Quantum® Trion® Primitives User Guide

UG-EFN-PRIMITIVES-v6.2
September 2025
www.efinixinc.com

Contents

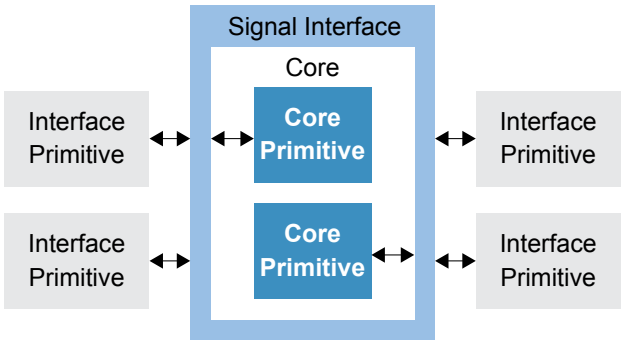
Introduction

This document defines the Efinity® software technology-mapped logic primitives, which are the basic building blocks of the user netlist that is passed to the place-and-route tool.

Core Primitives

The core primitives represent the functionality of the XLR cells, RAM blocks, DSP Blocks, and global clock buffers. These primitives connect to the interface primitives through a signal interface.

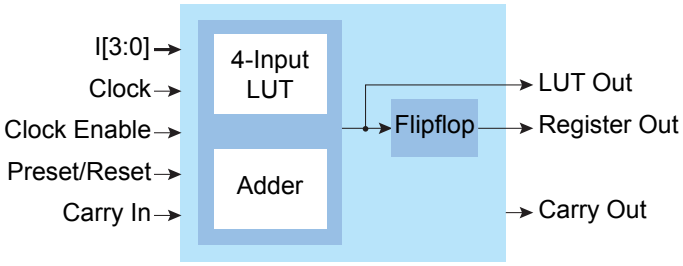
Figure 1: Core Primitives



Logic Cell

The logic cell consists of combinational logic, which can be a 4-input LUT or a full adder and a register. The register may be bypassed.

Figure 2: Logic Cell (Logical View)



Logic cell primitives:

- EFX_LUT4 on page 6
- EFX_ADD on page 9
- EFX_FF on page 12

EFX_LUT4

Simple 4-Input LUT ROM

The EFX_LUT4 primitive is a simple 4-input LUT ROM. Leave unused LUT inputs unconnected and set the LUTMASK value so that it does not depend on them. The software generates an error if the LUTMASK depends on an unconnected input.

EFX_LUT4 Ports

Figure 3: EFX_LUT4 Symbol

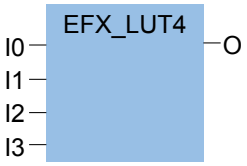


Table 1: EFX_LUT4 Ports

Port	Direction	Description
I0	Input	Data in 0.
I1	Input	Data in 1.
I2	Input	Data in 2.
I3	Input	Data in 3.
O	Output	Data out.

EFX_LUT4 Parameters

Table 2: EFX_LUT4 Parameters

Parameter	Allowed Values	Description
LUTMASK	Any 16 bit hexadecimal number	Content of LUT ROM.

EFX_LUT4 Function

Table 3: EFX_LUT4 Function

Inputs				Output
I3	I2	I1	I0	O
0	0	0	0	LUTMASK[0]
0	0	0	1	LUTMASK[1]
0	0	1	0	LUTMASK[2]

Inputs				Output
I3	I2	I1	I0	O
0	0	1	1	LUTMASK[3]
0	1	0	0	LUTMASK[4]
0	1	0	1	LUTMASK[5]
0	1	1	0	LUTMASK[6]
0	1	1	1	LUTMASK[7]
1	0	0	0	LUTMASK[8]
1	0	0	1	LUTMASK[9]
1	0	1	0	LUTMASK[10]
1	0	1	1	LUTMASK[11]
1	1	0	0	LUTMASK[12]
1	1	0	1	LUTMASK[13]
1	1	1	0	LUTMASK[14]
1	1	1	1	LUTMASK[15]

Figure 4: EFX_LUT4 Verilog HDL Instantiation

```
EFX_LUT4 # (
    .LUTMASK(16'hFFFE) // LUT contents (4 input 'OR')
) EFX_LUT4_inst (
    .O(O),          // LUT output
    .I0(I0),        // LUT input 0
    .I1(I1),        // LUT input 1
    .I2(I2),        // LUT input 2
    .I3(I3)         // LUT input 3
);
```

Figure 5: EFX_LUT4 VHDL Instantiation

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library efxphysicallib;
use efxphysicallib.efxcomponents.all;

entity LUT4_VHDL is
   port
   (
      din : in std_logic_vector(3 downto 0);
      dout : out std_logic
   );
end entity LUT4_VHDL;

architecture Behavioral of LUT4_VHDL is
begin

 EFX_LUT4_inst : EFX_LUT4
   generic map (
      LUTMASK => x"8888"
   )
   port map (
      I0 => din(0),
      I1 => din(1),
      I2 => din(2),
      I3 => din(3),
      O => dout
   );

end architecture Behavioral;
```

EFX_ADD

Simple Full Adder

The EFX_ADD primitive is a simple full adder. The carry-in (CI) and carry-out (CO) connections are dedicated routing between logic cells. Therefore, the first CI in an adder chain must be tied to ground. To access the CO signal through general logic, insert one adder cell to the end of the adder chain to propagate the CO to the sum.

If unused, connect the adder inputs (I1 and I0) to ground.

EFX_ADD Ports

Figure 6: EFX_ADD Symbol

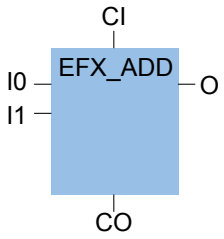


Table 4: EFX_ADD Ports

Port	Direction	Description
I0	Input	Data in 0.
I1	Input	Data in 1.
CI	Input	Carry in.
O	Output	Sum out.
CO	Output	Carry out.

EFX_ADD Parameters

Table 5: EFX_ADD Parameters

Parameter	Allowed Values	Description
I0_POLARITY	0, 1	0: Inverting, 1: Non-inverting (default).
I1_POLARITY	0, 1	0: Inverting, 1: Non-inverting (default).

EFX_ADD Function

Table 6: EFX_ADD Function

Inputs			Outputs	
CI	**I1**	**I0**	**CO**	**O**
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Figure 7: EFX_ADD Verilog HDL Instantiation

```
EFX_ADD # (
    .I0_POLARITY(1'b1),    // 0 inverting, 1 non-inverting
    .I1_POLARITY(1'b0)     // 0 inverting, 1 non-inverting
) EFX_ADD_inst (
    .O(O),          // Sum output
    .CO(CO),        // Carry output
    .I0(I0),        // Adder input 0
    .I1(I1),        // Adder input 1
    .CI(CI)         // Carry input
);
```

Figure 8: EFX_ADD VHDL Instantiation

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library efxphysicallib;
use efxphysicallib.efxcomponents.all;

entity ADDER_VHDL is
   port
   (
      din_a  : in std_logic_vector(2 downto 0);
      din_b  : in std_logic_vector(2 downto 0);
      sum : out std_logic_vector(2 downto 0)
   );
end entity ADDER_VHDL;

architecture Behavioral of ADDER_VHDL is
signal carry_out : std_logic_vector(1 downto 0);
begin

 EFX_ADD_inst_1 : EFX_ADD
   generic map (
      I0_POLARITY => 1,
      I1_POLARITY => 1
   )
   port map (
      I0 => din_a(0),
      I1 => din_b(0),
      CI => '0',
      O => sum(0),
      CO => carry_out(0)
   );

 EFX_ADD_inst_2 : EFX_ADD
   generic map (
      I0_POLARITY => 1,
      I1_POLARITY => 1
   )
   port map (
      I0 => din_a(1),
      I1 => din_b(1),
      CI => carry_out(0),
      O => sum(1),
      CO => carry_out(1)
   );

 EFX_ADD_inst_3 : EFX_ADD
   generic map (
      I0_POLARITY => 1,
      I1_POLARITY => 1
   )
   port map (
      I0 => din_a(2),
      I1 => din_b(2),
      CI => carry_out(1),
      O => sum(2)
   );

end architecture Behavioral;
```

EFX_FF

D Flip-flop with Clock Enable and Set/Reset Pin

The basic EFX_FF primitive is a D flip-flop with a clock enable and a set/reset pin that can be either asynchronous or synchronously asserted. You can positively or negatively trigger the clock, clock-enable and set/reset pins.

All input ports must be connected. If you do not use a flip-flop control port, connect it to ground or V_{CC}, depending on the polarity. The software issues a warning if a clock port is set to V_{CC} or ground.

EFX_FF Ports

Figure 9: EFX_FF Symbol

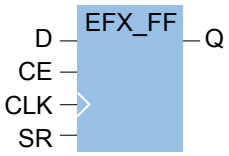


Table 7: EFX_FF Ports

Port	Direction	Description
D	Input	Input data.
CE	Input	Clock Enable.
CLK	Input	Clock.
SR	Input	Asynchronous/synchronous set/reset.
Q	Output	Output data.

EFX_FF Parameters

Table 8: EFX_FF Parameters

Parameter	Allowed Values	Description
CLK_POLARITY	0, 1	0 falling edge, 1 rising edge (default).
CE_POLARITY	0, 1	0 active low, 1 active high (default).
SR_POLARITY	0, 1	0 active low, 1 active high (default).
D_POLARITY	0, 1	0 inverting, 1 non-inverting (default).
SR_SYNC	0, 1	0 asynchronous (default), 1 synchronous.
SR_VALUE	0, 1	0 reset (default), 1 set.
SR_SYNC_PRIORITY	1	Reserved

EFX_FF Function

When the SR_SYNC parameter is asynchronous, the SR port overrides all other ports. When the SR_SYNC parameter is synchronous, the SR port is synchronous with the clock and higher priority than the CE port (the SR port takes effect even if CE is disabled).

Figure 10: EFX_FF Verilog HDL Instantiation

```
EFX_FF # (
    .CLK_POLARITY(1'b1),   // 0 falling edge, 1 rising edge
    .CE_POLARITY(1'b1),    // 0 active low, 1 active high
    .SR_POLARITY(1'b0),    // 0 active low, 1 active high
    .D_POLARITY(1'b1),     // 0 inverting, 1 non-inverting
    .SR_SYNC(1'b0),        // 0 asynchronous, 1 synchronous
    .SR_VALUE(1'b0)        // 0 reset, 1 set
) EFX_FF_inst (
    .Q(Q),         // FF output
    .D(D),         // D input
    .CE(CE),       // Clock-enable input
    .CLK(CLK),     // Clock input
    .SR(SR) // Set/reset input
);
```

Figure 11: EFX_FF VHDL Instantiation

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library efxphysicallib;
use efxphysicallib.efxcomponents.all;

entity D_FF_VHDL is
   port
   (
      clk : in std_logic;
      rst : in std_logic;
      ce  : in std_logic;
      d : in std_logic;
      q : out std_logic
   );
end entity D_FF_VHDL;

architecture Behavioral of D_FF_VHDL is
begin

 EFX_FF_inst : EFX_FF
   generic map (
      CLK_POLARITY => 1,
      CE_POLARITY => 1,
      SR_POLARITY => 1,
      D_POLARITY => 1,
      SR_SYNC => 1,
      SR_VALUE => 0,
      SR_SYNC_PRIORITY => 1
   )
   port map (
      D => d,
      CE => ce,
      CLK => clk,
      SR => rst,
      Q => q
   );

