Quantum® Titanium Primitives User Guide

UG-TIPRIM-v2.0
November 2024
www.efinixinc.com

Contents

Introduction

This document defines the Efinity® software technology-mapped logic primitives for Titanium FPGAs. These primitives 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
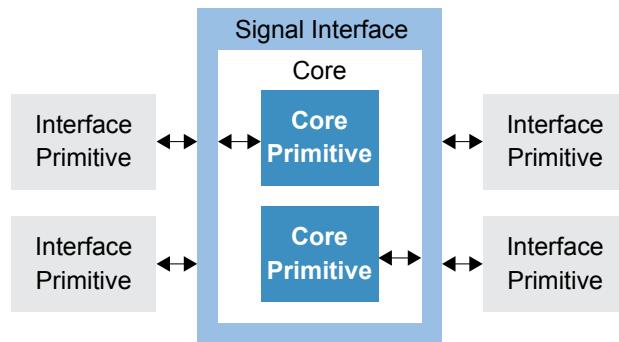
XLR Cell

The XLR cell functions as:
- A 4-input LUT that supports any combinational logic function with four inputs.
- A simple full adder.
- An 8-bit shift register that can be cascaded.
- A fracturable LUT or full adder.

The logic cell includes an optional flipflop. You can configure multiple logic cells to implement arithmetic functions such as adders, subtractors, and counters.

Figure 2: Logic Cell Functions
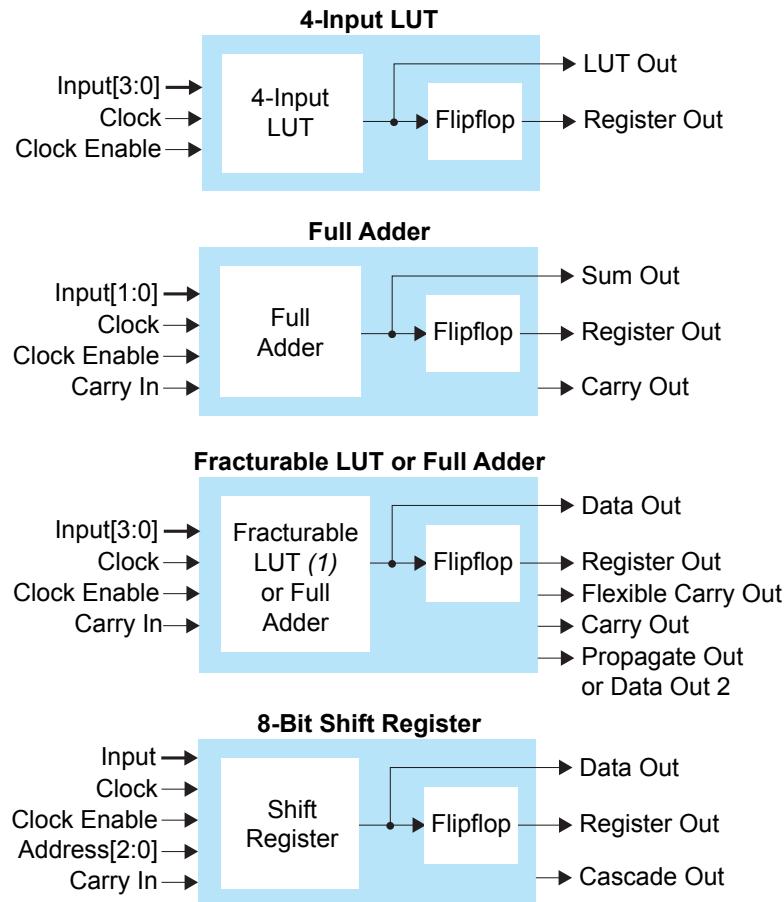
1. The fracturable LUT is a combination of a 3-input LUT and a 2-input LUT. They share 2 of the same inputs.

XLR cell primitives:
- **EFX_LUT4** on page 7
- **EFX_ADD** on page 10
- **EFX_COMB4** on page 13
- **EFX_FF** on page 18
- **EFX_SRL8** on page 20

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
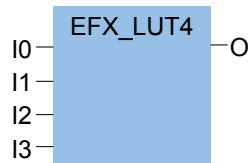
Table 1: EFX_LUT4 Ports

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]
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
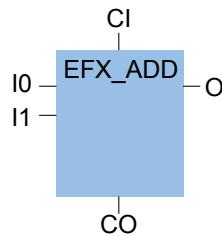
Table 4: EFX_ADD Ports

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
IO_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 # (
    .IO_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

```vhdl
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_COMB4

Simple 4-Input LUT ROM plus Simple Adder

The EFX_COMB4 primitive supports the full capabilities of the Titanium logic cell. The EFX_COMB4 has all the inputs and outputs of the EFX_LUT4 and the EFX_ADD plus two additional outputs, P and FCO. The P output is the internal 4-LUT signal that generates the propagate logic used to calculate carry-out (CO). The FCO is a flexible carry-out signal that can drive normal logic (CO can only drive the CI port of another EFX_COMB4).

EFX_COMB4 Ports

Figure 9: EFX_COMB4 Symbol
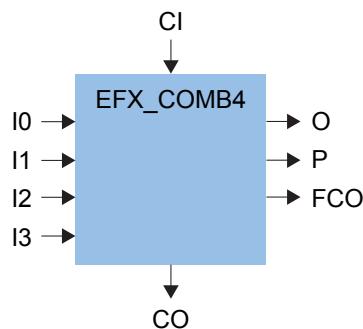
Table 7: EFX_COMB4 Ports

Port	Direction	Description
I0	Input	Data in 0.
I1	Input	Data in 1.
I2	Input	Data in 2.
I3	Input	Data in 3.
CI	Input	Carry in. This port is only fed by dedicated routing from the adjacent logic cell's CO connection. Therefore, tie the first CI in an adder chain to ground.[1]
O	Output	Data out.
P	Output	Propagate out.
CO	Output	Carry out. Dedicated fast connection to the CI of the neighboring EFX_COMB4 primitive. It passes the inverted value of the carry function.
FCO	Output	Flexible carry out. General-purpose output from the EFX_COMB4 that can connect to any block. It passes the un-inverted value of the carry function.

---

[1] In the place and route netlist, the CI input is unconnected and treated as GND in the LUTMASK.

EFX_COMB4 Parameters

Table 8: EFX_COMB4 Parameters

Parameter	Allowed Values	Description
LUTMASK	Any 16-bit hexadecimal number	Content of LUT ROM.
MODE	LOGIC, ARITH	Indicates if implementing a general logic function or an arithmetic function.

EFX_COMB4 Function

You can use EFX_COMB4 to map complicated functions that are not supported by EFX_LUT4 or EFX_ADD. For example, you can export the arithmetic propagate signal to implement a carry skip adder in soft logic or pack a 3-input and 2-input function into one logic cell.

The MODE parameter controls whether the EFX_COMB4 is performing general logic (LOGIC) or arithmetic functions (ARITH).

Table 9: EFX_COMB Modes

Signal	LOGIC Mode	ARITH Mode
I0	Input.	Input.
I1	Input.	Input.
I2	Input.	Unused.
I3	Input.	Unused.
O	Is a function of the I3, I2, I1, and I0 inputs.	Is a function of the CI, I1, and I0 inputs.
P	Is a function of the I1 and I0 inputs.	Is a function of the I1 and I0 inputs.
FCO	Unused.	Is a function of the CI, I1, and I0 inputs.
CI	Unused.	Carry in.
CO	Unused.	Is a function of the CI, I1, and I0 inputs.
Notes	When just using the O output, all 4 inputs are rotatable during routing. When using the O and P outputs, only the I1 and I0 inputs may be rotated during routing. *Only a flipflop being driven by the O output may be packed with this cell.*	Unlike EFX_ADD, the function of the outputs is dependent on the LUTMASK parameter. The I1 and I0 inputs may be rotated during routing.

Leave unused inputs unconnected and as a "don't care" in the LUTMASK. The Efinity® software issues an error if the LUTMASK depends on an unconnected input. A LUT cannot be driven by the same net on different input ports. Remove any redundant inputs and adjust the LUTMASK appropriately.

Note: EFX_ADD and EFX_COMB4 may not be intermingled in a single adder chain because the CO signal of an EFX_COMB4 is inverting, while the EFX_ADD is not.

Table 10: EFX_COMB4 Function (Logic Mode)

Inputs				Outputs	
I3	I2	I1	I0	O	P
0	0	0	0	LUTMASK[0]	LUTMASK[8]
0	0	0	1	LUTMASK[1]	LUTMASK[9]
0	0	1	0	LUTMASK[2]	LUTMASK[10]
0	0	1	1	LUTMASK[3]	LUTMASK[11]
0	1	0	0	LUTMASK[4]	LUTMASK[8]
0	1	0	1	LUTMASK[5]	LUTMASK[9]
0	1	1	0	LUTMASK[6]	LUTMASK[10]
0	1	1	1	LUTMASK[7]	LUTMASK[11]
1	0	0	0	LUTMASK[8]	LUTMASK[8]
1	0	0	1	LUTMASK[9]	LUTMASK[9]
1	0	1	0	LUTMASK[10]	LUTMASK[10]
1	0	1	1	LUTMASK[11]	LUTMASK[11]
1	1	0	0	LUTMASK[12]	LUTMASK[8]
1	1	0	1	LUTMASK[13]	LUTMASK[9]
1	1	1	0	LUTMASK[14]	LUTMASK[10]
1	1	1	1	LUTMASK[15]	LUTMASK[11]

Table 11: EFX_COMB4 Function (Arithmetic Mode)

Inputs			Outputs		
CI	I1	I0	O	PROP	GEN
0	0	0	LUTMASK[0]	LUTMASK[8]	LUTMASK[12]
0	0	1	LUTMASK[1]	LUTMASK[9]	LUTMASK[13]
0	1	0	LUTMASK[2]	LUTMASK[10]	LUTMASK[14]
0	1	1	LUTMASK[3]	LUTMASK[11]	LUTMASK[15]
1	0	0	LUTMASK[4]	LUTMASK[8]	LUTMASK[12]
1	0	1	LUTMASK[5]	LUTMASK[9]	LUTMASK[13]
1	1	0	LUTMASK[6]	LUTMASK[10]	LUTMASK[14]
1	1	1	LUTMASK[7]	LUTMASK[11]	LUTMASK[15]

In arithmetic mode, the carry function uses the CI input and the PROP and GEN signals. If the PROP signal is 1, the CI input is propagated to carry; otherwise, the GEN signal is sent to carry.

The CO output is a dedicated fast connection to the CI of neighboring COMB4 primitive. It passes the inverted value of the carry function. The FCO is a general-purpose output from the COMB4 that can connect to any block. It passes the un-inverted value of the carry function.

Table 12: Arithmetic Mode Signals and Outputs

Signals			Outputs		
PROP	GEN	CI	CO	FCO	P
0	0	0	1	0	0
0	0	1	1	0	0
0	1	0	0	1	0
0	1	1	0	1	0
1	0	0	1	0	1
1	0	1	0	1	1
1	1	0	1	0	1
1	1	1	0	1	1

Figure 10: EFX_COMB4 Verilog HDL Instantiation

```
// Example COMB4 instantiation as logic function
    EFX_COMB4 # (
        .LUTMASK(16'h8000),   // 16-bit function mask
        .MODE("LOGIC")        // string, "LOGIC" or "ARITH"
    )
    comb4_inst (
        .I0 (I0),             // 1-bit I0 input
        .I1 (I1),             // 1-bit I1 input
        .I2 (I2),             // 1-bit I2 input
        .I3 (I3),             // 1-bit I0 input
        .O  (O),              // 1-bit Data-Out output
        .P  (P)               // 1-bit Propagate-Out output
    );

