 to 5. Higher serialization factors (6, 7, 8, 9 and 10) can be achieved by combining the two serializers of a differential IO pair.

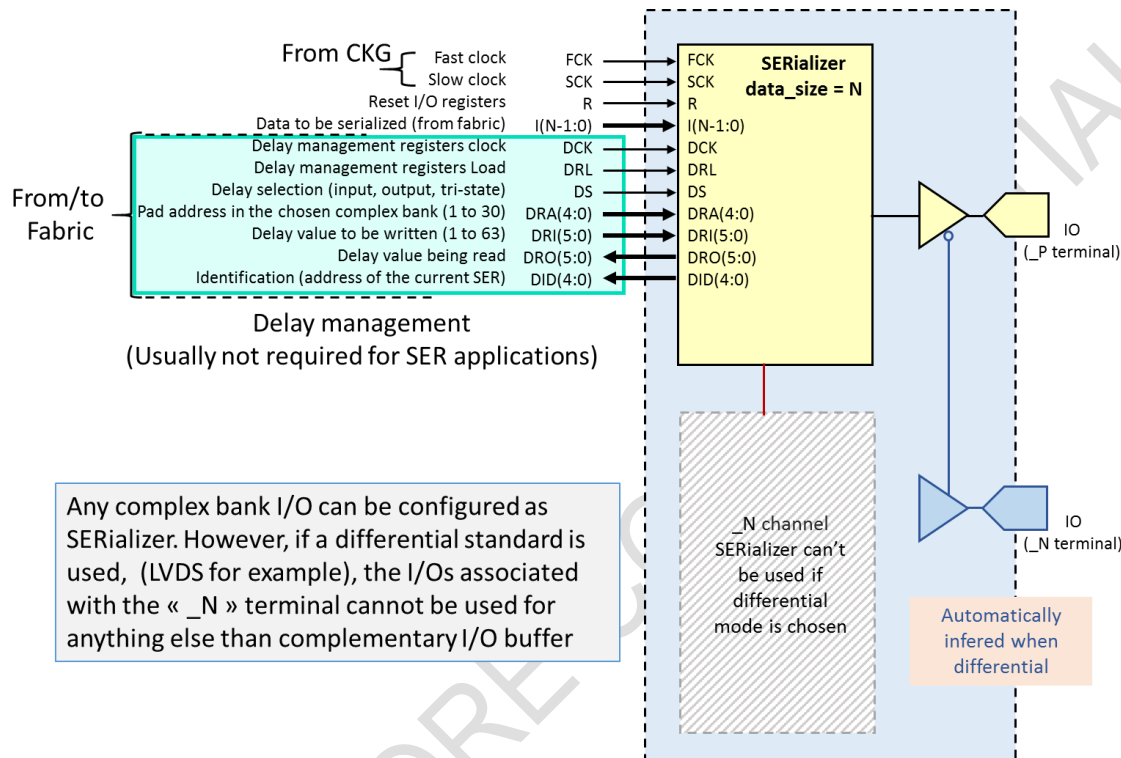


Figure 25: NX_SER primitive

4.6.2 Generics

data_size

type integer

default value Undefined (no default value)

This generic specifies the serialization factor. Example :

`data_size => 8`

location

type string

default value Undefined (no default value)

This generic specifies the position of the physical pad in the IO ring. Example :

location => "IOB10_D09N"

padType

type string

default value "LCVMOS_2.5V_2mA"

This generic specifies the electrical standard of the IO, including its power supply and output current drive. The list of the possible values is described in the NanoXplore_NXmap_User_Manual. Example :

padType => "LVCMOS_1.5V_8mA"

differential

type string => "True" or "False"

default value "False"

This generic specifies if the IO uses a differential standard. Example :

differential => "True"

slewRate

type string => "Slow", "Medium" or "Fast"

default value "Medium"

This generic specifies slewrate of the output buffer. Example :

slewRate => "Fast"

outputDelayLine

type string => "0" to "63" (" " – empty string to bypass the delay line)

default value "0"

This generic specifies the number of 160 ps delay taps used on the output path. Example :

outputDelayLine => "35"

outputCapacity

type string "integer value in pF" (range 0 to 40)

default value ""

This generic has no effect on the implementation process, but it's used by the timing analyzer. The value must be specified in ps..Example :

outpuCapacity => "15"

4.6.3 Ports

Ports	Direction	Type	Description
FCK	In	Std_logic	Fast clock (bit clock)
SCK	In	Std_logic	Slow clock (word clock)
R	In	Std_logic	Active high Reset
IO	Out	Std_logic	Input pad
I	In	Std_logic_vector (data_size-1 downto 0)	Data to be serialized from fabric
DCK	In	Std_logic	Delay lines management registers clock
DRL	In	Std_logic	Delay Registers Load
DS	In	Std_logic_vector (1 downto 0)	Delay Select : 00 => out & tri-state regs 01 => input delay register 10 => DPA delay register 11 => RESERVED
DRA	In	Std_logic_vector (4 downto 0)	Delay address (0 to 29)
DRI	In	Std_logic_vector (5 downto 0)	Data input to delay registers
DRO	Out (with tri-state)	Std_logic_vector (5 downto 0)	Delay value being read Active when DRA = DID, else high-impedance)
DID	Out	Std_logic_vector (4 downto 0)	Pad address identifier (0 to 29)

5 Reserved

There are some other components defined in NX library that are reserved for post synthesis and post place & route simulation. These components cannot be instantiated in pre synthesis VHDL.

The reserved components are:

- NX_BUFFER
- NX_CSC
- NX_SCC

NANOXPLORE CONFIDENTIAL