end architecture Behavioral;
```

EFX_RAM_5K

5 Kbit RAM Block

The EFX_RAM_5K primitive represents a configurable 5K bit RAM block that supports a variety of widths and depths. All inputs have programmable inversion, allowing positively or negatively triggered control signals.

The memory read and write ports have 8 modes (256 x 16, 512 x 8, 1024 x 4, 2048 x 2, 4096 x 1, 256 x 20, 512 x 10, 1024 x 5) for addressing the memory. The read and write ports support independently configured data widths.

Table 9: EFX_RAM_5K Allowed Read and Write Mode Combinations

		Write Mode							
	Memory Depth x Data Width	**256 x 16**	**512 x 8**	**1024 x 4**	**2048 x 2**	**4096 x 1**	**256 x 20**	**512 x 10**	**1024 x 5**
Read Mode	**256 x 16**	✓	✓	✓	✓	✓			
	512 x 8	✓	✓	✓	✓	✓			
	1024 x 4	✓	✓	✓	✓	✓			
	2048 x 2	✓	✓	✓	✓	✓			
	4096 x 1	✓	✓	✓	✓	✓			
	256 x 20						✓	✓	✓
	512 x 10						✓	✓	✓
	1024 x 5						✓	✓	✓

The following formula shows how the memory content is addressed for the different data widths:

[((ADDR + 1) * WIDTH) - 1 : (ADDR * WIDTH)]

You define the initial RAM content using INIT_N parameters. There are 20 INIT_N parameters and each parameter represents 256 bits of memory. The memory space covered by each INIT_N parameter uses the formula:

[((N+1) * 256) - 1 : (N * 256)]

When implementing an EFX_RAM_5K block:

- You must connect the RAM control ports (WCLK, WE, WCLKE, RCLK, and RE). If your design does not use these ports, connect them to ground or V_{CC} depending on their polarity. The software issues a warning if the read or write clock is connected to ground or VCC (except when implementing a ROM). The RAM contains an optional output register that improves t_{CO} at a cost of one latency stage. It uses the same clock signals as the read port, and is always enabled[1].
- You can only use the address lines that are valid in the particular mode, and you should connect all of them. Leave all other address lines unconnected. Connect required unused address lines to ground.

[1] Always enabled means the read data is always output one cycle after the address read, including the cycle after read-enable is disabled.

- Leave unused data lines unconnected.
- When implementing a ROM connect the WE, WCLK, and WCLKE to ground. Leave WDATA unconnected. Connect WADDR to ground or leave it unconnected based on the write mode you select. The write mode must be compatible with the read mode even though the write ports of the ROM are unused.

When you connect the same clock signal to the read and write clock ports, use the WRITE_MODE parameter to control the read port behavior when writing.

Note: If you use different clocks for the read and write clock ports, you must use READ_UNKNOWN.

EFX_RAM_5K Ports

Figure 12: EFX_RAM_5K Symbol

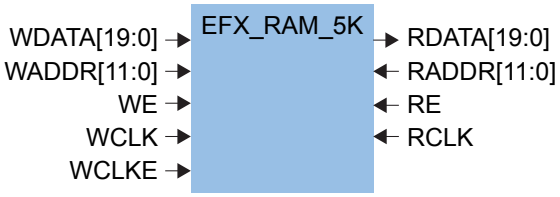


Table 10: EFX_RAM_5K Ports

Port Name	Direction	Description
WDATA[19:0]	Input	Write data
WADDR[11:0]	Input	Write address
WE	Input	Write enable
WCLK	Input	Write clock
WCLKE	Input	Write clock enable
RDATA[19:0]	Output	Read data
RADDR[11:0]	Input	Read address
RE	Input	Read enable
RCLK	Input	Read clock

EFX_RAM_5K Parameters

Table 11: EFX_RAM_5K Parameters

Every input port has programmable inversion support defined by <*port name*>_POLARITY.

Parameter Name	Allowed Values	Description
INIT_<*n*>	256 bit hexadecimal number	Initial RAM content (default = 0)
READ_WIDTH WRITE_WIDTH	16	256 x 16 (default)
	8	512 x 8
	4	1024 x 4
	2	2048 x 2
	1	4096 x 1
	20	256 x 20
	10	512 x 10
	5	1024 x 5
OUTPUT_REG	0, 1	0: disable output register (default) 1: enable output register
<*port name*>_POLARITY	0, 1	0: active low 1: active high (default)
WRITE_MODE	READ_FIRST, WRITE_FIRST, READ_UNKNOWN	When using the same clock for RCLK and WCLK, this parameter controls whether the read data is old or new. READ_FIRST–Old memory content is read. (default) WRITE_FIRST–Write data is passed to the read port. READ_UNKNOWN–Read and writes are unsynchronized, therefore, the results of the address can conflict.

EFX_RAM_5K Function

The EFXBRAM is physically implemented as a 256 x 20 memory array with decoder logic that maps to the read and write modes. The read and write ports are independent:

- Writes are guaranteed
- Read behavior depends on the WRITE_MODE value

The address at the physical memory array must not conflict.

Figure 13: EFX_RAM_5K Verilog HDL Instantiation

```
EFX_RAM_5K # (
   .READ_WIDTH(20),              // 20 256x20
   .WRITE_WIDTH(20),             // 20 256x20
   .OUTPUT_REG(1'b0),            // 1 add pipe-line read register
   .RCLK_POLARITY(1'b1),         // 0 falling edge, 1 rising edge
   .RE_POLARITY(1'b1),           // 0 active low, 1 active high
   .WCLK_POLARITY(1'b1),         // 0 falling edge, 1 rising edge
   .WE_POLARITY(1'b1),           // 0 active low, 1 active high
   .WCLKE_POLARITY(1'b1),        // 0 active low, 1 active high
   .WRITE_MODE("READ_FIRST"),    // Output "old" data
   .INIT_0(256'h0000000000000000000000000000000000000000000000000000000000000000),
   .INIT_1(256'h0000000000000000000000000000000000000000000000000000000000000000),
   .INIT_2(256'h0000000000000000000000000000000000000000000000000000000000000000),
   .INIT_3(256'h0000000000000000000000000000000000000000000000000000000000000000),
   .INIT_4(256'h0000000000000000000000000000000000000000000000000000000000000000),
   .INIT_5(256'h0000000000000000000000000000000000000000000000000000000000000000),
   .INIT_6(256'h0000000000000000000000000000000000000000000000000000000000000000),
   .INIT_7(256'h0000000000000000000000000000000000000000000000000000000000000000),
   .INIT_8(256'h0000000000000000000000000000000000000000000000000000000000000000),
   .INIT_9(256'h0000000000000000000000000000000000000000000000000000000000000000),
   .INIT_A(256'h0000000000000000000000000000000000000000000000000000000000000000),
   .INIT_B(256'h0000000000000000000000000000000000000000000000000000000000000000),
   .INIT_C(256'h0000000000000000000000000000000000000000000000000000000000000000),
   .INIT_D(256'h0000000000000000000000000000000000000000000000000000000000000000),
   .INIT_E(256'h0000000000000000000000000000000000000000000000000000000000000000),
   .INIT_F(256'h0000000000000000000000000000000000000000000000000000000000000000),
   .INIT_10(256'h0000000000000000000000000000000000000000000000000000000000000000),
   .INIT_11(256'h0000000000000000000000000000000000000000000000000000000000000000),
   .INIT_12(256'h0000000000000000000000000000000000000000000000000000000000000000),
   .INIT_13(256'h0000000000000000000000000000000000000000000000000000000000000000)
   ) EFX_RAM_5K_inst (
      .RDATA(RDATA),     // Read data output
      .RADDR(RADDR),     // Read address input
      .RCLK(RCLK),       // Read clock input
      .RE(RE),           // Read-enable input
      .WDATA(WDATA),     // Write data input
      .WADDR(WADDR),     // Write address input
      .WCLK(WCLK),       // Write clock input
      .WE(WE),           // Write-enable input
      .WCLKE(WCLKE)      // Write clock-enable input
);
```

Figure 14: EFX_RAM_5K VHDL Instantiation

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library efxphysicallib;
use efxphysicallib.efxcomponents.all;

entity rominit1i_5k is
   generic (
      ADD_SIZE : integer := 11;
      ADD_DEPTH : integer := 2**ADD_SIZE;
      DATA_DEPTH : integer := 2
   );
   port (
      RCLK : in std_logic;
      RADD : in std_logic_vector(ADD_SIZE-1 downto 0);
      RDATA : out std_logic_vector(DATA_DEPTH-1 downto 0)
   );
end entity rominit1i_5k;

architecture Behavioral of rominit1i_5k is
signal data_zeros : std_logic_vector(DATA_DEPTH-1 downto 0);
```

```
signal addr_zeros : std_logic_vector(ADD_SIZE-1 downto 0);
begin

   data_zeros <= (others => '0');
   addr_zeros <= (others => '0');

   MEM : EFX_RAM_5K
      generic map (
         READ_WIDTH => 2,
         WRITE_WIDTH => 2,
         WCLK_POLARITY => 1,
         WCLKE_POLARITY => 1,
         WE_POLARITY => 1,
         RCLK_POLARITY => 1,
         RE_POLARITY => 1,

         -- First segment of 512 (formed from 2301)
         INIT_0 => x"00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00",
         INIT_1 => x"00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00",
         INIT_2 => x"00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00",
         INIT_3 => x"00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00",

         -- Next segment 512 - 1024 (formed from 0123)
         INIT_4 => x"00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00",
         INIT_5 => x"00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00",
         INIT_6 => x"00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00",
         INIT_7 => x"00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00",

         -- Next segment 1024 - 1536
         INIT_8 => x"00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00",
         INIT_9 => x"00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00",
         INIT_A => x"00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00",
         INIT_B => x"00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00",

         -- Next segment 1024 - 2048
         INIT_C => x"00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00",
         INIT_D => x"00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00",
         INIT_E => x"00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00",
         INIT_F => x"00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00",

         INIT_10 => x"00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00",
         INIT_11 => x"00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00",
         INIT_12 => x"00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00",
         INIT_13 => x"00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00"
      )
      port map (
         WCLK => '0',
         WE => '0',
         WCLKE => '0',
         RCLK => RCLK,
         RE => '1',
         WADDR => addr_zeros,
         WDATA => data_zeros,
         RADDR => RADD,
         RDATA => RDATA
      );

end architecture Behavioral; -- rominit1i_5k
```

EFX_DPRAM_5K

5 Kbit True-Dual-Port RAM Block

The EFX_DPRAM_5K primitive represents a 5 Kbit true-dual-port RAM block that can be configured to support a variety of widths and depths. All inputs have programmable inversion capabilities, which allows you to trigger the control signals positively or negatively. To address the memory contents, you configure the memory A and B ports as 512 x 8, 1024 x 4, 2048 x 2, 4096 x 1, 512 x 10, or 1024 x 5. The read and write ports support independently configured data widths.

The true-dual-port RAM uses the same address bus for reading and writing on a port. Therefore, when a port has mixed widths, the software uses the widest address bus size to determine the address bus width. The direction (read or write) operating in the shallower address size ignores the address bus's LSB because they describe addresses that are outside the legal range for that mode. For example, for a 512 x 8 read and a 1024 x 4 write, the address is 10 bits wide to address all 1024 words being written. The write data is 4 bits wide and the read data is 8 bits wide. The true-dual-port RAM only uses the upper 9 bits of the address port during reading because it can only read 512 words from the memory.

Table 12: EFX_DPRAM_5K Allowed Read and Write Mode Combinations

		Write Mode					
	Memory Depth x Data Width	**512 x 8**	**1024 x 4**	**2048 x 2**	**4096 x 1**	**512 x 10** [2]	**1024 x 5** [2]
Read Mode	**512 x 8**	✓	✓	✓	✓		
	1024 x 4	✓	✓	✓	✓		
	2048 x 2	✓	✓	✓	✓		
	4096 x 1	✓	✓	✓	✓		
	512 x 10					✓	✓
	1024 x 5					✓	✓

The following formula shows how the memory content is addressed for the different data widths:

[((ADDR + 1) * WIDTH) - 1 : (ADDR * WIDTH)]

You define the initial RAM content through INIT_N parameters. Each INIT_N parameter represents 256 bits of memory; 20 parameters cover the 5K memory contents. The memory space covered by each INIT_N parameter uses this formula:

[((N+1) * 256) - 1 : (N * 256)]

When connecting the ports, use the following guidelines:

- You must connect the BRAM control ports (CLKA, WEA, CLKEA, CLKB, WEB, and CLKEB). Connect unused ports to GND or VCC depending on their polarity.
- If you want to disable a RAM port (A or B), disable the clock, write enable, clock enable, and address ports.
- WDATA can be disabled or disconnected.

[2] 5 Kbits only available in 512 x 10 and 1024 x 5modes.

- RDATA should be disconnected.

Note: Each BRAM output port contains an optional output register to improve t_{CO} at a cost of one stage of latency. It uses the same clock signals as the read port, and is always enabled[3].

When writing to a memory port, the WRITE_MODE_A/B parameters control the read port behavior:

You can only use the address lines that are valid for the mode you are using. For example, use only address bits ADDRA[8:0] if port A is in 512 x 8 mode.

All of the address lines for a mode should be connected. Connect unused, required address lines to GND. Leave all other address lines unconnected. For example, if the RAM port B is in 512 x 8 mode but is only implementing a 64 x 2 memory:

- ADDRB[11:9] are unconnected
- ADDRB[8:6] are connected to GND
- ADDRB[5:0] are used

Leave unused data lines unconnected. For example, if the RAM port A is in 512 x 8 mode, but is only implementing a 64 x 2 memory:

- WDATAA[19:2] and RDATAA[19:2] are unused and unconnected
- WDATAA[1:0] and RDATAA[1:0] are used and connected

When implementing a ROM:

- Connect WEA and WEB to GND
- Leave WDATAA and WDATAB unconnected or disabled

EFX_DPRAM_5K Ports

Figure 15: EFX_DPRAM_5K Symbol

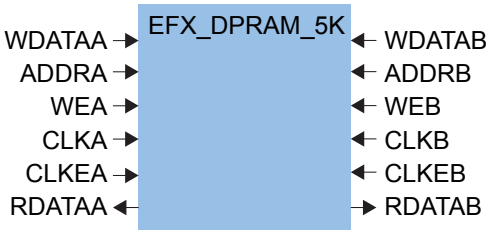


Table 13: EFX_DPRAM_5K Ports

Port Name	Direction	Description
WDATAA[9:0] WDATAB[9:0]	Input	Write data port A/B
ADDRA[11:0] ADDRB[11:0]	Input	Address port A/B
WEA WEB	Input	Write enable port A/B
CLKA CLKB	Input	Clock port A/B

[3] Always enabled means the read data is always output one cycle after the address read, including the cycle after read-enable is disabled.

Port Name	Direction	Description
CLKEA CLKEB	Input	Clock enable port A/B
RDATA[9:0] RDATB[9:0]	Output	Read data port A/B

EFX_DPRAM_5K Parameters

Table 14: EFX_DPRAM_5K Parameters

Every input port has programmable inversion support defined by <*port name*>_POLARITY.