// Example COMB4 instantiation as arithmetic function
    EFX_COMB4 # (
        .LUTMASK(16'h8696),   // 16-bit function mask
        .MODE("ARITH")        // string, "LOGIC" or "ARITH"
    )
    comb4_inst (
        .I0 (I0),             // 1-bit I0 input
        .I1 (I1),             // 1-bit I1 input
        .CI (CI),             // 1-bit Carry-In input
        .O  (O),              // 1-bit Data-Out output
        .CO (CO),             // 1-bit Carry-Out output
        .FCO(FCO),            // 1-bit Flexible Carry-Out output
        .P  (P)               // 1-bit Propagate-Out output
    );
```

Figure 11: EFX_COMB4 VHDL Instantiation

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

entity COMB4_VHDL is
    port
    (
        din : in std_logic_vector(3 downto 0);
        fco   : out std_logic;
        p0,p1 : out std_logic;
        dout0 : out std_logic;
        dout1 : out std_logic
    );
end entity COMB4_VHDL;

architecture Behavioral of COMB4_VHDL is
begin

 EFX_COMB4_inst0 : EFX_COMB4
    generic map (
        LUTMASK => x"abcd",
        MODE => "LOGIC"
    )
    port map (
        I0 => din(0),
        I1 => din(1),
        I2 => din(2),
        I3 => din(3),
        P => p0,
        O => dout0
    );

 EFX_COMB4_inst1 : EFX_COMB4
    generic map (
        LUTMASK => x"8696",
        MODE => "ARITH"
    )
    port map (
        I0 => din(0),
        I1 => din(1),
        FCO => fco,
        P => p1,
        O => dout1
    );

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 12: EFX_FF Symbol
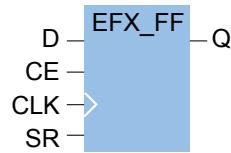
Table 13: EFX_FF Ports

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 14: 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 13: 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 14: 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_SRL8

8-Bit Shift Register

The EFX_SRL8 primitive is an 8-bit shift register with static or dynamic reads. It is controlled by a clock and clock enable, each of which can be positively or negatively triggered.

You must connect all of the EFX_SRL8 input ports. Connect unused EFX_SRL8 control ports to GND or VCC depending on their polarity. The Efinity® software issues a warning if a clock port is set to VCC or GND.

EFX_SRL8 Ports

Figure 15: EFX_SRL8 Symbol
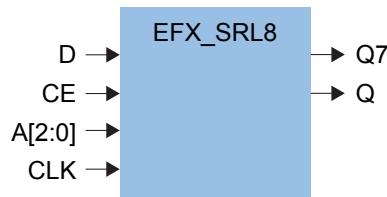
Table 15: EFX_SRL8 Ports

Port	Direction	Description
D	Input	Input data.
CE	Input	Clock enable.
CLK	Input	Clock.
A[2:0]	Input	Address of the shift register bit read.
Q	Output	Output data inverted value.
Q7	Output	Output of the last bit of the shift register used for cascading (non-inverted). Can only be connected to the D input of another shift register.

EFX_SRL8 Parameters

Table 16: EFX_SRL8 Parameters

Parameter	Allowed Values	Description
CLK_POLARITY	0, 1	0: Falling edge. 1: Rising edge.
CE_POLARITY	0, 1	0: Active-low. 1: Active-high.
INIT	8-bit value	Initial shift register content.

EFX_SRL8 Function

If the `CE` port is enabled, the `D` input value is captured into the first bit of the shift register on the active clock edge; the existing shift register content is shifted by 1. If the `CE` port is disabled, the value on the `D` input is ignored and the shift registers maintain their current values.

The inverted bit selected by the address ports `A` drives the `Q` output. The first bit is selected by value 3'b111, while the last bit is selected by 2'b000. For a fixed-size shift register, you can drive the `A` input ports by constant values. To read the shift-register content dynamically, drive the address ports with active signals.

Note: The shift register read is asynchronous to the clock. The eighth bit of the shift register always drives the output `Q7`. Use it to cascade to another EFX_SRL8 block to implement larger shift registers.

Use the `INIT` parameter to set the initial content of EFX_SRL8. If unused, EFX_SRL8 initializes to all zeros.

Table 17: EFX_SRL8 Function

A[2:0]	Q
111	1
110	2
101	3
100	4
011	5
010	6
001	7
000	8

Figure 16: EFX_SRL8 Verilog HDL Instantiation

```
// EFX_SRL8 Instantiation Template
EFX_SRL8 #(
    .CLK_POLARITY    (1'b1), // clk polarity
    .CE_POLARITY     (1'b1), // clk polarity
    .INIT            (8'h00) // 8-bit initial value
    )
    srl8_inst (
    .A   (A),               // 3-bit address select for Q
    .D   (D),               // 1-bit data-in
    .CLK (CLK),             // clock
    .CE  (CE),              // clock enable
    .Q   (Q),               // 1-bit data output
    .Q7  (Q7)               // 1-bit last shift register output
    );
```

Figure 17: EFX_SRL8 VHDL Instantiation

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

entity SRL8_VHDL is
    port
    (
        clk : in std_logic;
        ce  : in std_logic;
        d : in std_logic;
        a : in std_logic_vector (2 downto 0);
        q : out std_logic
    );
end entity SRL8_VHDL;

architecture Behavioral of SRL8_VHDL is
begin

 EFX_SRL8_inst0 : EFX_SRL8
    generic map (
        CLK_POLARITY => 1,
        CE_POLARITY => 1,
        INIT => x"11"
    )
    port map (
        D => d,
        CE => ce,
        CLK => clk,
        A  => a,
        -- Q7 => X, Q7 is not used
        Q => q
    );


end architecture Behavioral;
```

EFX_RAM10

10 Kbit RAM Block

The EFX_RAM10 primitive represents a configurable 10 Kbit RAM block[2] that supports a variety of widths and depths. All inputs have programmable invert capabilities, allowing positively or negatively triggered control signals.

The memory read and write ports can be configured into various modes for addressing the memory contents. The read and write ports support independently configured data widths.

Table 18: EFX_RAM10 Allowed Read and Write Mode Combinations

	Memory Depth x Data Width	Write Mode							
		512 x 16	**1024 x 8**	**2048 x 4**	**4096 x 2**	**8192 x 1**	**512 x 20**	**1024 x 10**	**2048 x 5**
Read Mode	**512 x 16**	✓	✓	✓	✓	✓			
	1024 x 8	✓	✓	✓	✓	✓			
	2048 x 4	✓	✓	✓	✓	✓			
	4096 x 2	✓	✓	✓	✓	✓			
	8192 x 1	✓	✓	✓	✓	✓			
	512 x 20						✓	✓	✓
	1024 x 10						✓	✓	✓
	2048 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 40 `INIT_N` parameters, each one represents 256 bits of the memory. The memory space covered by each `INIT_N` parameter uses this formula:

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

Used as RAM

When using the EFX_RAM10 primitive as RAM:

- You must connect the control ports (`WCLK`, `WE`, `WCLKE`, `WADDREN`, `RCLK`, `RE`, `RADDREN`, and `RST`). If the ports are unused, connect them to GND or VCC depending on their polarity.
- You can use an optional output register to improve t_{CO} at a cost of one stage of latency. It uses the same clock and read enable as the read port.
- If you use the same clock and clock polarity to control both the read and write ports of the memory, the memory is synchronous; otherwise it is asynchronous.

---

[2] 10 Kbits only available in 1024 x 10 and 2048 x 5 modes.

When writing to a synchronous memory and reading the same address, the read port has these modes:

- READ_FIRST–Old memory content is read. (default)
- WRITE_FIRST–Write data is passed to read port. To use this mode, `READ_WIDTH` and `WRITE_WIDTH` must be the same and the same signals must drive the `RADDR` and `WADDR` ports.
- READ_UNKNOWN–Read data becomes X. Use READ_UNKNOWN for asynchronous memory. Write operations ignore the read port, which guarantees that if the same address is written and read at the same time the write succeeds, but the read is undefined. This mode is inferred if the RAM uses two different read and write clocks or if the same clock is used but the HDL behavior is not deterministic.

The `WRITE_MODE` parameter controls the read port behavior.

Only the address lines valid in the mode may be used. Leave all other address lines unconnected. For example, only address bits `WADDR[8:0]` may be used in 512 x 16 write mode. You must connect all address lines for a mode. Connect required, unused address lines to GND. For example if the RAM is in 512 x 16 write mode but is only implementing a 64 x 2 memory address, `WADDR[12:9]` will be unconnected. Connect `WADDR[8:6]` to GND, `WADDR[5:0]` is used..

Leave unused data lines unconnected. For example if the RAM is in 512 x 16 write mode but is only implementing a 64 x 2 memory, `WDATA[19:2]` and `RDATA[19:2]` are unused and unconnected. `WDATA[1:0]` and `RDATA[1:0]` are connected and used.

Used as ROM

When using the EFX_RAM10 primitive as ROM:

- Connect `WE`, `WCLK`, `WCLKE`, and `WADDREN` to GND.
- Leave `WDATA` unconnected.
- Leave `WADDR` unconnected or connect to GND based on the write width you select.

Note: The write mode must be compatible with the read mode even though the write ports of the ROM are not used. For example, if the ROM is reading in 512 x 16 mode one compatible write mode is 1024 x 8.

The Efinity® software issues an error if the read or write clock is constant and issues warnings if the `WE`, `RE`, `WCLKE`, `WADDREN`, `RADDREN`, or `RST` ports are disabled. When implementing a ROM, no errors or warnings are given for the write port if `WCLK`, `WE`, `WADDREN`, and `WCLKE` are disabled.

`WADDREN` and `RADDREN` are enable ports for the write and read addresses. During normal operation they should be high using the value on the address lines to choose the address for reading or writing. When disabled (low) they act as an address stall and the read or write operation uses the previous address latched into the memory.

Byte Enable Support

The EFX_RAM10 block supports byte enable if the `WRITE_WIDTH` parameter is 16 or 20. In widths of 16 or 20, `WE[0]` controls writing to the lower half of the data and `WE[1]` controls writing to the upper half of the data.

Reset

The `RST` port can reset the output of the RAM and/or the RAM output register. The `RST_RAM` and `RST_OUTREG` parameters control the behavior.

- `RST_RAM` can be set to `NONE`, `ASYNC` (default), or `SYNC`. If set to `ASYNC`, the RAM output is reset to all 0s asynchronous to the `RCLK`. If it is `SYNC`, the output is reset to all 0s synchronously with the `RCLK`.
- `RST_OUTREG` can be set to `NONE` or `ASYNC` (default). This parameter is ignored if the output register is disabled.

EFX_RAM10 Ports

Figure 18: EFX_RAM10 Symbol
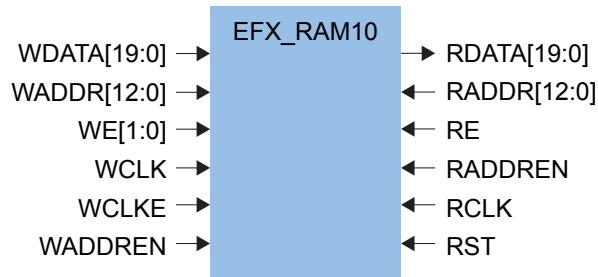
Table 19: EFX_RAM10 Ports

Port Name	Direction	Description
WDATA[19:0]	Input	Write data.
WADDR[12:0]	Input	Write address.
WE[1:0]	Input	Write enable.
WCLK	Input	Write clock.
WCLKE	Input	Write clock enable.
WADDREN	Input	Write address enable.
RDATA[19:0]	Output	Read data.
RADDR[12:0]	Input	Read address.
RE	Input	Read enable.
RADDREN	Input	Read address enable.
RCLK	Input	Read clock.
RST	Input	Reset the RAM output.

EFX_RAM10 Parameters

Table 20: EFX_RAM10 Parameters

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

Parameter Name	Allowed Values	Description
INIT_0, …INIT_F…, INIT_27	256-bit hexadecimal number	Initial RAM content.
READ_WIDTH WRITE_WIDTH	16	512 x 16 (default).
	8	1,024 x 8.
	4	2,048 x 4.
	2	4,096 x 2.
	1	8,192 x 1.
	20	512 x 20.
	10	1,024 x 10.
	5	2,048 x 5.
OUTPUT_REG	0, 1	0: disable output register (default). 1: enable output register.
RESET_RAM	NONE	RST signals does not affect the RAM output.
	ASYNC	RAM output resets asynchronously to RCLK.
	SYNC	RAM output resets synchronously to RCLK.
RESET_OUTREG	NONE	RST signals does not affect the RAM output register.
	ASYNC	RAM output register resets asynchronously to RCLK.
*<port name>*_POLARITY	0, 1	0: Active low 1: Active high (default)
WRITE_MODE	READ_FIRST	Old memory content is read. (default).
	WRITE_FIRST	Write data is passed to the read port. In this mode WCLKE must be enabled.
	READ_UNKNOWN	Read and writes are unsynchronized, therefore, the results of the address can conflict.

EFX_RAM10 Function

The EFXBRAM is physically implemented as a 512 x 20 memory array with decoder logic to map to the different read and write modes. The read and write ports are independent and the behavior is undefined when addresses conflict. The address at the physical memory array must not conflict.

Figure 19: EFX_RAM10 Verilog HDL Instantiation

```
EFX_RAM10 # (

    .WCLK_POLARITY    (1'b1),              // wclk polarity
    .WCLKE_POLARITY   (1'b1),              // wclke polarity
    .WADDREN_POLARITY (1'b1),              // waddren polarity
    .WE_POLARITY      (2'b11),             // we polarity
    .RCLK_POLARITY    (1'b1),              // rclk polarity
    .RE_POLARITY      (1'b1),              // re polarity
    .RST_POLARITY     (1'b1),              // rst polarity
    .RADDREN_POLARITY (1'b1),              // raddren polarity
    .READ_WIDTH       (16),                // read width
    .WRITE_WIDTH      (16),                // write width
    .OUTPUT_REG       (1'b0),              // Output register enable
    .WRITE_MODE       ("READ_FIRST"),      // write mode
    .RESET_RAM        ("ASYNC"),           // reset mode on ram
    .RESET_OUTREG     ("ASYNC"),           // reset mode on output register

    // 256-bit INIT string
    .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),
    .INIT_14 (256'h0000000000000000000000000000000000000000000000000000000000000000),
    .INIT_15 (256'h0000000000000000000000000000000000000000000000000000000000000000),
    .INIT_16 (256'h0000000000000000000000000000000000000000000000000000000000000000),
    .INIT_17 (256'h0000000000000000000000000000000000000000000000000000000000000000),
    .INIT_18 (256'h0000000000000000000000000000000000000000000000000000000000000000),
    .INIT_19 (256'h0000000000000000000000000000000000000000000000000000000000000000),
    .INIT_1A (256'h0000000000000000000000000000000000000000000000000000000000000000),
    .INIT_1B (256'h0000000000000000000000000000000000000000000000000000000000000000),
    .INIT_1C (256'h0000000000000000000000000000000000000000000000000000000000000000),
    .INIT_1D (256'h0000000000000000000000000000000000000000000000000000000000000000),
    .INIT_1E (256'h0000000000000000000000000000000000000000000000000000000000000000),
    .INIT_1F (256'h0000000000000000000000000000000000000000000000000000000000000000),
    .INIT_20 (256'h0000000000000000000000000000000000000000000000000000000000000000),
    .INIT_21 (256'h0000000000000000000000000000000000000000000000000000000000000000),
    .INIT_22 (256'h0000000000000000000000000000000000000000000000000000000000000000),
    .INIT_23 (256'h0000000000000000000000000000000000000000000000000000000000000000),
    .INIT_24 (256'h0000000000000000000000000000000000000000000000000000000000000000),
    .INIT_25 (256'h0000000000000000000000000000000000000000000000000000000000000000),
    .INIT_26 (256'h0000000000000000000000000000000000000000000000000000000000000000),
    .INIT_27 (256'h0000000000000000000000000000000000000000000000000000000000000000)
    )
    ram10_inst (
    .WCLK      (WCLK),          // write clk
    .WE        (WE),            // 2-bit write enable
    .WCLKE     (WCLKE),         // write clk enable
    .WADDREN   (WADDREN),       // write address enable
    .WDATA     (WDATA),         // write data input
    .WADDR     (WADDR),         // write address input
    .RCLK      (RCLK),          // read clk
    .RE        (RE),            // read enable
    .RST       (RST),           // reset
```

```
    .RADDREN    (RADDREN),      // read address enable
    .RDATA      (RDATA),        // read data output
    .RADDR      (RADDR)         // read address input
    );
```

Figure 20: EFX_RAM10 VHDL Instantiation

```
-- This RAM test design implements a
-- simple-dual-port RAM with
-- initial content.

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

entity ram10_VHDL is
    generic (
        AWIDTH : integer := 9;
        DWIDTH : integer := 16
    );
    port
    (
        wclk, wclke        : in std_logic;
        rclk, re, rst      : in std_logic;
        waddren, raddren   : in std_logic;
        we                 : in std_logic_vector(1 downto 0);
        wdata              : in std_logic_vector(DWIDTH-1 downto 0);
        waddr, raddr       : in std_logic_vector(AWIDTH-1 downto 0);
        rdata              : out std_logic_vector(DWIDTH-1 downto 0)
    );
end entity ram10_VHDL;

architecture Behavioral of ram10_VHDL is
    constant INIT_0 : unsigned(255 downto 0) :=
 x"E0E1E2E3E4E5E6E7E8E9EAEBECEDEEEFF0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF";
    constant INIT_1 : unsigned(255 downto 0) :=
 x"C0C1C2C3C4C5C6C7C8C9CACBCCCDCECFD0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF";
    constant INIT_2 : unsigned(255 downto 0) :=
 x"A0A1A2A3A4A5A6A7A8A9AAABACADAEAFB0B1B2B3B4B5B6B7B8B9BABBBCBDBEBF";
    constant INIT_3 : unsigned(255 downto 0) :=
 x"8081828384858687888998A8B8C8D8E8F909192939495969798999A9B9C9D9E9F";
    constant INIT_4 : unsigned(255 downto 0) :=
 x"6061626364656667768696A6B6C6D6E6F707172737475767778797A7B7C7D7E7F";
    constant INIT_5 : unsigned(255 downto 0) :=
 x"4041424344454647748494A4B4C4D4E4F505152535455565758595A5B5C5D5E5F";
    constant INIT_6 : unsigned(255 downto 0) :=
 x"2021222324252627728292A2B2C2D2E2F30313233343536373839 3A3B3C3D3E3F";
    constant INIT_7 : unsigned(255 downto 0) :=
 x"000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F";
    constant INIT_8 : unsigned(255 downto 0) :=
 x"E0E1E2E3E4E5E6E7E8E9EAEBECEDEEEFF0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF";
    constant INIT_9 : unsigned(255 downto 0) :=
 x"C0C1C2C3C4C5C6C7C8C9CACBCCCDCECFD0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF";
    constant INIT_A : unsigned(255 downto 0) :=
 x"A0A1A2A3A4A5A6A7A8A9AAABACADAEAFB0B1B2B3B4B5B6B7B8B9BABBBCBDBEBF";
    constant INIT_B : unsigned(255 downto 0) :=
 x"8081828384858687888998A8B8C8D8E8F909192939495969798999A9B9C9D9E9F";
    constant INIT_C : unsigned(255 downto 0) :=
 x"6061626364656667768696A6B6C6D6E6F707172737475767778797A7B7C7D7E7F";
    constant INIT_D : unsigned(255 downto 0) :=
 x"4041424344454647748494A4B4C4D4E4F505152535455565758595A5B5C5D5E5F";
    constant INIT_E : unsigned(255 downto 0) :=
 x"2021222324252627728292A2B2C2D2E2F30313233343536373839 3A3B3C3D3E3F";
    constant INIT_F : unsigned(255 downto 0) :=
 x"000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F";
    constant INIT_10 : unsigned(255 downto 0) :=
 x"E0E1E2E3E4E5E6E7E8E9EAEBECEDEEEFF0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF";
    constant INIT_11 : unsigned(255 downto 0) :=
 x"C0C1C2C3C4C5C6C7C8C9CACBCCCDCECFD0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF";
    constant INIT_12 : unsigned(255 downto 0) :=
 x"A0A1A2A3A4A5A6A7A8A9AAABACADAEAFB0B1B2B3B4B5B6B7B8B9BABBBCBDBEBF";
    constant INIT_13 : unsigned(255 downto 0) :=
 x"8081828384858687888998A8B8C8D8E8F909192939495969798999A9B9C9D9E9F";
    constant INIT_14 : unsigned(255 downto 0) :=
 x"6061626364656667768696A6B6C6D6E6F707172737475767778797A7B7C7D7E7F";
    constant INIT_15 : unsigned(255 downto 0) :=
 x"4041424344454647748494A4B4C4D4E4F505152535455565758595A5B5C5D5E5F";
    constant INIT_16 : unsigned(255 downto 0) :=
 x"2021222324252627728292A2B2C2D2E2F30313233343536373839 3A3B3C3D3E3F";
```

```vhdl
    constant INIT_17 : unsigned(255 downto 0) :=
x"00010203040506070809A0B0C0D0E0F101112131415161718191A1B1C1D1E1F";
    constant INIT_18 : unsigned(255 downto 0) :=
x"E0E1E2E3E4E5E6E7E8E9EAEBECEDEEEFF0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF";
    constant INIT_19 : unsigned(255 downto 0) :=
x"C0C1C2C3C4C5C6C7C8C9CACBCCCDCECFD0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF";
    constant INIT_1A : unsigned(255 downto 0) :=
x"A0A1A2A3A4A5A6A7A8A9AAABACADAEAFB0B1B2B3B4B5B6B7B8B9BABBBCBDBEBF";
    constant INIT_1B : unsigned(255 downto 0) :=
x"80818283848586878889A8A8B8C8D8E8F909192939495969798999A9B9C9D9E9F";
    constant INIT_1C : unsigned(255 downto 0) :=
x"606162636465666768696A6B6C6D6E6F707172737475767778797A7B7C7D7E7F";
    constant INIT_1D : unsigned(255 downto 0) :=
x"404142434445464748494A4B4C4D4E4F505152535455565758595A5B5C5D5E5F";
    constant INIT_1E : unsigned(255 downto 0) :=
x"202122232425262728292A2B2C2D2E2F303132333435363738393A3B3C3D3E3F";
    constant INIT_1F : unsigned(255 downto 0) :=
x"00010203040506070809A0B0C0D0E0F101112131415161718191A1B1C1D1E1F";
    constant INIT_20 : unsigned(255 downto 0) :=
x"E0E1E2E3E4E5E6E7E8E9EAEBECEDEEEFF0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF";
    constant INIT_21 : unsigned(255 downto 0) :=
x"C0C1C2C3C4C5C6C7C8C9CACBCCCDCECFD0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF";
    constant INIT_22 : unsigned(255 downto 0) :=
x"A0A1A2A3A4A5A6A7A8A9AAABACADAEAFB0B1B2B3B4B5B6B7B8B9BABBBCBDBEBF";
    constant INIT_23 : unsigned(255 downto 0) :=
x"80818283848586878889A8A8B8C8D8E8F909192939495969798999A9B9C9D9E9F";
    constant INIT_24 : unsigned(255 downto 0) :=
x"606162636465666768696A6B6C6D6E6F707172737475767778797A7B7C7D7E7F";
    constant INIT_25 : unsigned(255 downto 0) :=
x"404142434445464748494A4B4C4D4E4F505152535455565758595A5B5C5D5E5F";
    constant INIT_26 : unsigned(255 downto 0) :=
x"202122232425262728292A2B2C2D2E2F303132333435363738393A3B3C3D3E3F";
    constant INIT_27 : unsigned(255 downto 0) :=
x"00010203040506070809A0B0C0D0E0F101112131415161718191A1B1C1D1E1F";
begin

  ram : EFX_RAM10
    generic map (
      READ_WIDTH  => DWIDTH,
      WRITE_WIDTH => DWIDTH,

      WCLK_POLARITY => 1,
      WCLKE_POLARITY => 1,
      WADDREN_POLARITY => 1,
      WE_POLARITY => b"11",
      RCLK_POLARITY => 1,
      RE_POLARITY => 1,
      RST_POLARITY => 1,
      RADDREN_POLARITY => 1,

      WRITE_MODE   => "READ_FIRST",
      RESET_RAM    => "ASYNC",
      RESET_OUTREG => "ASYNC",

      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,
      INIT_14 => INIT_14,
      INIT_15 => INIT_15,
      INIT_16 => INIT_16,
      INIT_17 => INIT_17,
      INIT_18 => INIT_18,
      INIT_19 => INIT_19,
      INIT_1A => INIT_1A,
      INIT_1B => INIT_1B,
      INIT_1C => INIT_1C,
      INIT_1D => INIT_1D,
      INIT_1E => INIT_1E,
```

```vhdl
        INIT_1F => INIT_1F,
        INIT_20 => INIT_20,
        INIT_21 => INIT_21,
        INIT_22 => INIT_22,
        INIT_23 => INIT_23,
        INIT_24 => INIT_24,
        INIT_25 => INIT_25,
        INIT_26 => INIT_26,
        INIT_27 => INIT_27
    )
    port map (
        WCLK => wclk,
        WCLKE => wclke,
        WADDREN => waddren,
        RCLK => rclk,
        RE => re,
        RST => rst,
        RADDREN => raddren,
        WE => we,
        WDATA => wdata,
        WADDR => waddr,
        RDATA => rdata,
        RADDR => raddr
    );

end architecture Behavioral;
```

EFX_DPRAM10

10 Kbit True-Dual-Port RAM Block

The EFX_DPRAM10 block represents a 10 Kbit true-dual-port RAM block[3] that can be configured to support a variety of widths and depths. All inputs have programmable invert capabilities, which allows you to trigger the control signals positively or negatively.