Parameter Name	Allowed Values	Description
INIT_<*n*>	256 bit hexadecimal number	Initial RAM content (default = 0)
READ_WIDTH_A READ_WIDTH_B WRITE_WIDTH_A WRITE_WIDTH_B	8	512 x 8 (default)
	4	1024 x 4
	2	2048 x 2
	1	4096 x 1
	10	512 x 10
	5	1024 x 5
WRITE_MODE_A WRITE_MODE_B	READ_FIRST, WRITE_FIRST, NO_CHANGE	Controls the read port behavior READ_FIRST–Old memory content is read. (default) WRITE_FIRST–Write data is passed to the read port. NO_CHANGE–Previously read data is held.
OUTPUT_REG_A OUTPUT_REG_B	0, 1	0: disable output register (default) 1: enable output register
<*port name*>_POLARITY	0, 1	0: active low 1: active high (default)

EFX_DPRAM_5K Function

The BRAM is physically implemented as a 256 x 20 memory array with decoder logic to map to the read and write modes. The A and B ports are independent and the behavior is undefined when addresses conflict. The address at the physical memory array must not conflict.

Figure 16: EFX_DPRAM_5K Verilog HDL Instantiation

```
EFX_DPRAM_5K # (
    .READ_WIDTH_A(8),               // 8 512x8
    .WRITE_WIDTH_A(8),              // 8 512x8
    .OUTPUT_REG_A(1'b0),            // 1 add pipe-line read register
    .CLKA_POLARITY(1'b1),           // 0 falling edge, 1 rising edge
    .WEA_POLARITY(1'b1),            // 0 active low, 1 active high
    .CLKEA_POLARITY(1'b1),          // 0 falling edge, 1 rising edge
    .WRITE_MODE_A("READ_FIRST"),    // Output "old" data
    .READ_WIDTH_B(8),               // 8 512x8
    .WRITE_WIDTH_B(8),              // 8 512x8
    .OUTPUT_REG_B(1'b0),            // 1 add pipe-line read register
    .CLKB_POLARITY(1'b1),           // 0 falling edge, 1 rising edge
    .WEB_POLARITY(1'b1),            // 0 active low, 1 active high
```

```
   .CLKEB_POLARITY(1'b1),          // 0 falling edge, 1 rising edge
   .WRITE_MODE_B("READ_FIRST"),    // Output "old" data
   .INIT_0(256'h0000000000000000000000000000000000000000000000000000000000000000),
   .INIT_1(256'h0000000000000000000000000000000000000000000000000000000000000000),
   .INIT_2(256'h0000000000000000000000000000000000000000000000000000000000000000),
   .INIT_3(256'h0000000000000000000000000000000000000000000000000000000000000000),
   .INIT_4(256'h0000000000000000000000000000000000000000000000000000000000000000),
   .INIT_5(256'h0000000000000000000000000000000000000000000000000000000000000000),
   .INIT_6(256'h0000000000000000000000000000000000000000000000000000000000000000),
   .INIT_7(256'h0000000000000000000000000000000000000000000000000000000000000000),
   .INIT_8(256'h0000000000000000000000000000000000000000000000000000000000000000),
   .INIT_9(256'h0000000000000000000000000000000000000000000000000000000000000000),
   .INIT_A(256'h0000000000000000000000000000000000000000000000000000000000000000),
   .INIT_B(256'h0000000000000000000000000000000000000000000000000000000000000000),
   .INIT_C(256'h0000000000000000000000000000000000000000000000000000000000000000),
   .INIT_D(256'h0000000000000000000000000000000000000000000000000000000000000000),
   .INIT_E(256'h0000000000000000000000000000000000000000000000000000000000000000),
   .INIT_F(256'h0000000000000000000000000000000000000000000000000000000000000000),
   .INIT_10(256'h0000000000000000000000000000000000000000000000000000000000000000),
   .INIT_11(256'h0000000000000000000000000000000000000000000000000000000000000000),
   .INIT_12(256'h0000000000000000000000000000000000000000000000000000000000000000),
   .INIT_13(256'h0000000000000000000000000000000000000000000000000000000000000000)
   ) EFX_DPRAM_5K_inst (
      .RDATAA(RDATAA),    // Read data output A
      .ADDRA(ADDRA),      // Address input A
      .CLKA(CLKA),        // Clock input A
      .CLKEA(CLKEA),      // Clock-enable input A
      .WEA(WEA),          // Write-enable input A
      .WDATAA(WDATAA),    // Write data input A
      .RDATAB(RDATAB),    // Read data output B
      .ADDRB(ADDRB),      // Address input B
      .CLKB(CLKB),        // Clock input B
      .CLKEB(CLKEB),      // Clock-enable input B
      .WEB(WEB),          // Write-enable input B
      .WDATAB(WDATAB)     // Write data input B
);
```

Figure 17: EFX_DPRAM_5K VHDL Instantiation

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
library efxphysicallib;
use efxphysicallib.efxcomponents.all;

entity ram512x8_tdp_rbwi_VHDL is
   generic (
      AWIDTH : integer := 9;
      DWIDTH : integer := 8
   );
   port
   (
      wdataA, wdataB : in std_logic_vector(DWIDTH-1 downto 0);
      addrA, addrB : in std_logic_vector(AWIDTH-1 downto 0);
      clkA, weA : in std_logic;
      clkB, weB : in std_logic;
      rdataA, rdataB : out std_logic_vector(DWIDTH-1 downto 0)
   );
end entity ram512x8_tdp_rbwi_VHDL;

architecture Behavioral of ram512x8_tdp_rbwi_VHDL is
   constant INIT_0 : unsigned(255 downto 0) :=
 x"0000000000000000000000000000000000000000000000000000000000000000";
   constant INIT_1 : unsigned(255 downto 0) :=
 x"0000000000000000000000000000000000000000000000000000000000000000";
   constant INIT_2 : unsigned(255 downto 0) :=
 x"0000000000000000000000000000000000000000000000000000000000000000";
   constant INIT_3 : unsigned(255 downto 0) :=
 x"0000000000000000000000000000000000000000000000000000000000000000";
   constant INIT_4 : unsigned(255 downto 0) :=
 x"0000000000000000000000000000000000000000000000000000000000000000";
   constant INIT_5 : unsigned(255 downto 0) :=
 x"0000000000000000000000000000000000000000000000000000000000000000";
   constant INIT_6 : unsigned(255 downto 0) :=
 x"0000000000000000000000000000000000000000000000000000000000000000";
   constant INIT_7 : unsigned(255 downto 0) :=
 x"0000000000000000000000000000000000000000000000000000000000000000";
   constant INIT_8 : unsigned(255 downto 0) :=
 x"0000000000000000000000000000000000000000000000000000000000000000";
   constant INIT_9 : unsigned(255 downto 0) :=
 x"0000000000000000000000000000000000000000000000000000000000000000";
   constant INIT_A : unsigned(255 downto 0) :=
 x"0000000000000000000000000000000000000000000000000000000000000000";
```

```
   constant INIT_B : unsigned(255 downto 0) :=
 x"0000000000000000000000000000000000000000000000000000000000000000";
   constant INIT_C : unsigned(255 downto 0) :=
 x"0000000000000000000000000000000000000000000000000000000000000000";
   constant INIT_D : unsigned(255 downto 0) :=
 x"0000000000000000000000000000000000000000000000000000000000000000";
   constant INIT_E : unsigned(255 downto 0) :=
 x"0000000000000000000000000000000000000000000000000000000000000000";
   constant INIT_F : unsigned(255 downto 0) :=
 x"0000000000000000000000000000000000000000000000000000000000000000";
   constant INIT_10 : unsigned(255 downto 0) :=
 x"0000000000000000000000000000000000000000000000000000000000000000";
   constant INIT_11 : unsigned(255 downto 0) :=
 x"0000000000000000000000000000000000000000000000000000000000000000";
   constant INIT_12 : unsigned(255 downto 0) :=
 x"0000000000000000000000000000000000000000000000000000000000000000";
   constant INIT_13 : unsigned(255 downto 0) :=
 x"0000000000000000000000000000000000000000000000000000000000000000";
begin

   ram : EFX_DPRAM_5K
     generic map (
       READ_WIDTH_A => DWIDTH,
       WRITE_WIDTH_A => DWIDTH,
       READ_WIDTH_B => DWIDTH,
       WRITE_WIDTH_B => DWIDTH,
       WRITE_MODE_A => "READ_FIRST",
       WRITE_MODE_B => "READ_FIRST",
       INIT_0 => INIT_0,
       INIT_1 => INIT_1,
       INIT_2 => INIT_2,
       INIT_3 => INIT_3,
       INIT_4 => INIT_4,
       INIT_5 => INIT_5,
       INIT_6 => INIT_6,
       INIT_7 => INIT_7,
       INIT_8 => INIT_8,
       INIT_9 => INIT_9,
       INIT_A => INIT_A,
       INIT_B => INIT_B,
       INIT_C => INIT_C,
       INIT_D => INIT_D,
       INIT_E => INIT_E,
       INIT_F => INIT_F,
       INIT_10 => INIT_10,
       INIT_11 => INIT_11,
       INIT_12 => INIT_12,
       INIT_13 => INIT_13
     )
     port map (
       WDATAA => wdataA,
       ADDRA => addrA,
       CLKA => clkA,
       CLKEA => '1',
       WEA => weA,
       RDATAA => rdataA,
       WDATAB => wdataB,
       ADDRB => addrB,
       CLKB => clkB,
       CLKEB => '1',
       WEB => weB,
       RDATAB => rdataB
     );

end architecture Behavioral;
```

EFX_MULT

18 x 18 Multiplier

The EFX_MULT logical block represents a signed integer multiplier with optional input and output registers. The Quantum® fabric supports an 18 x 18 multiplier. All inputs have programmable inversion allowing positively or negatively triggered control signals.

When implementing an EFX_MULT block:

- If emulating an unsigned multiplier, set the MSB bit to ground.
- You must connect the multiplier control ports (CLK, CEA, RSTA, CEB, RSTB, CEO, and RSTO). Connect unused ports to V_{CC} or ground depending on their polarity. The software issues a warning if the clock is connected to V_{CC} or ground and the design does not bypass the registers.
- You must connect all data lines. Connect unused, required data lines to a sign bit. For example, when implementing a signed 8 x 8 multiplier, the software uses bits A[7:0]. Connect data bits A[17:8] to the signal driving A[7].

The command-line option `--max_mult` controls the maximum number of multiplier blocks that the software can infer.

- `-1` is auto and the tool infers as many blocks as appropriate
- `0` infers none
- *n* infers no more than *n* blocks

EFX_MULT Ports

Figure 18: EFX_MULT Symbol

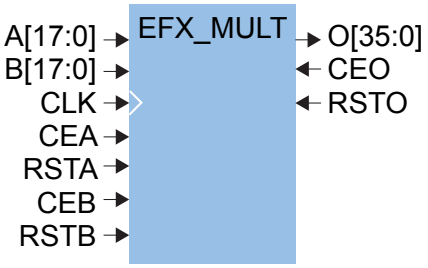


Table 15: EFX_MULT Ports

Port Name	Direction	Description
A[17:0]	Input	Operand A
B[17:0]	Input	Operand B
CLK	Input	Clock
CEA	Input	Clock enable A
RSTA	Input	Set/reset A
CEB	Input	Clock enable B
RSTB	Input	Set/reset B
O[35:0]	Output	Multiplier output

Port Name	Direction	Description
CEO	Input	Clock enable O
RSTO	Input	Set/reset O

EFX_MULT Parameters

Table 16: EFX_MULT Parameters

Every input port has programmable inversion support defined by <*port name*>_POLARITY.

Parameter Name	Allowed Values	Description
WIDTH	18	18 x 18 multiplier
A_REG	0, 1	0: Disable A registers (default) 1: Enable A registers
B_REG	0, 1	0: Disable B registers (default) 1: Enable B registers
O_REG	0, 1	0: Disable output registers (default) 1: Enable output registers
RSTA_SYNC	0, 1	0: Asynchronous (default) 1: Synchronous on A registers
RSTA_VALUE	0, 1	0: Reset (default) 1: Set on A register
RSTB_SYNC	0, 1	0: Asynchronous (default) 1: Synchronous on B registers
RSTB_VALUE	0, 1	0: Reset (default) 1: Set on B register
RSTO_SYNC	0, 1	0: Asynchronous (default) 1: Synchronous on output registers
RSTO_VALUE	0, 1	0: Reset (default) 1: Set on output register
<*port name*>_POLARITY	0, 1	0: Active low 1: Active high (default)
SR_SYNC_PRIORITY	0	Reserved

EFX_MULT Function

The EFX_MULT is a signed integer multiplier.

Figure 19: EFX_MULT Verilog HDL Instantiation

```
EFX_MULT # (
   .WIDTH(18),
   .A_REG(1),
   .B_REG(1),
   .O_REG(1),
   .CLK_POLARITY(1'b1),  // 0 falling edge, 1 rising edge
   .CEA_POLARITY(1'b1),  // 0 falling edge, 1 rising edge
   .RSTA_POLARITY(1'b0), // 0 falling edge, 1 rising edge
   .RSTA_SYNC(1'b0),     // 0 aynchronous, 1 synchronous
   .RSTA_VALUE(1'b0),    // 0 reset, 1 set
   .CEB_POLARITY(1'b1),  // 0 falling edge, 1 rising edge
   .RSTB_POLARITY(1'b0), // 0 falling edge, 1 rising edge
```