The memory A and B ports can be configured into several modes for addressing the memory contents. The read and write widths of a single port can be configured to allow writing in one data width while reading another.

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, if reading in 1024 x 8 mode and writing in 2048 x 4 mode the address must be 11 bits wide to be able to address all 2048 words being written. The write data would be 4 bits wide and the read data would be 8 bits wide. Only the upper 10 bits of the address port would be used during reading because it can only read 1024 words from the memory.

Table 21: EFX_DPRAM10 Allowed Read and Write Mode Combinations

	Memory Depth x Data Width	Write Mode					
		1024 x 8	2048 x 4	4096 x 2	8192 x 1	1024 x 10	2048 x 5
Read Mode	1024 x 8	✓	✓	✓	✓		
	2048 x 4	✓	✓	✓	✓		
	4096 x 2	✓	✓	✓	✓		
	8192 x 1	✓	✓	✓	✓		
	1024 x 10					✓	✓
	2048 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 content of the RAM using INIT_N parameters. Each INIT_N parameter represents 256 bits of the memory. The 40 INIT_N parameters cover the entire 10K memory content. The memory space covered by each INIT_N parameter uses this formula:

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

Used as RAM

When connecting the ports, use the following guidelines:

---

[3] 10 Kbits only available in 1024 x 10 and 2048 x 5 modes.

- You must connect the BRAM control ports (`CLKA`, `WEA`, `CLKEA`, `ADDRENA`, `RSTA`, `CLKB`, `WEB`, `CLKEB`, `ADDRENB`, `RSTB`) must be connected. If the ports are unused, connect them to GND or VCC depending on their polarity.
- If you want to disable a RAM port (A or B),
  - Disable all `CLK`, `WE`, `CLKE`, and `ADDR` ports.
  - `WDATA` can be disabled or left unconnected.
  - Leave `RDATA` unconnected.

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 signal and clock enable as the port.

When writing to a memory port the behavior of the read port can be one of the following:

- READ_FIRST–Old memory content is read. (default)
- WRITE_FIRST–Write data is passed to read port.
- NO_CHANGE–Previously read data is held.

The `WRITE_MODE_A/B` parameters control this behavior.

When the same clock and clock polarity is used to control both ports of the memory, the memory is synchronous; otherwise it is asynchronous.

If the same address is read from or written to by both ports, an address collision can occur. The behavior in this situation is:

- If the memory is synchronous and both ports are using WRITE_FIRST mode, you can write to one port and read from the other.
- If both ports write to the same address and the write data is identical, the write succeeds; otherwise the value written is unknown. In WRITE_FIRST mode, there is a special bypass path that guarantees that port will read the same data that is written, even if an address collision occurs.
- If the memory is asynchronous and a read to one port and a write to the other port occurs at the same address, there is an address collision. The write succeeds, and the read data is unknown.

You can only use the address lines that are valid for the mode you are using. For example only address bits `ADDRA[9:0]` can be used if port A is in 1024 x 8 mode; all other address lines must be left unconnected. 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 RAM port B is in 1024 x 8 mode but is only implementing a 64 x 2 memory:

- `ADDRB[12:10]` are unconnected.
- `ADDRB[9:6]` are connected to GND.
- `ADDRB[5:0]` are used.

Leave unused data lines unconnected. For example, if the RAM port A is in 1024 x 8 mode but is only implementing a 64 x 2 memory, `WDATAA[19:2]` and `RDATAA[19:2]` are unused and left unconnected. `WDATAA[1:0]` and `RDATAA[1:0]` are connected and used.

Used as ROM

When implementing a ROM, connect `WEA` and `WEB` to GND. Leave `WDATAA` and `WDATAB` unconnected or disabled.

Enable

The `ADDRENA` and `ADDRENB` are enable ports for the A and B addresses. During normal operation they are high and use the value on the address lines to choose the address for reading and/or writing. When disabled (low) they act as an address stall and the read and/or write operation uses the previous address latched into the memory.

Reset

The `RSTA` and `RSTB` ports can reset the output of the RAM and/or the RAM output register. The `RST_RAM_A/B` and `RST_OUTREG_A/B` parameters control the behavior. The default value is `ASYNC`. `OUTPUT_REG_A/B` is ignored if the output register is not enabled.

EFX_DPRAM10 Ports

Figure 21: EFX_DPRAM10 Symbol
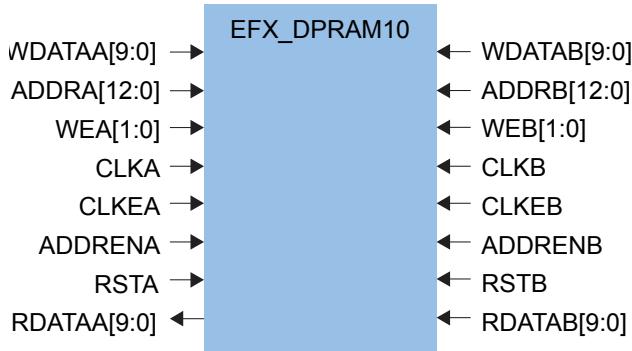
Table 22: EFX_DPRAM10 Ports

Port Name	Direction	Description
WDATAA[9:0] WDATAB[9:0]	Input	Write data port A/B.
WEA WEB	Input	Write enable port A/B.
ADDRA[12:0] ADDRB[12:0]	Input	Address port A/B.
ADDRENA ADDRENB	Input	Address enable port A/B.
CLKA CLKB	Input	Clock port A/B.
CLKEA CLKEB	Input	Clock enable port A/B.
RDATAA[9:0] RDATAB[9:0]	Output	Read data port A/B.
RSTA RSTB	Input	Reset port A/B.

EFX_DPRAM10 Parameters

Table 23: EFX_DPRAM10 Parameters

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

Parameter Name	Allowed Values	Description
INIT_0, …INIT_F…, INIT_27	256 bit hexadecimal number	Initial RAM content (default = 0).
READ_WIDTH_A READ_WIDTH_B WRITE_WIDTH_A WRITE_WIDTH_B	8	1,024 x 8 (default).
	4	2,048 x 4.
	2	4,096 x 2.
	1	8,192 x 1.
	10	1,024 x 10.
	5	2,048 x 5.
WRITE_MODE_A WRITE_MODE_B	READ_FIRST	Old data is output to RDATA.
	WRITE_FIRST	New data is output RDATA.
	NO_CHANGE	RDATA holds its last value.
OUTPUT_REG_A OUTPUT_REG_B	0, 1	0: disable output register (default). 1: enable output register.
RESET_RAM	NONE	RST signals do not affect the RAM output.
	ASYNC	RAM output resets asynchronously to RCLK.
	SYNC	RAM output resets synchronously to RCLK.
RESET_OUTREG	NONE	RST signals do not affect the RAM output register.
	ASYNC	RAM output register resets asynchronously to RCLK.
*<port name>*_POLARITY	0, 1	0: active low 1: active high (default)

EFX_DPRAM10 Function

The EFXBRAM is physically implemented as a 512 x 20 memory array with decoder logic to map to the different 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 22: EFX_DPRAM10 Verilog HDL Instantiation

```
// EFX_DPRAM10 Instantiation Template
EFX_DPRAM10 # (

    .CLKA_POLARITY    (1'b1),              // clka polarity
    .CLKEA_POLARITY   (1'b1),              // clkea polarity
    .WEA_POLARITY     (1'b1),              // wea polarity
    .ADDRENA_POLARITY (1'b1),              // addrena polarity
    .RSTA_POLARITY    (1'b1),              // rsta polarity
    .CLKB_POLARITY    (1'b1),              // clkb polarity
    .CLKEB_POLARITY   (1'b1),              // clkeb polarity
    .WEB_POLARITY     (1'b1),              // web polarity
    .ADDRENB_POLARITY (1'b1),              // addrenb polarity
    .RSTB_POLARITY        (1'b1),              // rstb polarity
    .READ_WIDTH_A     (8),                 // A-port read width
    .WRITE_WIDTH_A        (8),                 // A-port write width
    .READ_WIDTH_B     (8),                 // B-port read width
    .WRITE_WIDTH_B    (8),                 // B-port write width
    .OUTPUT_REG_A     (1'b0),              // A-port output register enable
    .OUTPUT_REG_B     (1'b0),              // B-port output register enable
    .WRITE_MODE_A     ("READ_FIRST"),      // A-port write mode
    .WRITE_MODE_B     ("READ_FIRST"),      // B-port write mode
    .RESET_RAM_A      ("ASYNC"),           // A-port reset mode on ram
    .RESET_RAM_B      ("ASYNC"),           // B-port reset mode on ram
    .RESET_OUTREG_A   ("ASYNC"),           // A-port reset mode on output register
    .RESET_OUTREG_B   ("ASYNC"),           // B-port reset mode on output register

    // 256-bit INIT string
    .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),
    .INIT_14 (256'h0000000000000000000000000000000000000000000000000000000000000000),
    .INIT_15 (256'h0000000000000000000000000000000000000000000000000000000000000000),
    .INIT_16 (256'h0000000000000000000000000000000000000000000000000000000000000000),
    .INIT_17 (256'h0000000000000000000000000000000000000000000000000000000000000000),
    .INIT_18 (256'h0000000000000000000000000000000000000000000000000000000000000000),
    .INIT_19 (256'h0000000000000000000000000000000000000000000000000000000000000000),
    .INIT_1A (256'h0000000000000000000000000000000000000000000000000000000000000000),
    .INIT_1B (256'h0000000000000000000000000000000000000000000000000000000000000000),
    .INIT_1C (256'h0000000000000000000000000000000000000000000000000000000000000000),
    .INIT_1D (256'h0000000000000000000000000000000000000000000000000000000000000000),
    .INIT_1E (256'h0000000000000000000000000000000000000000000000000000000000000000),
    .INIT_1F (256'h0000000000000000000000000000000000000000000000000000000000000000),
    .INIT_20 (256'h0000000000000000000000000000000000000000000000000000000000000000),
    .INIT_21 (256'h0000000000000000000000000000000000000000000000000000000000000000),
    .INIT_22 (256'h0000000000000000000000000000000000000000000000000000000000000000),
    .INIT_23 (256'h0000000000000000000000000000000000000000000000000000000000000000),
    .INIT_24 (256'h0000000000000000000000000000000000000000000000000000000000000000),
    .INIT_25 (256'h0000000000000000000000000000000000000000000000000000000000000000),
    .INIT_26 (256'h0000000000000000000000000000000000000000000000000000000000000000),
    .INIT_27 (256'h0000000000000000000000000000000000000000000000000000000000000000)
    )
    dpram10_inst (
```

```
    .CLKA        (CLKA),        // A-port clk
    .WEA         (WEA),         // A-port write enable
    .CLKEA       (CLKEA),       // A-port clk enable
    .RSTA        (RSTA),        // A-port reset
    .ADDRENA     (ADDRENA),     // A-port address enable
    .WDATAA      (WDATAA),      // A-port write data input
    .ADDRA       (ADDRA),       // A-port address input
    .RDATAA      (RDATAA),      // A-port read address output

    .CLKB        (CLKB),        // B-port clk
    .WEB         (WEB),         // B-port write enable
    .CLKEB       (CLKEB),       // B-port clk enable
    .RSTB        (RSTB),        // B-port reset
    .ADDRENB     (ADDRENB),     // B-port address enable
    .WDATAB      (WDATAB),      // B-port write data input
    .ADDRB       (ADDRB),       // B-port address input
    .RDATAB      (RDATAB)       // B-port read address output
);
```

Figure 23: EFX_DPRAM1 VHDL Instantiation

```
-- This RAM test design implements a
-- true-dual-port RAM with
-- initial content.

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

entity dpram10_VHDL is
    generic (
        AWIDTH : integer := 10;
        DWIDTH : integer := 8
    );
    port
    (
        clka, wea, clkea  : in std_logic;
        rsta, addrena     : in std_logic;
        addra             : in std_logic_vector(AWIDTH-1 downto 0);
        wdataa            : in std_logic_vector(DWIDTH-1 downto 0);
        rdataa            : out std_logic_vector(DWIDTH-1 downto 0);
        clkb, web, clkeb  : in std_logic;
        rstb, addrenb     : in std_logic;
        addrb             : in std_logic_vector(AWIDTH-1 downto 0);
        wdatab            : in std_logic_vector(DWIDTH-1 downto 0);
        rdatab            : out std_logic_vector(DWIDTH-1 downto 0)
    );
end entity dpram10_VHDL;

architecture Behavioral of dpram10_VHDL is
    constant INIT_0 : unsigned(255 downto 0) :=
 x"E0E1E2E3E4E5E6E7E8E9EAEBECEDEEEFF0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF";
    constant INIT_1 : unsigned(255 downto 0) :=
 x"C0C1C2C3C4C5C6C7C8C9CACBCCCDCECFD0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF";
    constant INIT_2 : unsigned(255 downto 0) :=
 x"A0A1A2A3A4A5A6A7A8A9AAABACADAEAFB0B1B2B3B4B5B6B7B8B9BABBBCBDBEBF";
    constant INIT_3 : unsigned(255 downto 0) :=
 x"808182838485868788898A8B8C8D8E8F909192939495969798999A9B9C9D9E9F";
    constant INIT_4 : unsigned(255 downto 0) :=
 x"606162636465666768696A6B6C6D6E6F707172737475767778797A7B7C7D7E7F";
    constant INIT_5 : unsigned(255 downto 0) :=
 x"404142434445464748494A4B4C4D4E4F505152535455565758595A5B5C5D5E5F";
    constant INIT_6 : unsigned(255 downto 0) :=
 x"202122232425262728292A2B2C2D2E2F303132333435363738393A3B3C3D3E3F";
    constant INIT_7 : unsigned(255 downto 0) :=
 x"000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F";
    constant INIT_8 : unsigned(255 downto 0) :=
 x"E0E1E2E3E4E5E6E7E8E9EAEBECEDEEEFF0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF";
    constant INIT_9 : unsigned(255 downto 0) :=
 x"C0C1C2C3C4C5C6C7C8C9CACBCCCDCECFD0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF";
    constant INIT_A : unsigned(255 downto 0) :=
 x"A0A1A2A3A4A5A6A7A8A9AAABACADAEAFB0B1B2B3B4B5B6B7B8B9BABBBCBDBEBF";
    constant INIT_B : unsigned(255 downto 0) :=
 x"808182838485868788898A8B8C8D8E8F909192939495969798999A9B9C9D9E9F";
    constant INIT_C : unsigned(255 downto 0) :=
 x"606162636465666768696A6B6C6D6E6F707172737475767778797A7B7C7D7E7F";
    constant INIT_D : unsigned(255 downto 0) :=
 x"404142434445464748494A4B4C4D4E4F505152535455565758595A5B5C5D5E5F";
    constant INIT_E : unsigned(255 downto 0) :=
 x"202122232425262728292A2B2C2D2E2F303132333435363738393A3B3C3D3E3F";
```

```vhdl
    constant INIT_F : unsigned(255 downto 0) :=
x"000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F";
    constant INIT_10 : unsigned(255 downto 0) :=
x"E0E1E2E3E4E5E6E7E8E9EAEBECEDEEEFF0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF";
    constant INIT_11 : unsigned(255 downto 0) :=
x"C0C1C2C3C4C5C6C7C8C9CACBCCCDCECFD0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF";
    constant INIT_12 : unsigned(255 downto 0) :=
x"A0A1A2A3A4A5A6A7A8A9AAABACADAEAFB0B1B2B3B4B5B6B7B8B9BABBBCBDBEBF";
    constant INIT_13 : unsigned(255 downto 0) :=
x"808182838485868788898A8B8C8D8E8F909192939495969798999A9B9C9D9E9F";
    constant INIT_14 : unsigned(255 downto 0) :=
x"606162636465666768696A6B6C6D6E6F707172737475767778797A7B7C7D7E7F";
    constant INIT_15 : unsigned(255 downto 0) :=
x"404142434445464748494A4B4C4D4E4F505152535455565758595A5B5C5D5E5F";
    constant INIT_16 : unsigned(255 downto 0) :=
x"202122232425262728292A2B2C2D2E2F303132333435363738393A3B3C3D3E3F";
    constant INIT_17 : unsigned(255 downto 0) :=
x"000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F";
    constant INIT_18 : unsigned(255 downto 0) :=
x"E0E1E2E3E4E5E6E7E8E9EAEBECEDEEEFF0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF";
    constant INIT_19 : unsigned(255 downto 0) :=
x"C0C1C2C3C4C5C6C7C8C9CACBCCCDCECFD0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF";
    constant INIT_1A : unsigned(255 downto 0) :=
x"A0A1A2A3A4A5A6A7A8A9AAABACADAEAFB0B1B2B3B4B5B6B7B8B9BABBBCBDBEBF";
    constant INIT_1B : unsigned(255 downto 0) :=
x"808182838485868788898A8B8C8D8E8F909192939495969798999A9B9C9D9E9F";
    constant INIT_1C : unsigned(255 downto 0) :=
x"606162636465666768696A6B6C6D6E6F707172737475767778797A7B7C7D7E7F";
    constant INIT_1D : unsigned(255 downto 0) :=
x"404142434445464748494A4B4C4D4E4F505152535455565758595A5B5C5D5E5F";
    constant INIT_1E : unsigned(255 downto 0) :=
x"202122232425262728292A2B2C2D2E2F303132333435363738393A3B3C3D3E3F";
    constant INIT_1F : unsigned(255 downto 0) :=
x"000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F";
    constant INIT_20 : unsigned(255 downto 0) :=
x"E0E1E2E3E4E5E6E7E8E9EAEBECEDEEEFF0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF";
    constant INIT_21 : unsigned(255 downto 0) :=
x"C0C1C2C3C4C5C6C7C8C9CACBCCCDCECFD0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF";
    constant INIT_22 : unsigned(255 downto 0) :=
x"A0A1A2A3A4A5A6A7A8A9AAABACADAEAFB0B1B2B3B4B5B6B7B8B9BABBBCBDBEBF";
    constant INIT_23 : unsigned(255 downto 0) :=
x"808182838485868788898A8B8C8D8E8F909192939495969798999A9B9C9D9E9F";
    constant INIT_24 : unsigned(255 downto 0) :=
x"606162636465666768696A6B6C6D6E6F707172737475767778797A7B7C7D7E7F";
    constant INIT_25 : unsigned(255 downto 0) :=
x"404142434445464748494A4B4C4D4E4F505152535455565758595A5B5C5D5E5F";
    constant INIT_26 : unsigned(255 downto 0) :=
x"202122232425262728292A2B2C2D2E2F303132333435363738393A3B3C3D3E3F";
    constant INIT_27 : unsigned(255 downto 0) :=
x"000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F";
begin

    ram : EFX_DPRAM10
      generic map (
        READ_WIDTH_A  => DWIDTH,
        WRITE_WIDTH_A => DWIDTH,
        READ_WIDTH_B  => DWIDTH,
        WRITE_WIDTH_B => DWIDTH,

        CLKA_POLARITY     => 1,
        CLKEA_POLARITY    => 1,
        WEA_POLARITY      => 1,
        ADDRENA_POLARITY  => 1,
        RSTA_POLARITY     => 1,
        CLKB_POLARITY     => 1,
        CLKEB_POLARITY    => 1,
        WEB_POLARITY      => 1,
        ADDRENB_POLARITY  => 1,
        RSTB_POLARITY     => 1,

        WRITE_MODE_A      => "READ_FIRST",
        RESET_RAM_A       => "ASYNC",
        RESET_OUTREG_A    => "ASYNC",
        WRITE_MODE_B      => "READ_FIRST",
        RESET_RAM_B       => "ASYNC",
        RESET_OUTREG_B    => "ASYNC",

        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,
        INIT_14 => INIT_14,
        INIT_15 => INIT_15,
        INIT_16 => INIT_16,
        INIT_17 => INIT_17,
        INIT_18 => INIT_18,
        INIT_19 => INIT_19,
        INIT_1A => INIT_1A,
        INIT_1B => INIT_1B,
        INIT_1C => INIT_1C,
        INIT_1D => INIT_1D,
        INIT_1E => INIT_1E,
        INIT_1F => INIT_1F,
        INIT_20 => INIT_20,
        INIT_21 => INIT_21,
        INIT_22 => INIT_22,
        INIT_23 => INIT_23,
        INIT_24 => INIT_24,
        INIT_25 => INIT_25,
        INIT_26 => INIT_26,
        INIT_27 => INIT_27
    )
    port map (
      CLKA => clka,
      CLKEA => clkea,
      ADDRENA => addrena,
      WEA => wea,
      RSTA => rsta,
      WDATAA => wdataa,
      ADDRA => addra,
      RDATAA => rdataa,
      CLKB => clkb,
      CLKEB => clkeb,
      ADDRENB => addrenb,
      WEB => web,
      RSTB => rstb,
      WDATAB => wdatab,
      ADDRB => addrb,
      RDATAB => rdatab
    );

end architecture Behavioral;
```

DSP Block

The Titanium FPGA has high-performance, complex DSP blocks that can perform multiplication, addition, subtraction, accumulation, and 4-bit variable right shifting. The 4-bit variable right shift supports one lane in normal mode, two lanes in dual mode and four lanes in quad mode. Each DSP block has four modes, which support the following multiplication operations:

- *Normal*—One 19 x 18 integer multiplication with 48-bit addition/subtraction.
- *Dual*—One 11 x 10 integer multiplication and one 8 x 8 integer multiplication with two 24-bit additions/subtractions.
- *Quad*—One 7 x 6 integer multiplication and three 4 x 4 integer multiplications with four 12-bit additions/subtractions.

> (!) **Important:** The 7 x 6 Quad mode output is truncated to 12-bit.

- *Float*—One fused-multiply-add/subtract/accumulate (FMA) BFLOAT16 multiplication.

The integer multipliers can represent signed or unsigned values based on the `SIGNED` parameter. When multiple EFX_DSP12 or EFX_DSP24 primitives are mapped to the same DSP block, they must have the same `SIGNED` value. The inputs to the multiplier are the A and B data inputs. Optionally, you can use the result of the multiplier in an addition or subtraction operation.