```
   .RSTB_SYNC(1'b0),     // 0 aynchronous, 1 synchronous
   .RSTB_VALUE(1'b0),    // 0 reset, 1 set
   .CEO_POLARITY(1'b1),  // 0 falling edge, 1 rising edge
   .RSTO_POLARITY(1'b0), // 0 falling edge, 1 rising edge
   .RSTO_SYNC(1'b0),     // 0 aynchronous, 1 synchronous
   .RSTO_VALUE(1'b0)     // 0 reset, 1 set
   ) mult (
      .CLK(CLK),
      .CEA(CEA),
      .RSTA(SRA),
      .CEB(CEB),
      .RSTB(SRB),
      .CEO(CEO),
      .RSTO(SRO),
      .A(A),
      .B(B),
      .O(O)
);
```

Figure 20: EFX_MULT VHDL Instantiation

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library efxphysicallib;
use efxphysicallib.efxcomponents.all;

entity mult_s18xs18_ffesr2i_VHDL is
   port
   (
      clk : in std_logic;
      cea, clra : in std_logic;
      ceb, clrb : in std_logic;
      cex, clrx : in std_logic;
      a,b : in std_logic_vector(17 downto 0);
      x : out std_logic_vector(35 downto 0)
   );
end entity mult_s18xs18_ffesr2i_VHDL;

architecture Behavioral of mult_s18xs18_ffesr2i_VHDL is
begin

 EFX_MULT_inst : EFX_MULT
   generic map (
      WIDTH => 18,
      A_REG => 1,
      B_REG => 1,
      O_REG => 1,
      RSTA_SYNC => 1,
      RSTB_SYNC => 1,
      RSTO_SYNC => 1,
      SR_SYNC_PRIORITY => 0
   )
   port map (
      CLK => clk,
      CEA => cea,
      RSTA => clra,
      CEB => ceb,
      RSTB => clrb,
      CEO => cex,
      RSTO => clrx,
      A => a,
      B => b,
      O => x
   );

end architecture Behavioral;
```

EFX_GBUFCE

Global Clock Buffer

The EFX_GBUFCE logic block represents the global clock buffer driving the global clock network. The CE port gates the clock and is active high.

You must connect all EFX_GBUFCE input ports. If you do not use a port, connect it to ground or V_{CC} depending on its polarity. The software issues an error if the clock input I is set to V_{CC} or ground.

Synthesis creates EFX_GBUFCE logical blocks for every clock source in the user netlist. This implementation allows the place-and-route tools to identify the clock sources that should be placed on pins capable of being clocks or to route core-generated clocks.

EFX_GBUFCE Ports

Figure 21: EFX_GBUFCE Symbol

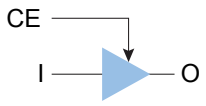


Table 17: EFX_GBUFCE Ports

Port Name	Direction	Description
I	Input	Input data
CE	Input	Clock enable
O	Output	Output data

EFX_GBUFCE Parameters

Table 18: EFX_GBUFCE Parameters

Parameter Name	Allowed Values	Description
CE_POLARITY	0, 1	0 active low, 1 active high (default)

EFX_GBUFCE Function

The function table assumes all inputs are active-high polarity.

Table 19: EFX_GBUFCE Function

Inputs		Output
CE	**I**	**O**
0	X	0
1	0	0
1	1	1

Figure 22: EFX_GBUFCE Verilog HDL Instantiation

```
EFX_GBUFCE # (
   .CE_POLARITY(1'b1)   // 0 active low, 1 active high
) EFX_GBUFCE_inst (
   .O(O),               // Clock output to global clock network
   .I(I),               // Clock input
   .CE(CE)              // Clock gate
);
```

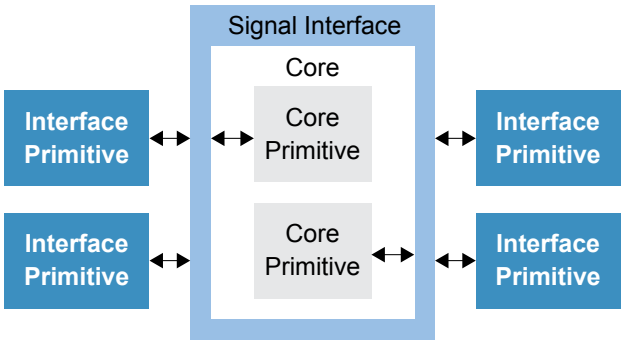
Figure 23: EFX_GBUFCE VHDL Instantiation

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library efxphysicallib;
use efxphysicallib.efxcomponents.all;

entity gbufce_i_VHDL is
   port
   (
      clk, d, ce : in std_logic;
      q : out std_logic
   );
end gbufce_i_VHDL;

architecture behavioral of gbufce_i_VHDL is
signal clknet : std_logic;
begin

  dut : EFX_GBUFCE
    port map (
       CE => ce,
       I => clk,
       O => clknet
    );

  ffx : EFX_FF
    port map (
       Q => q,
       D => d,
       CLK => clknet,
       CE => '1',
       SR => '0'
    );

end behavioral;
```

Interface Primitives

The interface primitives represent the functionality of the blocks that interface between the core and, for some primitives, the package pins (or pads). These primitives are the essential building blocks for the interfaces. They connect to the core primitives through a signal interface.

Figure 24: Interface Primitives



Important: The Efinity software v2025.1 only supports a subset of the possible interface primitives. Additional primitives will be supported in upcoming releases.

Table 20: Primitive Support by Version

Efinity Version	Supported Primitives
2024.2 (full release)	EFX_IBUF, EFX_OBUF, EFX_IO_BUF, EFX_CLKOUT, EFX_IREG, EFX_OREG, EFX_IOREG, EFX_IDDIO, EFX_ODDIO, EFX_JTAG_CTRL, EFX_JTAG_V1 EFX_GPIO_V1, EFX_GPIO_V2, EFX_PLL_V1, EFX_PLL_V2, OSC_V1
2025.1 (full release)	All primitives supported in v2024.2. EFX_LVDS_RX_V1, EFX_LVDS_TX_V1

EFX_IBUF

Single-Ended Input Buffer

The input buffer receives a signal from the GPIO input pad.

EFX_IBUF Ports

Figure 25: EFX_IBUF Symbol

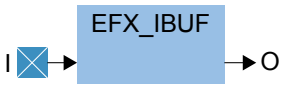


Table 21: EFX_ Ports

Port	Direction	Description
I	Input	GPIO pad used for input only.
O	Output	Output connection to the core.

EFX_IBUF Parameters

Table 22: EFX_IBUF Parameters

Parameter	Allowed Values	Description
PULL_OPTION	NONE WEAK_PULLUP WEAK_PULLDOWN	The type of pullup used on the input. NONE: Disable any internal pull-up or pull-down resistor (Default) WEAK_PULLUP: Enable the weak pull-up resistor WEAK_PULLDOWN: Enable the weak pull-down resistor

EFX_IBUF Function

Use these examples to instantiate the buffer.

Figure 26: EFX_IBUF Verilog HDL Instantiation

```
EFX_IBUF # (
   .PULL_OPTION("NONE")  // "NONE" (Default), "WEAK_PULLUP" , "WEAK_PULLDOWN"
) EFX_IBUF_inst (
   .I(I), // Buffer input (connect to top-level port)
   .O(O)  // Buffer output
);
```

Figure 27: EFX_IBUF VHDL Instantiation

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library efxphysicallib;
use efxphysicallib.efxcomponents.all
entity EFX_IBUF_VHDL is
port ( i : in  std_logic;
       o : out std_logic
);
end entity EFX_IBUF_VHDL;
architecture Behavioral of EFX_IBUF_VHDL is
 begin
 EFX_IBUF_inst : EFX_IBUF
   generic map (
      PULL_OPTION => "NONE"
   )
   port map (
      I => i,
      O => o
   );
 end architecture Behavioral;
```

EFX_OBUF

Single-Ended Output Buffer

This simple output buffer sends a signal to the GPIO output pad. If you need an output enable signal, use the EFX_IO_BUF on page 34 primitive and do not enable the optional input signal.

EFX_OBUF Ports

Figure 28: EFX_OBUF Symbol

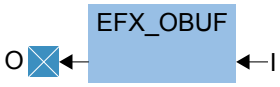


Table 23: EFX_OBUF Ports

Port	Direction	Description
I	Input	Input connection from the core.
O	Output	GPIO pad used as the output.

EFX_OBUF Parameters

None.

EFX_OBUF Function

Use these examples to instantiate the output buffer.

Figure 29: EFX_OBUF Verilog HDL Instantiation

```
EFX_OBUF EFX_OBUF_inst (
   .I(I),  // Buffer input
   .O(O)   // Buffer output (connect to top-level port)
);
```

Figure 30: EFX_OBUF VHDL Instantiation

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library efxphysicallib;
use efxphysicallib.efxcomponents.all
entity EFX_OBUF_VHDL is
port ( i : in  std_logic;
       o : out std_logic
);
end entity EFX_OBUF_VHDL;
architecture Behavioral of EFX_OBUF_VHDL is
 begin
 EFX_OBUF_inst : EFX_OBUF
   port map (
      I => i,
      O => o
   );
 end architecture Behavioral;
```

EFX_IO_BUF

Single-Ended Bi-Directional Buffer

This buffer sends and receives signals to and from the GPIO pad in inout mode. Use this primitive if you need an output enable.

EFX_IO_BUF Ports

Figure 31: EFX_IO_BUF Symbol

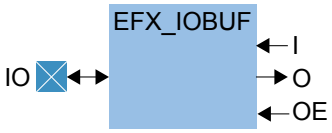


Table 24: EFX_IO_BUF Ports

Port	Direction	Description
I	Input	Output connection from the core.
O	Output	Optional input connection to the core.
OE	Input	Output enable connection from the core.
IO	Inout	GPIO pad used in inout mode.

EFX_IO_BUF Parameters

Table 25: EFX_IO_BUF Parameters

Parameter	Allowed Values	Description
PULL_OPTION	NONE WEAK_PULLUP WEAK_PULLDOWN	The type of pullup used on the input. NONE: Disable any internal pull-up or pull-down resistor (Default) WEAK_PULLUP: Enable the weak pull-up resistor WEAK_PULLDOWN: Enable the weak pull-down resistor

EFX_IO_BUF Function

Use these example to instantiate the I/O buffer.

Figure 32: EFX_IO_BUF Verilog HDL Instantiation

```
EFX_IO_BUF# (
   .PULL_OPTION("NONE")  // "NONE" (Default), "WEAK_PULLUP" , "WEAK_PULLDOWN"
) EFX_IO_BUF_inst (
   .I(I),   // Buffer input
   .OE(OE), // Enable output, high=output, low=input
   .O(O),   // Buffer output
   .IO(IO)  // Buffer inout port (Connect to top-level port)
);
```

Figure 33: EFX_IO_BUF VHDL Instantiation

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library efxphysicallib;
use efxphysicallib.efxcomponents.all
entity EFX_IO_BUF_VHDL is
port ( i : in  std_logic;
       oe : in std_logic);
       o : out std_logic;
       io: inout std_logic
);
end entity EFX_IO_BUF_VHDL ;
architecture Behavioral of EFX_IO_BUF_VHDL is
 begin
 EFX_IO_BUF_inst : EFX_IO_BUF
   generic map (
      PULL_OPTION => "NONE"
   )
   port map (
      I => i,
      OE => oe,
      o => o,
      io => io
   );
 end architecture Behavioral;
```

EFX_CLKOUT

Clock Output Buffer

Use this primitive when you want to output a clock signal to the GPIO pad.

EFX_CLKOUT Ports

Figure 34: EFX_CLKOUT Symbol

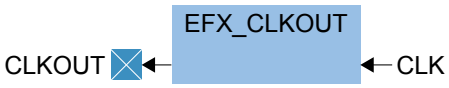


Table 26: EFX_CLKOUT Ports

Port	Direction	Description
CLK	Input	Clock signal from the core.
CLKOUT	Output	GPIO pad used in clkout mode.

EFX_CLKOUT Parameters

Table 27: EFX_CLKOUT Parameters

Parameter	Allowed Values	Description
IS_CLK_INVERTED	0, 1	Specify whether the output clock is inverted. 0: Not inverted (default). 1: Inverted.

EFX_CLKOUT Function

Use these examples to instantiate the clock output signal.

Figure 35: EFX_CLKOUT Verilog HDL Instantiation

```
EFX_CLKOUT# (
   .IS_CLK_INVERTED(0)     // 0 = No inversion (Default)
                           // 1 = Output clock signal is inverted
) EFX_CLKOUT_inst (
   .CLK(CLK),              // Clock input
   .CLKOUT(CLKOUT)         // Clock output (Connect to top-level port)
);
```

Figure 36: EFX_CLKOUT VHDL Instantiation

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library efxphysicallib;
use efxphysicallib.efxcomponents.all
entity EFX_CLKOUT_VHDL is
port ( clk : in  std_logic;
       clkout : out std_logic
);
end entity EFX_CLKOUT_VHDL ;
architecture Behavioral of EFX_CLKOUT_VHDL is
 begin
 EFX_CLKOUT_inst : EFX_CLKOUT
   generic map (
      IS_CLK_INVERTED=> 0
   )
   port map (
      CLK => clk,
      CLKOUT => clkout
   );
 end architecture Behavioral;
```

EFX_IREG

Single-Ended Input Register

This primitive is a registered input from the GPIO input pad.

EFX_IREG Ports

Figure 37: EFX_IREG Symbol

Table 28: EFX_IREG Ports

Port	Direction	Description
I	Input	GPIO pad used in input mode.
CLK	Input	Clock signal from the core.
O	Output	Output connection to the core.

EFX_IREG Parameters

Table 29: EFX_IREG Parameters

Parameter	Allowed Values	Description
IS_CLK_INVERTED	0, 1	Specify whether the clock is inverted. 0: Not inverted, data capture on rising edge (default). 1: Inverted, data capture on falling edge.
PULL_OPTION	NONE WEAK_PULLUP WEAK_PULLDOWN	The type of pullup used on the input. NONE: Disable any internal pull-up or pull-down resistor (Default) WEAK_PULLUP: Enable the weak pull-up resistor WEAK_PULLDOWN: Enable the weak pull-down resistor

EFX_IREG Function

Use these examples to instantiate the registered input primitive.

Figure 38: EFX_IREG Verilog HDL Instantiation

```
EFX_IREG # (
   .PULL_OPTION("NONE"), // "NONE" (Default), "WEAK_PULLUP" , "WEAK_PULLDOWN"
   .IS_CLK_INVERTED(0)   // 0 rising edge, 1 falling edge
) EFX_IREG_inst (
   .I(I),       // Data Input (Connect to top-level port)
   .CLK(CLK),   // Clock Input
   .O(O)        // Register Output
);
```

Figure 39: EFX_IREG VHDL Instantiation

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library efxphysicallib;
use efxphysicallib.efxcomponents.all
entity EFX_IREG_VHDL is
port ( i : in std_logic;
       clk : in std_logic;
       o : out std_logic
);
end entity EFX_IREG_VHDL;
architecture Behavioral of EFX_IREG_VHDL is
 begin
 EFX_IREG_inst : EFX_IREG
   generic map (
      PULL_OPTION => "NONE",
      IS_CLK_INVERTED => 0
   )
   port map (
      I => i,
      CLK => clk,
      O => o
   );
 end architecture Behavioral;
```

EFX_OREG

Single-Ended Output Register

This primitive is a registered output to the GPIO output pad.

EFX_OREG Ports

Figure 40: EFX_OREG Symbol

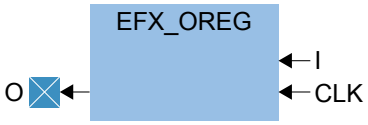


Table 30: EFX_OREG Ports

Port	Direction	Description
I	Input	Input connection from the core.
CLK	Input	Clock signal from the core.
O	Output	GPIO pad used in output mode.

EFX_OREG Parameters

Table 31: EFX_OREG Parameters

Parameter	Allowed Values	Description
IS_CLK_INVERTED	0, 1	Specify whether the clock is inverted. 0: Not inverted, data capture on rising edge (default). 1: Inverted, data capture on falling edge.

EFX_OREG Function

Use these examples to instantiate the registered output primitive.

Figure 41: EFX_OREG Verilog HDL Instantiation

```
EFX_OREG # (
   .IS_CLK_INVERTED(0)  // 0 rising edge, 1 falling edge
) EFX_OREG_inst (
   .I(I),       // Data Input
   .CLK(CLK),   // Clock Input
   .O(O)        // Register Output (Connect to top-level port)
);
```

Figure 42: EFX_OREG VHDL Instantiation

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library efxphysicallib;
use efxphysicallib.efxcomponents.all
entity EFX_OREG_VHDL is
port ( i : in std_logic;
       clk : in std_logic;
       o : out std_logic
);
end entity EFX_OREG_VHDL;
architecture Behavioral of EFX_OREG_VHDL is
 begin
 EFX_OREG_inst : EFX_OREG
   generic map (
      IS_CLK_INVERTED => 0
   )
   port map (
      I => i,
      CLK => clk,
      O => o
   );
 end architecture Behavioral;
```

EFX_IOREG

Single-Ended Bi-Directional Register

This primitive is a registered bi-directional from the GPIO inout pad.

EFX_IOREG Ports

Figure 43: EFX_IOREG Symbol

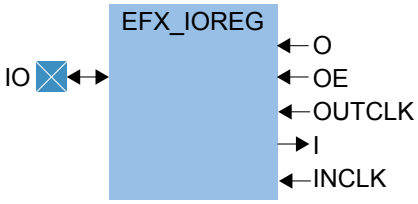


Table 32: EFX_IOREG Ports

Port	Direction	Description
O	Input	Output connection from the core.
OE	Input	Output enable connection from the core.
OUTCLK	Input	Clock signal from the core.
I	Output	Input connection to the core.
INCLK	Input	Clock signal form the core.
IO	Bi-directional	Bi-drectional signal to/from the GPIO pad in inout mode.

EFX_IOREG Parameters

Table 33: EFX_IOREG Parameters

Parameter	Allowed Values	Description
IS_INCLK_INVERTED	0, 1	Specify whether the input clock is inverted. 0: Not inverted, data capture on rising edge (default). 1: Inverted, data capture on falling edge.
IS_OUTCLK_INVERTED	0, 1	Specify whether the output clock is inverted. 0: Not inverted, data capture on rising edge (default). 1: Inverted, data capture on falling edge.
PULL_OPTION	NONE WEAK_PULLUP WEAK_PULLDOWN	The type of pullup used on the input. NONE: Disable any internal pull-up or pull-down resistor (Default) WEAK_PULLUP: Enable the weak pull-up resistor WEAK_PULLDOWN: Enable the weak pull-down resistor

EFX_IOREG Function

Use these examples to instantiate the registered inout primitive.

Figure 44: EFX_IOREG Verilog HDL Instantiation

```
EFX_IOREG # (
   .PULL_OPTION("NONE"),   // "NONE" (Default), "WEAK_PULLUP" ,
 "WEAK_PULLDOWN"
   .IS_INCLK_INVERTED(0),  // 0 (Default), 1 to invert the INCLK clock signal
   .IS_OUTCLK_INVERTED(0)  // 0 (Default), 1 to invert the OUTCLK clock
 signal
) EFX_IOREG_inst (
   .I(I),              // Register input
   .INCLK(INCLK),      // Input Clock
   .OE(OE),            // Enable output, high=output mode, low=input mode
   .O(O)               // Register output
   .OUTCLK(OUTCLK),    // Output Clock
   .IO(O)              // Connect to top-level inout port
);
```

Figure 45: EFX_IOREG VHDL Instantiation

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library efxphysicallib;
use efxphysicallib.efxcomponents.all
entity EFX_IOREG_VHDL is
port ( i : in std_logic;
       inclk : in std_logic;
       oe : in std_logic;
       o : out std_logic;
       outclk : out std_logic;
       io : inout std_logic
);
end entity EFX_IOREG_VHDL;
architecture Behavioral of EFX_IOREG_VHDL is
 begin
 EFX_IOREG_inst : EFX_IOREG
   generic map (
      PULL_OPTION => "NONE",
      IS_INCLK_INVERTED => 0,
      IS_OUTCLK_INVERTED => 0
   )
   port map (
      I => i,
      INCLK => inclk,
      OE => oe,
      OUTCLK => outclk,
      O => o,
      IO => io
   );
 end architecture Behavioral;
```

EFX_IDDIO

Input Double Data I/O Register

This primitive is for the double data I/O (DDIO) register from the GPIO input. The DDIO register captures data on both positive and negative clock edges. The core receives 2 bit wide data from the interface.

EFX_IDDIO Ports

Figure 46: EFX_IDDIO Symbol

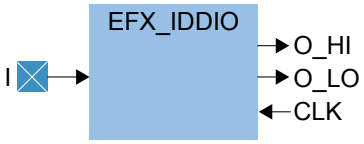


Table 34: EFX_IDDIO Ports

Port	Direction	Description
O_HI	Output	Output connection to the core. Captures pad data on the rising clock edge.
O_LO	Output	Output connection to the core. Captures pad data on the falling clock edge.
CLK	Input	Clock signal from the core.
I	Input	GPIO pad in input mode.

EFX_IDDIO Parameters

Table 35: EFX_IDDIO Parameters

Parameter	Allowed Values	Description
IS_CLK_INVERTED	0, 1	Specify whether the clock is inverted. 0: Not inverted, data capture on rising edge (default). 1: Inverted, data capture on falling edge.
PULL_OPTION	NONE WEAK_PULLUP WEAK_PULLDOWN	The type of pullup used on the input. NONE: Disable any internal pull-up or pull-down resistor (Default) WEAK_PULLUP: Enable the weak pull-up resistor WEAK_PULLDOWN: Enable the weak pull-down resistor
MODE	DDIO DDIO_RESYNC	The data synchronization mode. Refer to "Double Data I/O" in the data sheet for timing diagrams. DDIO: Normal mode. (Default) DDIO_RESYNC: Resync mode.

EFX_IDDIO Function

These examples show how to instantiate the input DDIO register.

Figure 47: EFX_IDDIO Verilog HDL Instantiation

```
EFX_IDDIO # (
   .PULL_OPTION("NONE"),   // "NONE" (Default), "WEAK_PULLUP",
                           // "WEAK_PULLDOWN"
   .IS_CLK_INVERTED(0),    // 0 (Default), 1 to invert the clock signal
   .MODE("DDIO")           // "DDIO" (Default), "DDIO_RESYNC"
) EFX_IDDIO_inst (
   .O_HI(O_HI),    // Register output HI
   .O_LO(O_LO),    // Register output LO
   .I(I)           // Data input (Connect to top-level port)
   .CLK(CLK)       // Clock input
);
```

Figure 48: EFX_IDDIO VHDL Instantiation

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library efxphysicallib;
use efxphysicallib.efxcomponents.all

entity EFX_IDDIO_VHDL is
port ( o_hi : out std_logic;
       o_lo : out std_logic;
       i : in std_logic;
       clk : in std_logic
);
end entity EFX_IDDIO_VHDL;

architecture Behavioral of EFX_IDDIO_VHDL is
 begin
 EFX_IDDIO_inst : EFX_IDDIO
   generic map (
      PULL_OPTION => "NONE",
      IS_CLK_INVERTED => 0,
      MODE => "DDIO"
   )
   port map (
      O_HI => o_hi,
      O_LO => o_lo,
      I => i,
      CLK => clk
   );
 end architecture Behavioral;
```

EFX_ODDIO

Output Double Data I/O Register

This primitive is for the double data I/O (DDIO) register to the GPIO output. The DDIO register captures data on both positive and negative clock edges. The core receives 2 bit wide data from the interface.

EFX_ODDIO Ports

Figure 49: EFX_ODDIO Symbol

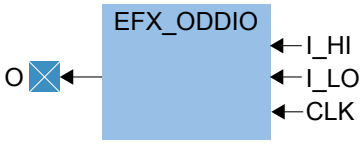


Table 36: EFX_ODDIO Ports

Port	Direction	Description
I_HI	Input	Output connection from the core. Captures pad data on the rising clock edge and outputs it on the same edge.
I_LO	Input	Output connection from the core. Captures pad data on the falling clock edge and outputs it on the falling edge.
CLK	Input	Clock signal from the core.
O	Output	GPIO pad in output mode.

EFX_ODDIO Parameters

Table 37: EFX_ODDIO Parameters

Parameter	Allowed Values	Description
IS_CLK_INVERTED	0, 1	Specify whether the clock is inverted. 0: Not inverted, data capture on rising edge (default). 1: Inverted, data capture on falling edge.
MODE	DDIO DDIO_RESYNC	The data synchronization mode. Refer to "Double Data I/O" in the data sheet for timing diagrams. DDIO: Normal mode. (Default) DDIO_RESYNC: Resync mode.

EFX_ODDIO Function

These examples show how to instantiate the output DDIO register.

Figure 50: EFX_ODDIO Verilog HDL Instantiation

```
EFX_ODDIO # (
   .IS_CLK_INVERTED(0),  // 0 (Default), 1 to invert the clock signal
   .MODE("DDIO")         // "DDIO" (Default) , "DDIO_RESYNC"
) EFX_ODDIO_inst (
   .I_HI(I_HI),   // Data input HI
   .I_LO(I_LO),   // Data input LO
   .O(O)          // Register output (connect to top-level port)
   .CLK(CLK)      // Clock input
);
```

Figure 51: EFX_ODDIO VHDL Instantiation

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library efxphysicallib;
use efxphysicallib.efxcomponents.all

entity EFX_ODDIO_VHDL is
port ( i_hi : in std_logic;
       i_lo : in std_logic;
       o : out std_logic;
       clk : in std_logic
);
end entity EFX_ODDIO_VHDL;

architecture Behavioral of EFX_ODDIO_VHDL is
 begin
 EFX_ODDIO_inst : EFX_ODDIO
   generic map (
      IS_CLK_INVERTED => 0,
      MODE => "DDIO"
   )
   port map (
      I_HI => i_hi,
      I_LO => i_lo,
      O => o,
      CLK => clk
   );
 end architecture Behavioral;
```

EFX_JTAG_CTRL

JTAG Interface

This primitve is for the 4-pin JTAG interface.