Figure 24: DSP Block Diagram
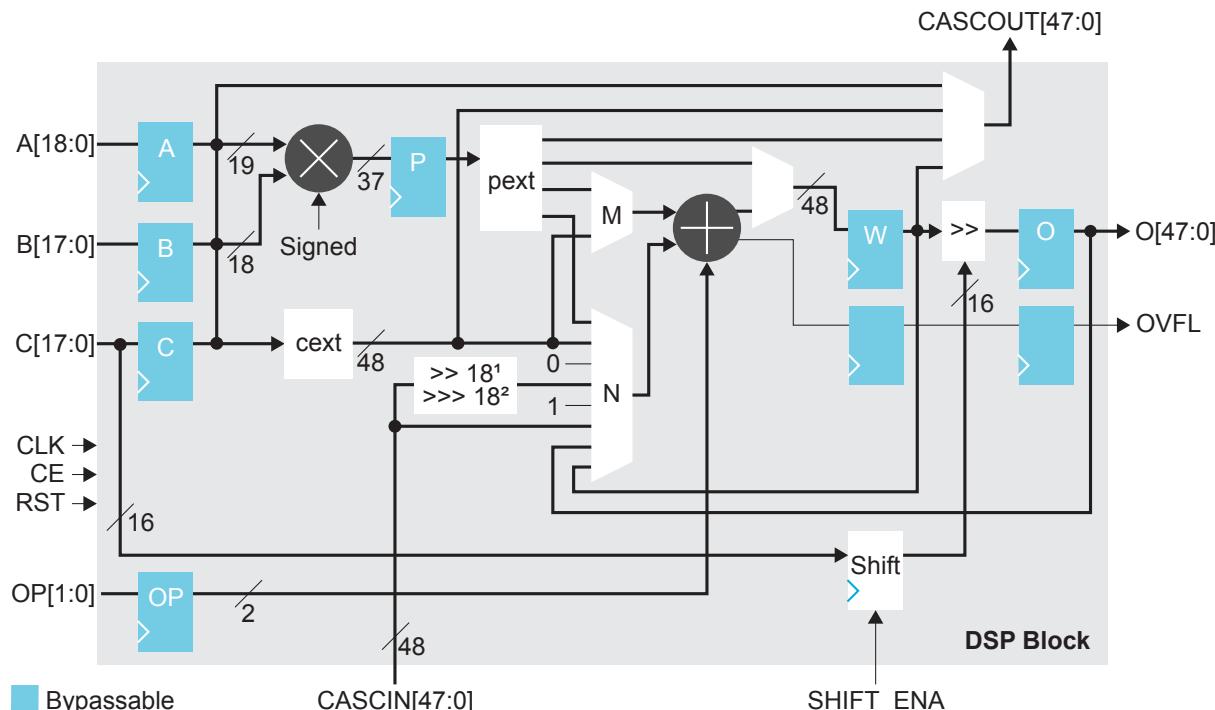
1. Logical right-shift-by-18.
2. Arithmetic right-shift-by-18.

> 📖 **Learn more:** Refer to the Quantum® Titanium Primitives User Guide for details on the Titanium DSP block primitives.

DSP Block Modes

The mode of the DSP Block primitive determines how the block interprets data inputs and outputs, and how the block performs extension and shifting functions.

Table 24: Bit Slicing by Mode

MODE	Multiplier Size	A	B	C	Shift	O
NORMAL	19 x 18	A[18:0]	B[17:0]	C[17:0]	C[3:0]	O[47:0]
DUAL	11 x 10	A[18:8]	B[17:8]	C[17:8]	C[7:4]	O[47:24]
	8 x 8	A[7:0]	B[7:0]	C[7:0]	C[3:0]	O[23:0]
QUAD	7 x 6	A[18:12]	B[17:12]	C[17:12]	C[15:12]	O[47:36]
	4 x 4	A[11:8]	B[11:8]	C[11:8]	C[11:8]	O[35:24]
	4 x 4	A[7:4]	B[7:4]	C[7:4]	C[7:4]	O[23:12]
	4 x 4	A[3:0]	B[3:0]	C[3:0]	C[3:0]	O[11:0]
BFLOAT	16 x 16	A[15:0]	B[15:0]	C[15:0]	N/A	O[47:0]

Normal Mode

In NORMAL mode the multiplier implements a 19 x 18 integer multiplier with a 37-bit result P. The multiplier may be signed or unsigned depending on the SIGNED parameter. The result is extended to 48 bits.

The C input is 18 bits and is extended to 48 bits based on the C_EXT parameter.

The logical shifter can shift the result 0 to 15 bits to the right. The shift is arithmetic or logical depending on the sign of the DSP Block. The C inputs capture the shift value when the SHIFT_ENA port is asserted.

Dual Mode

In DUAL mode the multiplier implements an 11 x 10 integer multiplier with a 21 bit result and an 8 x 8 integer multiplier with a 16-bit result. The results of both multipliers are extended to 24 bits.

The C input is divided between the two data paths. Each data path is extended to 24 bits based on the C_EXT parameter.

The logical shifter can shift the result 0 to 15 bits to the right. The shift is arithmetic or logical depending on the sign of the DSP Block. The C inputs capture the shift value when the SHIFT_ENA port is asserted.

Quad Mode

In QUAD mode the multiplier implements one 7 x 6 integer multiplier with a 13-bit result and three 4 x 4 integer multipliers with an 8-bit result. The results of the three 4 x 4 multipliers are extended to 12 bits.

The 7 x 6 multiplier's 13-bit result is truncated to 12 bits. If P_EXT is ALIGN_RIGHT, the MSB is removed; if it is ALIGN_LEFT, the LSB is removed.

The C input is divided between the four data paths. Each data path is extended to 12 bits based on the C_EXT parameter.

The logical shifter can shift the result 0 to 15 bits to the right. The shift is arithmetic or logical depending on the sign of the DSP Block. The C inputs capture the shift value when the SHIFT_ENA port is asserted.

BFLOAT Mode

In BFLOAT mode the multiplier implements one fused multiply-add (FMA) function. The A, B, and C inputs are represented as BFLOAT16 while the accumulator and output are represented as FP32. In BFLOAT mode:

- Set the `A_REG`, `B_REG`, `P_REG`, `OP_REG`, and `W_REG` parameters to 1.
- If the C input is used, set the `C_REG` parameter to 1.
- Set `M_SEL` to P.
- Set `N_SEL` to `CONST0`, `C`, `CASCIN`, or `W`.
- Set `W_SEL` to `ADD_SUB`.
- Set `CASCOUT_SEL` to `W` or `ABC`. If set to `ABC` the A, B, and C inputs are expected to be capturing the 33-bit internal FP32 representation expected by the adder.
- Set the `P_EXT` and `C_EXT` parameters to `ALIGN_RIGHT`.

The shifter is bypassed.

The O output is in FP32 mode:

- Bits [47:43] carry the error flags {INVALID, INFINITE, OVERFLOW, UNDERFLOW, INEXACT},.
- Bits [42:32] are unused.
- Bits [31:0] are the FP32 representation.

The `USE_CE` and `USE_RST` parameters are only valid for the A, B, C, OP, and O registers. They are ignored for the P and W registers.

Cascading Blocks

When the `CASCIN` or `CASCOUT` ports are used to chain DSP blocks the bit-slicing of the data matches the output port `O`. All DSP Blocks in a chain must be in the same mode.

DSP Block Primitives

The Efinity® software has three primitives to support the Titanium DSP Block:

- *EFX_DSP48*—Supports the full functionality of the DSP block in all modes (NORMAL, BFLOAT, DUAL, and QUAD)
- *EFX_DSP24*—Supports one 8 x 8 multiplier in DUAL mode.
- *EFX_DSP12*—Supports one 4 x 4 multiplier in QUAD mode.

Floating-Point Support

The Titanium DSP block supports the bfloat16 number format for the A, B, and C inputs and single-precision IEEE 754 32-bit floating point (FP32) number format for the CASCIN input and O and CASCOUT outputs.

Bfloat16 is a truncated, 16-bit version of the 32-bit IEEE Std 754 single-precision floating point format, and is often used in AI and machine learning applications. Because they have the same number of exponent bits, you can easily convert bfloat16 to to FP32.

Table 25: Bfloat16 Format

15	14						7	6						0
Sign			Exponent							Fraction				

Table 26: FP32 Format

31	30					23	22																							0
Sign			Exponent										Fraction																

EFX_DSP48

Full Function DSP Block

The EFX_DSP48 primitive models the full functionality of the DSP block in all modes (NORMAL, BFLOAT, DUAL, and QUAD). In DUAL or QUAD modes, the physical arrangement of the individual data paths must be considered as well as special handling of the MSB data bits.

Auto-Instantiation and Compilation

The DSP packing is done automatically by the Efinity software. Efinity packs 2 DSP24 blocks or at maximum of 4 DSP12 blocks under the same clock and reset signal into a single DSP48 resource. Mixing DSP24 and DSP12 packing is not allowed.

You can instantiate DSP48 in DUAL or QUAD mode for manual packing.

Manual DSP Instantiation

The manual DSP instantiation is carried out in the following modes:

Table 27: Dual Mode

Multiplier	Input	Output
11 x 10	A[18:8] B[17:8]	O[47:24] Output sign: Extended to 24-bits.
8 x 8	A[7:0] B[7:0]	O[23:0] Output sign: Extended to 24-bits.

Table 28: Quad Mode

Multiplier	Input	Output
7 x 6	A[18:12] B[17:12]	O[47:36] Output sign: Extended to 12-bits. ⚠ **Important:** The output is 12-bit only. A 13-bit output is truncated to 12-bit, hence the maximum range for calculation is limited.
4 x 4	A[11:8] B[11:8]	O[35:24] Output sign: Extended to 12-bits.
4 x 4	A[7:4] B[7:4]	O[23:12] Output sign: Extended to 12-bits.
4 x 4	A[3:0] B[3:0]	O[11:0] Output sign: Extended to 12-bits.

EFX_DSP48 Ports

Figure 25: EFX_DSP48 Symbol
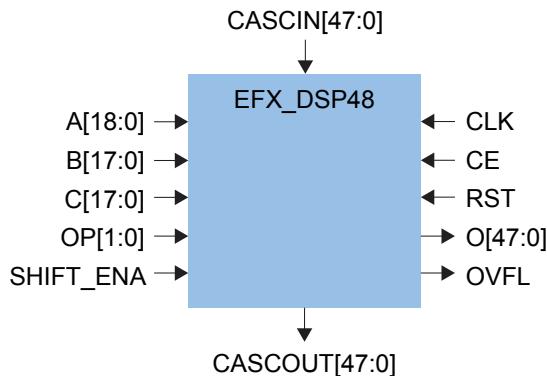
Table 29: EFX_DSP48 Ports

Port Name	Direction	Description
A[18:0]	Input	Operand A.
B[17:0]	Input	Operand B.
C[17:0]	Input	Operand C.
OP[1:0]	Input	Add/subtract function control. 00: M + N 01: M - N 10: -M + N 11: -M - N
SHIFT_ENA	Input	Load the shifter register S from the C input.
CASCIN[47:0]	Input	Dedicated input from the DSP block below.
CLK	Input	Clock.
CE	Input	Clock enable.
RST	Input	Set/reset.
O[47:0]	Output	DSP output.
CASCOUT[47:0]	Output	Dedicated output to the DSP block above.
OVFL	Output	Overflow/underflow flag. Signed—The operation behaves as you would expect. Unsigned—Internally, the operation is performed using signed arithmetic. When OP is 00, the operation behaves as you would expect. For the other operands, it reports overflow or underflow if information is lost during the operation. When using dual or quad mode, the overflow bits are calculated independently and ORed together.

Add/subtract behavior depends on the mode and sign:
- In BFLOAT mode, all 4 add/subtract functions are available.
- In an integer mode (NORMAL, DUAL, or QUAD) the 00, 01, and 10 functions operate as expected, but 11 performs -M-N-1.