EFX_JTAG_CTRL Ports

Figure 52: EFX_JTAG_CTRL Symbol

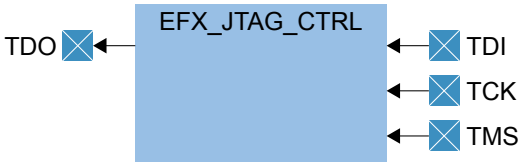


Table 38: EFX_JTAG_CTRL Ports

Port	Direction	Description
TDO	Output	Test data out.
TCK	Input	Test clock.
TDI	Input	Test data in.
TMS	Input	Test mode select.

EFX_JTAG_CTRL Parameters

Table 39: EFX_JTAG_CTRL Parameters

Parameter	Allowed Values	Description
DEVICE	T4F49, T4F81, T8-WB, T8F49, T8F81, T8Q144, T13F169, T13F256, T20-WB, T20W80, T20F169, T20F256, T20F324, T20F400, T35F324, T35F400, T55F324, T55F484, T55F576, T85F324, T85F484, T85F576, T120F324, T120F484, T120F576	The FPGA and package.

EFX_JTAG_CTRL Function

Use these examples to instantiate the JTAG pins.

Figure 53: EFX_JTAG_CTRL Verilog HDL Instantiation

```
EFX_JTAG_CTRL EFX_JTAG_CTRL_inst(
   .TDO(TDO),
   .TCK(TCK),
   .TDI(TDI),
   .TMS(TMS)
);
```

Figure 54: EFX_JTAG_CTRL VHDL Instantiation

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library efxphysicallib;
use efxphysicallib.efxcomponents.all

entity EFX_JTAG_CTRL_VHDL is
port ( tdo : out std_logic;
       tck : in std_logic;
       tdi : in std_logic;
       tms : in std_logic
);
end entity EFX_JTAG_CTRL_VHDL;

architecture Behavioral of EFX_JTAG_CTRL_VHDL is
 begin
 EFX_JTAG_CTRL_inst : EFX_JTAG_CTRL
   port map (
      TDO => tdo,
      TCK => tck,
      TDI => tdi,
      TMS => tms
   );
 end architecture Behavioral;
```

EFX_JTAG_V1

JTAG User TAP Interface

This primitve is for the JTAG user TAP interface.

EFX_JTAG_V1 Ports

Figure 55: EFX_JTAG_V1 Symbol

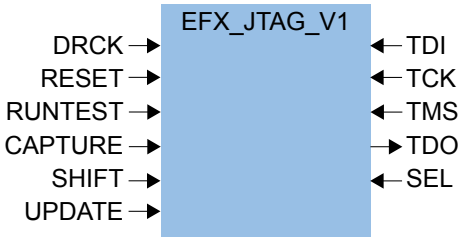


Table 40: EFX_JTAG_V1

Signal	Direction	Description
TDI	Input	JTAG test data in pin.
TCK	Input	JTAG test clock pin.
TMS	Input	JTAG mode select pin.
SEL	Input	User instructive active pin.
DRCK	Input	Gated test clock.
RESET	Input	Reset.
RUNTEST	Input	Run test pin.
CAPTURE	Input	Capture pin.
SHIFT	Input	Shift pin.
UPDATE	Input	Update pin.
TDO	Output	JTAG test data out pin.

EFX_JTAG_V1 Parameters

Table 41: EFX_JTAG_V1 Parameters

Parameter	Allowed Values	Description
RESOURCE	JTAG_USER1, JTAG_USER2, JTAG_USER3, JTAG_USER4	The JTAG user TAP interface name. T4 and T8 FPGAs in F49 and F81 packages only support JTAG_USER1 and JTAG_USER2.

EFX_JTAG_V1 Function

Use these examples to instantiate the JTAG pins.

Figure 56: EFX_JTAG_V1 Verilog HDL Instantiation

```
EFX_JTAG_V1 # (
   .RESOURCE("JTAG_USER1")
) EFX_JTAG_V1_inst(
   .CAPTURE(CAPTURE),
   .DRCK(DRCK),
   .RESET(RESET),
   .RUNTEST(RUNTEST),
   .SEL(SEL),
   .SHIFT(SHIFT),
   .TCK(TCK),
   .TDI(TDI),
   .TMS(TMS),
   .UPDATE(UPDATE),
   .TDO(TDO)
);
```

Figure 57: EFX_JTAG_V1 VHDL Instantiation

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library efxphysicallib;
use efxphysicallib.efxcomponents.all

entity EFX_JTAG_V1_VHDL is
port ( capture : out std_logic;
       drck : out std_logic;
       reset : out std_logic;
       runtest : out std_logic;
       sel : out std_logic;
       shift : out std_logic;
       tck : out std_logic;
       tdi : out std_logic;
       tms : out std_logic;
       update : out std_logic;
       tdo : in std_logic
);
end entity EFX_JTAG_V1_VHDL;

architecture Behavioral of EFX_JTAG_V1_VHDL is
 begin
 EFX_JTAG_V1_inst : EFX_JTAG_V1
   generic map (
      RESOURCE => "JTAG_USER1"
   )
   port map (
      CAPTURE => capture,
      DRCK => drck,
      RESET => reset,
      RUNTEST => runtest,
      SEL => sel,
      SHIFT => shift,
      TCK => tck,
      TDI => tdi,
      TMS => tms,
      UPDATE => update,
      TDO => tdo
   );
 end architecture Behavioral;
```

EFX_GPIO_V1

Basic GPIO

This simple version of the GPIO is only available in T4 and T8 FPGAs in F49 and F81 packages.

Learn more: Refer to the data sheet for the complete description of the GPIO functionality and features. This user guide only describes the primitive ports and parameters.

EFX_GPIO_V1 Ports

Figure 58: EFX_GPIO_V1 Symbol

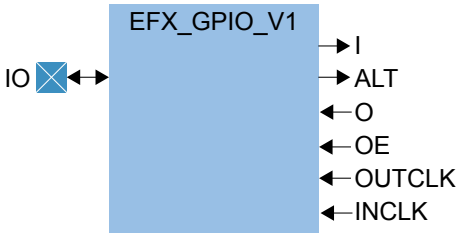


Table 42: EFX_GPIO_V1 Ports

Signal	Direction	Description
I	Output	Input data from the GPIO pad to the core fabric.
ALT	Output	Alternative input connection (in the Interface Designer, the input **Register Option** is **none**). Alternative connections are GCLK, GCTRL, and PLL_CLKIN.
O	Input	Output data to GPIO pad from the core fabric.
OE	Input	Output enable from core fabric to the I/O block. Can be registered.
OUTCLK	Input	Core clock that controls the output and OE register. This clock is not visible in the user netlist.
INCLK	Input	Core clock that controls the input register. This clock is not visible in the user netlist.

Table 43: EFX_GPIO_V1 Pads

Signal	Direction	Description
IO	Bidirectional	GPIO pad.

EFX_GPIO_V1 Parameters

Table 44: EFX_GPIO_V1 Parameters

Parameter	Allowed Values	Description
MODE	INPUT, OUTPUT, INOUT, CLKOUT	The GPIO direction or clock mode.
OUT_REG	BYPASS, REG, INVREG	Bypass (default), register, or inverted register the output.
IN_REG	BYPASS, REG	Bypass (default) or register the input.
OE_REG	BYPASS, REG	Bypass (default) or register the output enable.
PULL_OPTION	NONE, WEAK_PULLUP, WEAK_PULLDOWN	The pull direction, if any. NONE: Default
IS_OUTCLK_INVERTED	0, 1	Specifies whether the output clock is inverted. 0: Not inverted (default) 1: Inverted
IS_INCLK_INVERTED	0, 1	Specifies whether the intput clock is inverted. 0: Not inverted (default) 1: Inverted
CONNECTION_TYPE	NORMAL, GCLK, GCTRL, PLL_CLKIN, PLL_EXTFB, MIPI_CLKIN	The alternate connection type. Used with the ALT port. NORMAL: Default
SCHMITT_TRIGGER	0, 1	Specifies whether to use the Schmitt trigger. 0: Do not use (default) 1: Use
DRIVE_STRENGTH	1, 2, 3, 4	Indicates drive strength level. 1: Default
SLEW_RATE	0, 1	Specifies whether to turn on slew rate. 0: Turn off (default) 1: Turn on
IO_STANDARD	1.8_V_LVCMOS, 2.5_V_LVCMOS, 3.3_V_LVTTL_LVCMOS	Indicates the I/O standard for the GPIO. 3.3_V_LVTTL_LVCMOS: Default

EFX_GPIO_V1 Function

These examples show how to instantiate the Trion GPIO V1 primitive.

Figure 59: EFX_GPIO_V1 Verilog HDL Instantiation

```
EFX_GPIO_V1 # (
   .MODE("INPUT"),
   .OUT_REG("BYPASS"),
   .IN_REG("BYPASS"),
   .OE_REG("BYPASS"),
   .PULL_OPTION("NONE"),
   .IS_OUTCLK_INVERTED(0),
   .IS_INCLK_INVERTED(0),
   .CONNECTION_TYPE("NONE"),
   .DRIVE_STRENGTH(1),
   .SLEW_RATE(0),
   .IO_STANDARD("3.3_V_LVTTL_LVCMOS")
) EFX_GPIO_V1_inst (
   .I(I),                // 1 bit output: Input data from GPIO to the core
 fabric
   .ALT(ALT),            // Alternative connection
   .O(O),                // 1 bit input: Output data to GPIO from the core
 fabric
   .OE(OE),              // Output enable
   .OUTCLK(OUTCLK),      // Core clock that control O / OE
   .INCLK(INCLK),        // Core clock that control I
   .IO(IO)               // Connect to top-level port
);
```

Figure 60: EFX_GPIO_V1 VHDL Instantiation

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library efxphysicallib;
use efxphysicallib.efxcomponents.all

entity EFX_GPIO_V1_VHDL is
port ( i : in std_logic;
       alt : out std_logic;
       o : out std_logic;
       outclk : in std_logic;
       inclk : in std_logic;
       io : inout std_logic
);
end entity EFX_GPIO_V1_VHDL;

architecture Behavioral of EFX_GPIO_V1_VHDL is
 begin
 EFX_GPIO_V1_inst : EFX_GPIO_V1
   generic map (
      MODE => "INPUT",
      OUT_REG => "BYPASS",
      IN_REG => "BYPASS",
      OE_REG => "BYPASS",
      PULL_OPTION => "NONE",
      IS_OUTCLK_INVERTED => 0,
      IS_INCLK_INVERTED => 1,
      CONNECTION_TYPE => "NONE",
      SCHMITT_TRIGGER => 0,
      DRIVE_STRENGTH => 1,
      SLEW_RATE => 0,
      IO_STANDARD => "3.3_V_LVTTL_LVCMOS"
   )
   port map (
      I => o,
      ALT => alt,
      O => i,
      OE => oe,
      OUTCLK => outclk,
      INCLK => inclk,
      IO => io
   );
 end architecture Behavioral;
```

EFX_GPIO_V2

GPIO with Double Data I/O Function

This version of the GPIO includes a double data I/O (DDIO) option. It is available for all Trion FPGAs except for T4 and T8 FPGAs in F49 and F81 packages.

Learn more: Refer to the data sheet for the complete description of the GPIO functionality and features. This user guide only describes the primitive ports and parameters.

EFX_GPIO_V2 Ports

Figure 61: EFX_GPIO_V2 Symbol

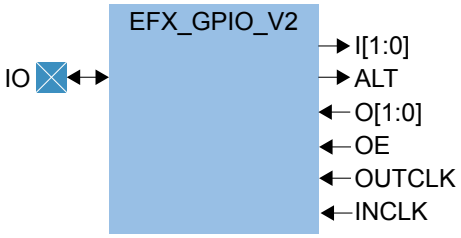


Table 45: EFX_GPIO_V2 Ports

Signal	Direction	Description
I[1:0]	Output	Input data from the GPIO pad to the core fabric. I[0] is the normal input to the core. In DDIO mode, I[0] is the data captured on the positive clock edge (HI pin name in the Interface Designer) and I[1] is the data captured on the negative clock edge (LO pin name in the Interface Designer).
ALT	Output	Alternative input connection (in the Interface Designer, **Register Option** is **none**). Alternative connections are GCLK, GCTRL, PLL_CLKIN, and MIPI_CLKIN.[4]
O[1:0]	Input	Output data to GPIO pad from the core fabric. O[0] is the normal output from the core. In DDIO mode, O[0] is the data output on the positive clock edge (HI pin name in the Interface Designer) and O[1] is the data output on the negative clock edge (LO pin name in the Interface Designer).
OE	Input	Output enable from core fabric to the I/O block. Can be registered.
OUTCLK	Input	Core clock that controls the output and OE registers. This clock is not visible in the user netlist unless you instantiate an EFX_GPIO_V2 primitive.
INCLK	Input	Core clock that controls the input registers. This clock is not visible in the user netlist unless you instantiate an EFX_GPIO_V2 primitive.

Table 46: EFX_GPIO_V2 Pads

Signal	Direction	Description
IO	Bidirectional	GPIO pad.

[4] MIPI_CLKIN is only available in packages that support MIPI.

EFX_GPIO_V2 Parameters

Table 47: EFX_GPIO_V2 Parameters

Parameter	Allowed Values	Description
MODE	NONE, INPUT, OUTPUT, INOUT, CLKOUT	The GPIO direction or clock mode.
OUT_REG	BYPASS, REG, INVREG, DDIO, DDIO_RESYNC	Bypass (default), register, inverted register, or use DDIO for the output.
IN_REG	BYPASS, REG, DDIO, DDIO_RESYNC	Bypass (default), register, or use DDIO for the input.
OE_REG	BYPASS, REG	Bypass (default) or register.
PULL_OPTION	NONE, WEAK_PULLUP, WEAK_PULLDOWN	The pull direction, if any. NONE: Default
IS_OUTCLK_INVERTED	0, 1	Specifies whether the output clock is inverted. 0: Not inverted (default) 1: Inverted
IS_INCLK_INVERTED	0, 1	Specifies whether the intput clock is inverted. 0: Not inverted (default) 1: Inverted
CONNECTION_TYPE	NORMAL, GCLK, GCTRL, PLL_CLKIN, PLL_EXTFB, MIPI_CLKIN	The alternate connection type. Used with the ALT port. NORMAL: Default
SCHMITT_TRIGGER	0, 1	Specifies whether to use the Schmitt trigger. 0: Do not use (default) 1: Use
DRIVE_STRENGTH	1, 2, 3, 4	Indicates drive strength level. 1: Default
SLEW_RATE	0, 1	Specifies whether to turn on slew rate. 0: Turn off (default) 1: Turn on
IO_STANDARD	1.8_V_LVCMOS, 2.5_V_LVCMOS, 3.3_V_LVTTL_LVCMOS	Indicates the I/O standard for the GPIO. 3.3_V_LVTTL_LVCMOS: Default

EFX_GPIO_V2 Function

These examples show how to instantiate the Trion GPIO V2 primitive.

Figure 62: EFX_GPIO_V2 Verilog HDL Instantiation

```
EFX_GPIO_V2 # (
   .MODE("INPUT"),
   .OUT_REG("BYPASS"),
   .IN_REG("BYPASS"),
   .OE_REG("BYPASS"),
   .PULL_OPTION("NONE"),
   .IS_OUTCLK_INVERTED(0),
   .IS_INCLK_INVERTED(0),
   .CONNECTION_TYPE("NONE"),
   .DRIVE_STRENGTH(1),
   .SLEW_RATE(0),
   .IO_STANDARD("3.3_V_LVTTL_LVCMOS")
) EFX_GPIO_V2_inst (
   .I(I),                // 2 bit output: Input data from GPIO to the core
 fabric
   .ALT(ALT),            // Alternative connection
   .O(O),                // 2 bit input: Output data to GPIO from the core
 fabric
   .OE(OE),              // Output enable
   .OUTCLK(OUTCLK),      // Core clock that control O / OE
   .INCLK(INCLK),        // Core clock that control I
   .IO(IO)               // Connect to top-level port
);
```

Figure 63: EFX_GPIO_V2 VHDL Instantiation

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library efxphysicallib;
use efxphysicallib.efxcomponents.all

entity EFX_GPIO_V2_VHDL is
port ( i : in std_logic_vector (1 downto 0);
       alt : out std_logic;
       o : out std_logic_vector (1 downto 0);
       outclk : in std_logic;
       inclk : in std_logic;
       io : inout std_logic
);
end entity EFX_GPIO_V2_VHDL;

architecture Behavioral of EFX_GPIO_V2_VHDL is
 begin
 EFX_GPIO_V2_inst : EFX_GPIO_V2
   generic map (
      MODE => "INPUT",
      OUT_REG => "BYPASS",
      IN_REG => "BYPASS",
      OE_REG => "BYPASS",
      PULL_OPTION => "NONE",
      IS_OUTCLK_INVERTED => 0,
      IS_INCLK_INVERTED => 1,
      CONNECTION_TYPE => "NONE",
      SCHMITT_TRIGGER => 0,
      DRIVE_STRENGTH => 1,
      SLEW_RATE => 0,
      IO_STANDARD => "3.3_V_LVTTL_LVCMOS"
   )
   port map (
      I => o,
      ALT => alt,
      O => i,
      OE => oe,
      OUTCLK => outclk,
      INCLK => inclk,
      IO => io
   );
 end architecture Behavioral;
```

EFX_LVDS_RX_V1

LVDS Receiver

This LVDS primitive is available in all Trion FPGAs except the T8 and T8 in F49 and F81 packages.

Learn more: Refer to the data sheet for the complete description of the LVDS functionality and features. This user guide only describes the primitive ports and parameters.

EFX_LVDS_RX_V1 Ports

Figure 64: EFX_LVDS_RX_V1 Symbol

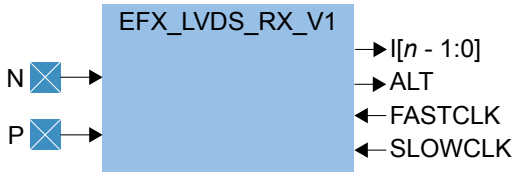


Table 48: EFX_LVDS_RX_V1 Ports

Signal	Direction	Notes
I[*n*-1:0]	Output	Parallel input data where *n* is the de-serialization factor. A width of 1 bypasses the deserializer.
ALT	Output	Alternative input, only available for an LVDS RX resource in bypass mode (deserialization width is 1; alternate connection type). Alternative connections are PLL_CLKIN.
FASTCLK	Input	Fast clock to de-serialize the data from the LVDS pads.
SLOWCLK	Input	Slow clock to latch the incoming data to the core.

Table 49: EFX_LVDS_RX_V1 Pads

Pad	Direction	Description
RXP	Input	Differential P pad.
RXN	Input	Differential N pad.

EFX_LVDS_RX_V1 Parameters

Table 50: EFX_LVDS_RX_V1 Parameters

Parameter	Allowed Values	Description
DESERIALIZATION_WIDTH	2, 3, 4, 5, 6, 7, 8	De-serialization factor.
DESERIALIZATION_EN	0, 1	When set to 0, the de-serializer is bypassed and the LVDS buffer is used as a normal input.
TERMINATION_TYPE	ON, OFF	Termination setting: ON: Enable on-die termination. OFF: Disabled.
CONNECTION_TYPE	NORMAL, PLL_CLKIN	Sets the alternate connection type. NORMAL: Normal LVDS RX function (same as choosing **normal** in the Interface Designer). PLL_CLKIN: Use the PLL clock input alternate function.
DELAY	0 - 63	Set the amount of static delay; each step adds approximately 25 ps of delay.

EFX_LVDS_RX_V1 Function

These examples show how to instantiate the Trion LVDS RX V1 primitive.

Figure 65: EFX_LVDS_RX_V1 Verilog HDL Instantiation

```
EFX_LVDS_RX_V1 # (
   .DESERIALIZATION_EN(0),
   .DESERIALIZATION_WIDTH(8),
   .TERMINATION_TYPE("OFF"),
   .CONNECTION_TYPE("NORMAL")
) EFX_LVDS_RX_V1_inst (
   .I(I),
   .ALT(ALT),
   .P(P),
   .N(N),
   .SLOWCLK(SLOWCLK),
   .FASTCLK(FASTCLK)
);
```

Figure 66: EFX_LVDS_RX_V1 VHDL Instantiation

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library efxphysicallib;
use efxphysicallib.efxcomponents.all

entity EFX_LVDS_RX_V1_VHDL is
port ( i : out std_logic_vector (7 downto 0);
       alt : out std_logic;
       p : in std_logic;
       n : in std_logic;
       slowclk : in std_logic;
       fastclk : in std_logic;
       rst : in std_logic
);
end entity EFX_LVDS_RX_V1_VHDL;

architecture Behavioral of EFX_LVDS_RX_V1_VHDL is
 begin
 EFX_LVDS_RX_V1_inst : EFX_LVDS_RX_V1
   generic map (
      DESERIALIZATION_EN => 0,
      DESERIALIZATION_WIDTH => 8,
      TERMINATION_TYPE => "OFF",
      CONNECTION_TYPE => "NORMAL"
   )
   port map (
      I => i,
      ALT => alt,
      P => p,
      N => n,
      SLOWCLK => slowclk,
      FASTCLK => fastclk
   );
 end architecture Behavioral;
```

EFX_LVDS_TX_V1

LVDS Transmit

This LVDS primitive is available in all Trion FPGAs except the T8 and T8 in F49 and F81 packages.

Learn more: Refer to the data sheet for the complete description of the LVDS functionality and features. This user guide only describes the primitive ports and parameters.

EFX_LVDS_TX_V1 Ports

Figure 67: EFX_LVDS_TX_V1 Symbol

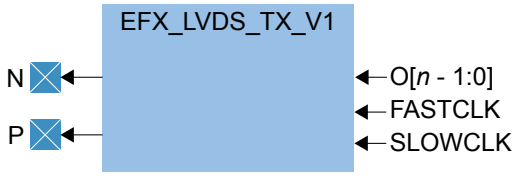


Table 51: EFX_LVDS_TX_V1 Ports

Signal	Direction	Notes
O[*n*-1:0]	Input	Parallel output data where *n* is the serialization factor. A width of 1 bypasses the serializer.
FASTCLK	Input	Fast clock to serialize the data to the LVDS pads.
SLOWCLK	Input	Slow clock to latch the incoming data from the core.

Table 52: EFX_LVDS_TX_V1 Pads

Pad	Direction	Description
TXP	Output	Differential P pad.
TXN	Output	Differential N pad.

EFX_LVDS_TX_V1 Parameters

Table 53: EFX_LVDS_TX_V1 Parameters

Parameter	Allowed Values	Description
SERIALIZATION_WIDTH	2, 3, 4, 5, 6, 7, 8	Serialization factor.
SERIALIZATION_EN	0, 1	When set to 0, the serializer is bypassed.
MODE	DATA, CLKOUT	Sets the mode: DATA: Simple output buffer or serialized output. CLKOUT: Use the transmitter as a clock output.
DIV2_CLKOUT	0, 1	Divider setting: 0: No clock divider. 1: Divide clock by 2.
OUTPUT_LOAD	3, 5, 7, 10	Sets the output load in pF. Use an output load of 7 pF or higher to achieve the maximum supported toggle rate. The Q144 package does not support output load 3.
REDUCED_SWING	0, 1	Sets the reduced VOD swing feature (similar to slow slew rate): 0: Disabled. 1: Enabled.

EFX_LVDS_TX_V1 Function

These examples show how to instantiate the Trion LVDS TX V1 primitive.

Figure 68: EFX_LVDS_TX_V1 Verilog HDL Instantiation

```
EFX_LVDS_TX_V1 # (
   .SERIALIZATION_EN(0),
   .SERIALIZATION_WIDTH(8),
   .MODE("DATA"),
   .DIV2_CLKOUT(1),
   .OUTPUT_LOAD(7),
   .REDUCED_SWING(0)
) EFX_LVDS_TX_V1_inst (
   .P(P),
   .N(N),
   .O(O),
   .SLOWCLK(SLOWCLK),
   .FASTCLK(FASTCLK),
   .OE(OE)
);
```

Figure 69: EFX_LVDS_TX_V1 VHDL Instantiation

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library efxphysicallib;
use efxphysicallib.efxcomponents.all

entity EFX_LVDS_TX_V1_VHDL is
port ( p : out std_logic;
       n : out std_logic;
       o : in std_logic_vector(7 downto 0);
       slowclk : in std_logic;
       fastclk : in std_logic;
       oe : in std_logic;
       rst: in std_logic
);
end entity EFX_LVDS_TX_V1_VHDL;

architecture Behavioral of EFX_LVDS_TX_V1_VHDL is
 begin
 EFX_LVDS_TX_V1_inst : EFX_LVDS_TX_V1
   generic map (
      SERIALIZATION_EN => 0,
      SERIALIZATION_WIDTH => 8,
      MODE => "DATA",
      DIV2_CLKOUT => 1,
      OUTPUT_LOAD => 7,
      REDUCED_SWING => 0
   )
   port map (
      P => p,
      N => n,
      O => o,
      SLOWCLK => slowclk,
      FASTCLK => fastclk,
      OE => oe
   );
 end architecture Behavioral;
```

EFX_PLL_V1

Simple PLL

This simple version of the PLL is only available in T4 and T8 FPGAs in F49 and F81 packages.

Learn more: Refer to the data sheet for the complete description of the PLL functionality and features. This user guide only describes the primitive ports and parameters.

EFX_PLL_V1 Ports

Figure 70: EFX_PLL_V1 Symbol

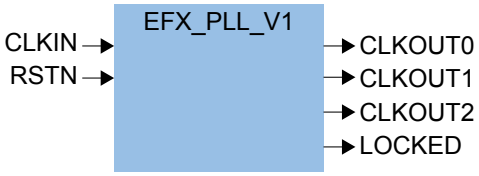


Table 54: EFX_PLL_V1 Ports

Port	Direction	Description
CLKIN	Input	Reference clock. This port is also a GPIO pin; the GPIO pins' alternate function is configured as a reference clock.
RSTN	Input	Active-low PLL reset signal. When asserted, this signal resets the PLL; when de-asserted, it enables the PLL. Connect this signal in your design to power up or reset the PLL. Assert the RSTN pin for a minimum pulse of 10 ns to reset the PLL.
CLKOUT0 CLKOUT1 CLKOUT2	Output	PLL output. The designer can route these signals as input clocks to the core's GCLK network.
LOCKED[5]	Output	Goes high when PLL achieves lock; goes low when a loss of lock is detected. Connect this signal in your design to monitor the lock status. This signal is analog asynchronous.