EFX_DSP48 Parameters

Table 30: EFX_DSP48 Parameters

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

Parameter Name	Allowed Values	Description
MODE	NORMAL	One 19 x 18 integer multiply.
	DUAL	One 11 x 10 and one 8 x 8 integer multiply.
	QUAD	One 7 x 6 and three 4 x 4 integer multiplies.
	BFLOAT	One 16 x 16 BFLOAT16.
A_REG	0, 1	1: Enable A registers. 0: Disabled.
A_REG_USE_CE	0, 1	1: A register uses the CE pin. 0: Disabled.
A_REG_ USE_RST	0, 1	1: A register uses the RST pin. 0: Disabled.
B_REG	0, 1	1: Enable B registers. 0: Disabled.
B_REG_ USE_CE	0, 1	1: B register uses the CE pin. 0: Disabled.
B_REG_ USE_RST	0, 1	1: B register uses the RST pin. 0: Disabled.
C_REG	0, 1	1: Enable C registers. 0: Disabled.
C_REG_ USE_CE	0, 1	1: C register uses the CE pin. 0: Disabled.
C_REG_ USE_RST	0, 1	1: C register uses the RST pin. 0: Disabled.
OP_REG	0, 1	1: Enable OP register. 0: Disabled.
OP_REG_ USE_CE	0, 1	1: OP register uses the CE pin. 0: Disabled.
OP_REG_ USE_RST	0, 1	1: OP register uses the RST pin. 0: Disabled.
P_REG	0, 1	1: Enable P registers. 0: Disabled.
P_REG_ USE_CE	0, 1	1: P register uses the CE pin. 0: Disabled.
P_REG_ USE_RST	0, 1	1: P register uses the RST pin. 0: Disabled.
W_REG	0, 1	1: Enable W registers. 0: Disabled.

Parameter Name	Allowed Values	Description
W_REG_ USE_CE	0, 1	1: W register uses the CE pin. 0: Disabled.
W_REG_ USE_RST	0, 1	1: W register uses the RST pin. 0: Disabled.
O_REG	0, 1	1: Enable output registers. 0: Disabled.
O_REG_ USE_CE	0, 1	1: O register uses the CE pin. 0: Disabled.
O_REG_ USE_RST	0, 1	1: O register uses the RST pin. 0: Disabled.
SHIFTER	0, 1	1: Enables output shifting. When using output shifting: • If you bypass the adder, disable W_REG and enable O_REG • If you use the adder, enable both W_REG and O_REG 0: Disables output shifting (default).
RST_SYNC	0, 1	1: Synchronous. 0: Asynchronous.
SIGNED	0, 1	1: Signed arithmetic. 0: Unsigned. Ignored in BFLOAT mode.
P_EXT	ALIGN_LEFT, ALIGN_RIGHT	Extends the multiplier output to 48 bits. ALIGN_RIGHT: The MSB is sign-extended or zero-extended based on the MODE and SIGNED value. ALIGN_LEFT: The LSB is zero-extended.
C_EXT	ALIGN_LEFT, ALIGN_RIGHT	This extends the multiplier output to 48-bits. ALIGN_RIGHT: The MSB is sign-extended or zero-extended based on the MODE and SIGNED value. ALIGN_LEFT: The LSB is zero-extended.
M_SEL	P, C	Selects the data for the M input to the add/sub block. ABC: Lower 16 bits of the A, B, and C inputs. C: Input C. P: Multiplier output.
N_SEL	CONST0, CONST1, C, P, CASCIN, CASCIN_ASR18, CASCIN_LSR18, W, O	Selects the data for the N input to the add/sub block. CONST0/1: Chooses a constant 0 or 1. In DUAL and QUAD modes, the 1 is passed to each slice value; C: Input C. P: Multiplier output; CASCIN: CASCIN input; CASCIN_ASR18: CASCIN input, but does an arithmetic shift it right by 18 bits. Only legal in NORMAL mode. CASCIN_LSR18: CASCIN input, but does a logical shift it right by 18 bits. Only legal in NORMAL mode. W: Input to the logical shifter. If selected, enable the W register. O: DSP output. If selected, enable the O and W registers.

Parameter Name	Allowed Values	Description
W_SEL	P, X	Selects the data for the logical shifter. P: Multiplier output. X: Add/sub output. If you bypass the adder, disable W_REG.
CASCOUT_SEL	C, P, W, ABC	Selects the data for the CASCOUT output. ABC: Lower 16 bits of the A, B, and C inputs. C: C input. Illegal in BFLOAT mode. P: Multiplier output. Illegal in BFLOAT mode. W: Input to the logical shifter. When set to W, you cannot bypass the adder. CASCOUT_SEL is legal in NORMAL, DUAL, and QUAD modes. Allows you to pass in a full CASCIN result. With some external logic, allows a DSP CASCOUT to represent an FP32 number {a[15:0], b[15:0], c[15:0]}
ROUNDING[4]	RTZ	Round towards zero.
	RUP	Round up.
	RDOWN	Round down.
	RNI	Round nearest, ties to infinity.
	RTO	Round to odd.
	RNE	Round nearest, ties to even.
<name>_POLARITY	0, 1	1: Active high 0: Active low.

---

[4] ROUNDING is only used in BFLOAT mode.

EFX_DSP48 Function

The EFX_DSP48 is a signed integer multiplier.

Figure 26: EFX_DSP48 Verilog HDL Instantiation

```
// DSP48 Instantiation Template
EFX_DSP48 #(

    .MODE                ("NORMAL"),      // Normal mode
    .A_REG               (0),             // enable A-register
    .B_REG               (0),             // enable B-register
    .C_REG               (0),             // enable C-register
    .P_REG               (0),             // enable P-register
    .OP_REG              (0),             // enable OP-register
    .W_REG               (0),             // enable W-register
    .O_REG               (0),             // enable O-register
    .RST_SYNC            (0),             // set sync/async reset
    .SIGNED              (1),             // set signed/unsigned multiply
    .P_EXT               ("ALIGN_RIGHT"), // left/right alignment for P
    .C_EXT               ("ALIGN_RIGHT"), // left/right alighment for C
    .M_SEL               ("P"),           // select M-input to the adder
    .N_SEL               ("C"),           // select N-input to the adder
    .W_SEL               ("X"),           // select input to the shifter
    .CASCOUT_SEL         ("W"),           // select cascout
    .CLK_POLARITY        (1),             // clk polarity
    .CE_POLARITY         (1),             // ce polarity
    .RST_POLARITY        (1),             // rst polarity
    .SHIFT_ENA_POLARITY  (1),             // shift_ena polarity
    .ROUNDING            ("RNE"),         // rounding method
    .A_REG_USE_CE        (1),             // A-register use clock enable
    .B_REG_USE_CE        (1),             // B-register use clock enable
    .C_REG_USE_CE        (1),             // C-register use clock enable
    .OP_REG_USE_CE       (1),             // OP-register use clock enable
    .P_REG_USE_CE        (1),             // P-register use clock enable
    .W_REG_USE_CE        (1),             // W-register use clock enable
    .O_REG_USE_CE        (1),             // Oregister use clock enable
    .A_REG_USE_RST       (1),             // A-register use reset
    .B_REG_USE_RST       (1),             // B-register use reset
    .C_REG_USE_RST       (1),             // C-register use reset
    .OP_REG_USE_RST      (1),             // OP-register use reset
    .P_REG_USE_RST       (1),             // P-register use reset
    .W_REG_USE_RST       (1),             // W-register use reset
    .O_REG_USE_RST       (1)              // O-register use reset
    )
    dsp48_inst (
    .A (A),                              // 19-bit A input
    .B (B),                              // 18-bit B input
    .C (C),                              // 17-bit C input
    .CASCIN(0),                          // 48-bit cascin from another DSP block
    .OP (2'b00),                         // 2-bit operation mode
    .SHIFT_ENA(1'b1),                    // 1-bit shift_ena input
    .CLK (clk),                          // 1-bit clock
    .CE (ce),                            // 1-bit clock enable
    .RST (rst),                          // 1-bit reset
    .O (O),                              // 48-bit output
    .CASCOUT(),                          // 48-bit cascout, hard-wired to another DSP block
    .OVFL()                              // 1-bit overflow flag
);
```

Figure 27: EFX_DSP48 VHDL Instantiation

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

entity dsp48_VHDL is
    port
    (
        clk, ce, rst, shift_ena    : in std_logic;
        a                          : in std_logic_vector(18 downto 0);
        b,c                        : in std_logic_vector(17 downto 0);
        op                         : in std_logic_vector(1 downto 0);
        o                          : out std_logic_vector(47 downto 0);
        ovfl                       : out std_logic
    );
end entity dsp48_VHDL;

architecture Behavioral of dsp48_VHDL is
begin

 EFX_DSP48_inst : EFX_DSP48
    generic map (
        MODE => "NORMAL",
        A_REG => 0,
        B_REG => 0,
        C_REG => 0,
        P_REG => 0,
        OP_REG => 0,
        W_REG => 0,
        O_REG => 0,

        RST_SYNC => 0,
        SIGNED   => 1,

        P_EXT => "ALIGN_RIGHT",
        C_EXT => "ALIGN_RIGHT",
        M_SEL => "P",
        N_SEL => "C",
        W_SEL => "P",
        CASCOUT_SEL => "W",

        CLK_POLARITY  => 1,
        CE_POLARITY  => 1,
        RST_POLARITY  => 1,
        SHIFT_ENA_POLARITY  => 1,

        A_REG_USE_CE  => 1,
        B_REG_USE_CE  => 1,
        C_REG_USE_CE  => 1,
        OP_REG_USE_CE  => 1,
        P_REG_USE_CE  => 1,
        W_REG_USE_CE  => 1,
        O_REG_USE_CE  => 1,

        A_REG_USE_RST  => 1,
        B_REG_USE_RST  => 1,
        C_REG_USE_RST  => 1,
        OP_REG_USE_RST  => 1,
        P_REG_USE_RST  => 1,
        W_REG_USE_RST  => 1,
        O_REG_USE_RST  => 1,

        ROUNDING => "RNE"
    )
    port map (
        A => a,
        B => b,
        C => c,
        CASCIN => (others => '0'),
        OP => op,
        SHIFT_ENA => shift_ena,
        CLK => clk,
        CE => ce,
        RST => rst,
        O => o,
        CASCOUT => open,
        OVFL => ovfl
    );