[5] The circuitry that generates the lock signal relies on a reference clock edge to transition the lock signal. A sudden removal of the reference clock will result in there being no positive clock edge with which to change the lock state from 1 back to 0. Therefore, the lock signal will remain on 1.

EFX_PLL_V1 Parameters

Table 55: EFX_PLL_V1 Parameters

Parameter	Allowed Values	Description
M	1 - 255	Pre-divider.
N	1 - 15	Multiplier.
O	1, 2, 4, 8	Post-divider.
REFCLK_FREQ	10 - 50	Reference clock frequency.
CLKOUT0_DIV	2, 4, 8, 16, 32, 64, 128, 256	Output clock divider for CLKOUT0.
CLKOUT1_DIV	2, 4, 8, 16, 32, 64, 128, 256	Output clock divider for CLKOUT1.
CLKOUT2_DIV	2, 4, 8, 16, 32, 64, 128, 256	Output clock divider for CLKOUT2.

EFX_PLL_V1 Function

These examples show how to instantiate the Trion simple PLL V1 primitive.

Figure 71: EFX_PLL_V1 Verilog HDL Instantiation

```
EFX_PLL_V1 # (
   .M(1),
   .N(1),
   .O(1),
   .CLKOUT0_DIV(2),
   .CLKOUT1_DIV(2),
   .CLKOUT2_DIV(2),
   .REFCLK_FREQ(25.0)
) EFX_PLL_V1_inst (
   .CLKOUT0(CLKOUT0),
   .CLKOUT1(CLKOUT1),
   .CLKOUT2(CLKOUT2),
   .LOCKED (LOCKED),
   .RSTN(RSTN),
   .CLKIN(CLKIN)          // Reference Clock Input, connect to either
                          // top-level port or EFX_IBUF's O port
);
```

Figure 72: EFX_PLL_V1 VHDL Instantiation

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library efxphysicallib;
use efxphysicallib.efxcomponents.all

entity EFX_PLL_V1_VHDL is
port ( clkout0 : out std_logic;
       clkout1 : out std_logic;
       clkout2: out std_logic;
       locked : out std_logic;
       rstn : in std_logic;
       clkin : in std_logic
);

end entity EFX_PLL_V1_VHDL;

architecture Behavioral of EFX_PLL_V1_VHDL is
 begin
 EFX_PLL_V1_inst : EFX_PLL_V1
   generic map (
      M => 1,
      N => 1,
      O => 1,
      CLKOUT0_DIV => 2,
      CLKOUT1_DIV=> 2,
      CLKOUT2_DIV => 2,
      REFCLK_FREQ => 25.0
   )
   port map (
      CLKOUT0 => clkout0,
      CLKOUT1 => clkout1,
      CLKOUT2 => clkout2,
      LOCKED => locked,
      RSTN => rstn,
      CLKIN => clkin
   );
 end architecture Behavioral;
```

EFX_PLL_V2

Advanced PLL

This complex version of the PLL is available in all Trion FPGA except T4 and T8 FPGAs in F49 and F81 packages.

Learn more: Refer to the data sheet for the complete description of the PLL functionality and features. This user guide only describes the primitive ports and parameters.

EFX_PLL_V2 Ports

Figure 73: EFX_PLL_V2 Symbol

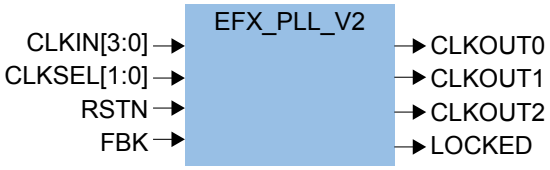


Table 56: EFX_PLL_V2 Ports

Signal	Direction	Description
CLKIN[3:0]	Input	Reference clocks driven by I/O pads or core clock tree.
CLKSEL[1:0]	Input	You can dynamically select the reference clock from one of the clock in pins.
RSTN	Input	Active-low PLL reset signal. When asserted, this signal resets the PLL; when de-asserted, it enables the PLL. De-assert only when the CLKIN signal is stable. Connect this signal in your design to power up or reset the PLL. Assert the RSTN pin for a minimum pulse of 10 ns to reset the PLL. Assert RSTN when dynamically changing the selected PLL reference clock.
FBK	Input	Connect to a clock out interface pin when the PLL feedback mode is not **internal**.
CLKOUT0 CLKOUT1 CLKOUT2	Output	PLL output. The designer can route these signals as input clocks to the core's GCLK network.
LOCKED[6]	Output	Goes high when PLL achieves lock; goes low when a loss of lock is detected; remains at previous state if the CLKIN goes discontinuous. Connect this signal in your design to monitor the lock status. This signal is not synchronized to any clock and the minimum high or low pulse width of the lock signal may be smaller than the CLKOUT's period.

[6] The circuitry that generates the lock signal relies on a reference clock edge to transition the lock signal. A sudden removal of the reference clock will result in there being no positive clock edge with which to change the lock state from 1 back to 0. Therefore, the lock signal will remain on 1.

EFX_PLL_V2 Parameters

Table 57: EFX_PLL_V2 Parameters

Parameter	Allowed Values	Description
N	1 - 15	Pre-divider.
M	1 - 255	Multiplier.
O	1, 2, 4, 8	Post divider.
CLKOUT0_DIV	1 - 128	Output divider for CLKOUT0.
CLKOUT1_DIV	1 - 128	Output divider for CLKOUT1.
CLKOUT2_DIV	1 - 128	Output divider for CLKOUT2.
CLKOUT0_PHASE	0, 45, 90, 135, 180, 270	Output phase for CLKOUT0.
CLKOUT1_PHASE	0, 45, 90, 135, 180, 270	Output phase for CLKOUT1.
CLKOUT2_PHASE	0, 45, 90, 135, 180, 270	Output phase for CLKOUT2.
FEEDBACK_CLK	INTERNAL, CLK0, CLK1, CLK2	Indicates which clock is the feedback clock.
FEEDBACK_MODE	INTERNAL, LOCAL, CORE	Indicates the feedback mode.
REFCLK_FREQ	10 - 330	Reference clock frequency.

Follow these guidelines for the feedback parameters.

Table 58: Parameter Guidelines

Mode	Settings
Single reference clock	Connect reference clock signal to CLKIN[0]. Set reference clock select signal to `2'b00`.
Internal feedback mode	Set FEEDBACK_CLK parameter to `INTERNAL`. Set FEEDBACK_MODE parameter to `INTERNAL`. Set feedback port to `1'b0`.
Local feedback mode	Set FEEDBACK_CLK parameter to `CLK0`. Set FEEDBACK_MODE parameter to `LOCAL`. Connect the feedback port directly to the CLKOUT0 port of the same PLL primitive.
Core feedback mode	Set FEEDBACK_CLK parameter to `CLK0`, `CLK1`, or `CLK2`. Set FEEDBACK_MODE parameter to `CORE`. Connect the feedback port directly to the CLKOUT0, CLKOUT1, or CLKOUT2 port of the same primitive (that is, connect to the output specified with FEEDBACK_CLK).

EFX_PLL_V2 Function

These examples show how to instantiate the PLL V2 primitive.

Figure 74: EFX_PLL_V2 Verilog HDL Instantiation

```
EFX_PLL_V2 # (
   .M(1),
   .N(1),
   .O(1),
   .CLKOUT0_DIV(1),
   .CLKOUT1_DIV(1),
   .CLKOUT2_DIV(1),
   .CLKOUT0_PHASE(0),
   .CLKOUT1_PHASE(0),
   .CLKOUT2_PHASE(0),
   .FEEDBACK_CLK("INTERNAL"),
   .FEEDBACK_MODE ("INTERNAL"),
   .REFCLK_FREQ(25.0)
) EFX_PLL_V2_inst (
   .CLKOUT0(CLKOUT0),
   .CLKOUT1(CLKOUT1),
   .CLKOUT2(CLKOUT2),
   .LOCKED(LOCKED),
   .RSTN(RSTN),
   .FBK(FBK),              // Feedback Clock Input
   .CLKIN(CLKIN),          // Reference Clock Input
   .CLKSEL(CLKSEL)         // Reference Clock Input Sel
);
```

Figure 75: EFX_PLL_V2 VHDL Instantiation

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library efxphysicallib;
use efxphysicallib.efxcomponents.all

entity EFX_PLL_V2_VHDL is
port ( clkout0 : out std_logic;
       clkout1: out std_logic;
       clkout2: out std_logic;
       locked : out std_logic;
       fbk : in std_logic;
       rstn : in std_logic;
       clkin : in std_logic_vector (3 downto 0);
       clksel : in std_logic_vector (1 downto 0)
);

end entity EFX_PLL_V2_VHDL;

architecture Behavioral of EFX_PLL_V2_VHDL is
 begin
 EFX_PLL_V2_inst : EFX_PLL_V2
   generic map (
      M => 1,
      N => 1,
      O => 1,
      CLKOUT0_DIV => 1,
      CLKOUT1_DIV => 1,
      CLKOUT2_DIV => 1,
      CLKOUT0_PHASE => 0,
      CLKOUT1_PHASE=> 0,
      CLKOUT2_PHASE => 0,
      FEEDBACK_CLK => "INTERNAL",
      FEEDBACK_MODE => "INTERNAL"
      REFCLK_FREQ => 25.0
   )
   port map (
      CLKOUT0 => clkout0,
      CLKOUT1 => clkout1,
      CLKOUT2 => clkout2,
      LOCKED => locked,
      RSTN => rstn,
      FBK => fbk,
      CLKIN => clkin,
      CLKSEL => clksel
   );
```

```
end architecture Behavioral;
```

EFX_OSC_V1

Oscillator

This oscillator is only available in T4 and T8 FPGAs in F49 and F81 packages.

Learn more: Refer to the data sheet for the complete description of the oscillator functionality and features. This user guide only describes the primitive ports and parameters.

EFX_OSC_V1 Ports

Figure 76: EFX_OSC_V1 Symbol

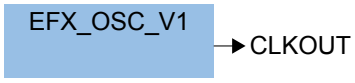


Table 59: EFX_OSC_V1 Ports

Port	Direction	Description
CLKOUT	Output	10 KHz oscillator clock output.

EFX_OSC_V1 Parameters

None.

EFX_OSC_V1 Function

These examples show how to instantiate the OSC V1 primitive.

Figure 77: EFX_OSC_V1 Verilog HDL Instantiation

```
EFX_OSC_V1 EFX_OSC_V1_inst (
   .CLKOUT(CLKOUT)
);
```

Figure 78: EFX_OSC_V1 VHDL Instantiation

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library efxphysicallib;
use efxphysicallib.efxcomponents.all

entity EFX_OSC_V1_VHDL is
port ( clkout : out std_logic);
end entity EFX_OSC_V1_VHDL;

architecture Behavioral of EFX_OSC_V1_VHDL is
 begin
 EFX_OSC_V1_inst : EFX_OSC_V1
   port map (
      CLKOUT => clkout
   );
 end architecture Behavioral;
```

Revision History

Table 60: Revision History

Date	Version	Description
September	6.2	Removed DDIO_PIPE from Table 35: EFX_IDDIO Parameters on page 44; Trion FPGAs do not support his mode. (DOC-2706)
June 2025	6.1	Corrected EFX_ADD parameter; IO_POLARITY should be I0_POLARITY (zero not letter O). (DOC-2580)
May 2025	6.0	Added LVDS interface primitives. (DOC-2483) Corrected syntax errors for interface primitive instantiations. (DOC-2194)
November 2024	5.0	Added interface primitives. (DOC-2086)
November 2023	4.6	Improved Allowed Read and Write Mode Combinations tables for EFX_RAM_5K and EFX_DRRAM_5K. (DOC-1566)
June 2023	4.5	Updated table EFX_FF Parameters and EFX_MULT Parameters. (DOC-1237)
March 2022	4.4	Updated description for the optional output register in EFX_RAM_5K and EFX_DPRAM_5K.
June 2021	4.3	Renamed as Quantum Trion Primitives User Guide.
June 2020	4.2	Added primitive instantiation examples in VHDL. Removed EFX_RAM_10K block description. Updated document formatting.
October 2018	4.1	Added the EFX_RAM_10K primitive. Removed the EFX_LATCH primitive.
April 2018	4.0	EFX_RAM_5K–Added description of the READ_UNKNOWN option for the WRITE_MODE parameter. EFX_MULT–Changed the following signals: • SRA to RSTA • SRB to RSTB • SBO to RSTO
November 2017	3.1	Added EFX_DPRAM_5K primitive. Updated EFX_RAM_5K description. Removed EFX_GBUF description.
May 2017	3.0	Removed OPM family information. Changed OPH family name to Quantum. Added GBUF primitive.
May 2016	2.0	Added EFX_RAM_5K and EFX_MULT primitive descriptions.
April 2015	1.0	Initial release.