end architecture Behavioral;
```

EFX_DSP24

Dual-Mode 8 x 8 DSP Block

The EFX_DSP24 primitive can model the functionality of one 8 x 8 multiplier in DUAL mode. The placer can place two EFX_DSP24 primitives in one DSP block. The placer is responsible for sign or zero extending the MSB bits of the EFX_DSP24 placed in the top location.

DSP24 Properties

The EFX_DSP24 primitive is automatically packed and placed into the DSP block in DUAL mode, and supports a maximum width of 8 x 8.

You must use the EFX_DSP48 primitive if you want to use the 11 x 10 format in the MSB lane in DUAL mode.

EFX_DSP24 Ports

Figure 28: EFX_DSP24 Symbol
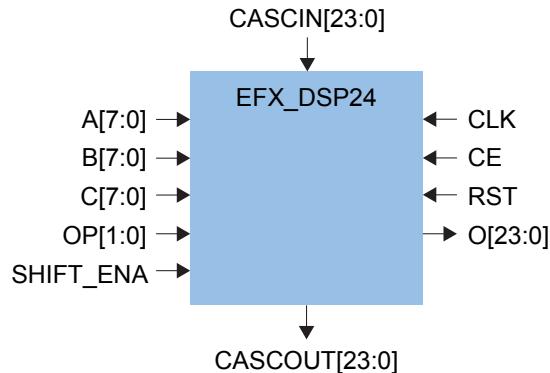
Table 31: EFX_DSP24 Ports

Port Name	Direction	Description
A[7:0]	Input	Operand A.
B[7:0]	Input	Operand B.
C[7:0]	Input	Operand C.
OP[1:0]	Input	Add/subtract function control 00: M + N 01: M - N 10: -M + N 11: -M - N
SHIFT_ENA	Input	Load the shifter register S from the C input.
CASCIN[23:0]	Input	Dedicated input from the DSP block below.
CLK	Input	Clock.
CE	Input	Clock enable.
RST	Input	Set/reset.
O[23:0]	Output	DSP output.
CASCOUT[23:0]	Output	Dedicated output to the DSP block above.

The add/subtract behavior for 00, 01, and 10 functions operate as expected, but 11 performs -M-N-1.

EFX_DSP24 Parameters

Table 32: EFX_DSP24 Parameters

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

Parameter Name	Allowed Values	Description
A_REG	0, 1	1: Enable A registers. 0: Disabled.
A_REG_USE_CE	0, 1	1: A register uses the CE pin. 0: Disabled.
A_REG_ USE_RST	0, 1	1: A register uses the RST pin. 0: Disabled.
B_REG	0, 1	1: Enable B registers. 0: Disabled.
B_REG_ USE_CE	0, 1	1: B register uses the CE pin. 0: Disabled.
B_REG_ USE_RST	0, 1	1: B register uses the RST pin. 0: Disabled.
C_REG	0, 1	1: Enable C registers. 0: Disabled.

Parameter Name	Allowed Values	Description
C_REG_ USE_CE	0, 1	1: C register uses the CE pin. 0: Disabled.
C_REG_ USE_RST	0, 1	1: C register uses the RST pin. 0: Disabled.
OP_REG	0, 1	1: Enable OP register. 0: Disabled.
OP_REG_ USE_CE	0, 1	1: OP register uses the CE pin. 0: Disabled.
OP_REG_ USE_RST	0, 1	1: OP register uses the RST pin. 0: Disabled.
P_REG	0, 1	1: Enable P registers. 0: Disabled.
P_REG_ USE_CE	0, 1	1: P register uses the CE pin. 0: Disabled.
P_REG_ USE_RST	0, 1	1: P register uses the RST pin. 0: Disabled.
W_REG	0, 1	1: Enable W registers. 0: Disabled.
W_REG_ USE_CE	0, 1	1: W register uses the CE pin. 0: Disabled.
W_REG_ USE_RST	0, 1	1: W register uses the RST pin. 0: Disabled.
O_REG	0, 1	1: Enable output registers. 0: Disabled.
O_REG_ USE_CE	0, 1	1: O register uses the CE pin. 0: Disabled.
O_REG_ USE_RST	0, 1	1: O register uses the RST pin. 0: Disabled.
SHIFTER	0, 1	1: Enables output shifting. When using output shifting: • If you bypass the adder, disable W_REG and enable O_REG • If you use the adder, enable both W_REG and O_REG 0: Disables output shifting (default).
RST_SYNC	0, 1	1: Synchronous. 0: Asynchronous.
SIGNED	0, 1	1: Signed arithmetic. 0: Unsigned.
P_EXT	ALIGN_LEFT, ALIGN_RIGHT	Extends the multiplier output to 24 bits. ALIGN_RIGHT: The MSB is sign-extended or zero-extended based on the MODE and SIGNED value. ALIGN_LEFT: The LSB is zero-extended.
C_EXT	ALIGN_LEFT, ALIGN_RIGHT	This extends the multiplier output to 24-bits. ALIGN_RIGHT: The MSB is sign-extended or zero-extended based on the MODE and SIGNED value. ALIGN_LEFT: The LSB is zero-extended.

Parameter Name	Allowed Values	Description
M_SEL	P, C	Selects the data for the M input to the add/sub block. C: Input C. P: Multiplier output.
N_SEL	CONST0, CONST1, C, P, CASCIN, W, or O	Selects the data for the N input to the add/sub block. CONST0/1: Chooses a constant 0 or 1. C: Input C. P: Multiplier output; CASCIN: CASCIN input; W: Input to the logical shifter. If selected, enable the W register. O: DSP output. If selected, enable the O and W registers.
W_SEL	P, X	Selects the data for the logical shifter. P: Multiplier output. X: Add/sub output. If you bypass the adder, disable W_REG.
CASCOUT_SEL	C, P, W, ABC	Selects the data for the CASCOUT output. ABC: Lower 8-bits of the A, B, and C inputs. C: C input. P: Multiplier output. W: Logical shifter input. When set to W, you cannot bypass the adder.
*<name>*_POLARITY	0, 1	0: Active low. 1: Active high.

EFX_DSP24 Function

The EFX_DSP24 is a signed integer multiplier.

Figure 29: EFX_DSP24 Verilog HDL Instantiation

```
// DSP24 Instantiation Template
EFX_DSP24 #(

    .A_REG                  (0),                // enable A-register
    .B_REG                  (0),                // enable B-register
    .C_REG                  (0),                // enable C-register
    .P_REG                  (0),                // enable P-register
    .OP_REG                 (0),                // enable OP-register
    .W_REG                  (0),                // enable W-register
    .O_REG                  (0),                // enable O-register
    .RST_SYNC               (0),                // set sync/async reset
    .SIGNED                 (1),                // set signed/unsigned multiply
    .P_EXT                  ("ALIGN_RIGHT"),    // left/right alighment for P
    .C_EXT                  ("ALIGN_RIGHT"),    // left/right alighment for C
    .M_SEL                  ("P"),              // select M-input to the adder
    .N_SEL                  ("C"),              // select N-input to the adder
    .W_SEL                  ("X"),              // select input to the shifter
    .CASCOUT_SEL            ("W"),              // select cascout
    .CLK_POLARITY           (1),                // clk polarity
    .CE_POLARITY            (1),                // ce polarity
    .RST_POLARITY           (1),                // rst polarity
    .SHIFT_ENA_POLARITY     (1),                // shift_ena polarity
    .A_REG_USE_CE           (1),                // A-register use clock enable
    .B_REG_USE_CE           (1),                // B-register use clock enable
    .C_REG_USE_CE           (1),                // C-register use clock enable
    .OP_REG_USE_CE          (1),                // OP-register use clock enable
    .P_REG_USE_CE           (1),                // P-register use clock enable
    .W_REG_USE_CE           (1),                // W-register use clock enable
    .O_REG_USE_CE           (1),                // Oregister use clock enable
    .A_REG_USE_RST          (1),                // A-register use reset
    .B_REG_USE_RST          (1),                // B-register use reset
    .C_REG_USE_RST          (1),                // C-register use reset
    .OP_REG_USE_RST         (1),                // OP-register use reset
    .P_REG_USE_RST          (1),                // P-register use reset
    .W_REG_USE_RST          (1),                // W-register use reset
    .O_REG_USE_RST          (1)                 // O-register use reset
    )
    dsp24_inst (
    .A (A),                                     // 8-bit A input
    .B (B),                                     // 8-bit B input
    .C (C),                                     // 8-bit C input
    .CASCIN(0),                                 // 24-bit cascin from another DSP block
    .OP (2'b00),                                // 2-bit operation mode
    .SHIFT_ENA(1'b1),                           // 1-bit shift_ena input
    .CLK (clk),                                 // 1-bit clock
    .CE (ce),                                   // 1-bit clock enable
    .RST (rst),                                 // 1-bit reset
    .O (O),                                     // 24-bit output
    .CASCOUT(),                                 // 24-bit cascout, hard-wired to another DSP block
);
```

Figure 30: EFX_DSP24 VHDL Instantiation

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

entity dsp24_VHDL is
    port
    (
        clk, ce, rst, shift_ena    : in std_logic;
        a,b,c                      : in std_logic_vector(7 downto 0);
        op                         : in std_logic_vector(1 downto 0);
        o                          : out std_logic_vector(23 downto 0)
    );
end entity dsp24_VHDL;

architecture Behavioral of dsp24_VHDL is
begin

 EFX_DSP24_inst : EFX_DSP24
    generic map (
        A_REG => 0,
        B_REG => 0,
        C_REG => 0,
        P_REG => 0,
        OP_REG => 0,
        W_REG => 0,
        O_REG => 0,

        RST_SYNC => 0,
        SIGNED   => 1,

        P_EXT => "ALIGN_RIGHT",
        C_EXT => "ALIGN_RIGHT",
        M_SEL => "P",
        N_SEL => "C",
        W_SEL => "P",
        CASCOUT_SEL => "W",

        CLK_POLARITY  => 1,
        CE_POLARITY   => 1,
        RST_POLARITY  => 1,
        SHIFT_ENA_POLARITY  => 1,

        A_REG_USE_CE  => 1,
        B_REG_USE_CE  => 1,
        C_REG_USE_CE  => 1,
        OP_REG_USE_CE  => 1,
        P_REG_USE_CE  => 1,
        W_REG_USE_CE  => 1,
        O_REG_USE_CE  => 1,

        A_REG_USE_RST  => 1,
        B_REG_USE_RST  => 1,
        C_REG_USE_RST  => 1,
        OP_REG_USE_RST  => 1,
        P_REG_USE_RST  => 1,
        W_REG_USE_RST  => 1,
        O_REG_USE_RST  => 1
    )
    port map (
        A => a,
        B => b,
        C => c,
        CASCIN => (others => '0'),
        OP => op,
        SHIFT_ENA => shift_ena,
        CLK => clk,
        CE => ce,
        RST => rst,
        O => o,
        CASCOUT => open

    );end architecture Behavioral;
```

EFX_DSP12

Quad-Mode 4 x 4 DSP Block

The EFX_DSP12 primitive models the functionality of one 4 x 4 multiplier in QUAD mode. The placer can place four EFX_DSP12 primitives in one DSP block. The placer is responsible for sign or zero extending the MSB bits of the EFX_DSP12 placed in the top location.

DSP12 Properties

The EFX_DSP12 primitive is automatically packed and placed into the DSP block in QUAD mode, and supports a maximum width of 4 x 4.

You must use the EFX_DSP48 primitive if you want to use the 7 x 6 format in the MSB lane in QUAD mode.

EFX_DSP12 Ports

Figure 31: EFX_DSP12 Symbol
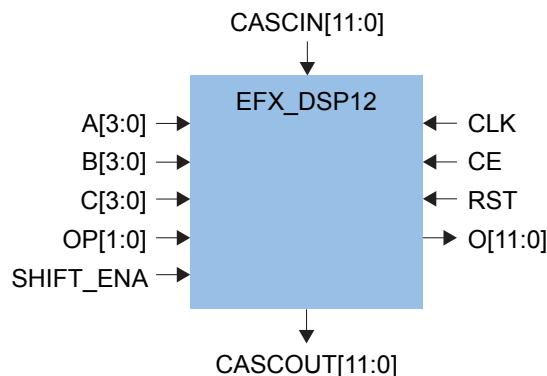
Table 33: EFX_DSP12 Ports

Port Name	Direction	Description
A[3:0]	Input	Operand A.
B[3:0]	Input	Operand B.
C[3:0]	Input	Operand C.
OP[1:0]	Input	Add/subtract function control. 00: M + N 01: M - N 10: -M + N 11: -M - N
SHIFT_ENA	Input	Load the shifter register S from the C input.
CASCIN[11:0]	Input	Dedicated input from the DSP block below.
CLK	Input	Clock.
CE	Input	Clock enable.
RST	Input	Set/reset.
O[11:0]	Output	DSP output.
CASCOUT[11:0]	Output	Dedicated output to the DSP block above.

The add/subtract behavior for 00, 01, and 10 functions operate as expected, but 11 performs -M-N-1.

EFX_DSP12 Parameters

Table 34: EFX_DSP12 Parameters

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

Parameter Name	Allowed Values	Description
A_REG	0, 1	1: Enable A registers. 0: Disabled.
A_REG_USE_CE	0, 1	1: A register uses the CE pin. 0: Disabled.
A_REG_ USE_RST	0, 1	1: A register uses the RST pin. 0: Disabled.
B_REG	0, 1	1: Enable B registers. 0: Disabled.
B_REG_ USE_CE	0, 1	1: B register uses the CE pin. 0: Disabled.
B_REG_ USE_RST	0, 1	1: B register uses the RST pin. 0: Disabled.
C_REG	0, 1	1: Enable C registers. 0: Disabled.

Parameter Name	Allowed Values	Description
C_REG_ USE_CE	0, 1	1: C register uses the CE pin. 0: Disabled.
C_REG_ USE_RST	0, 1	1: C register uses the RST pin. 0: Disabled.
OP_REG	0, 1	1: Enable OP register. 0: Disabled.
OP_REG_ USE_CE	0, 1	1: OP register uses the CE pin. 0: Disabled.
OP_REG_ USE_RST	0, 1	1: OP register uses the RST pin. 0: Disabled.
P_REG	0, 1	1: Enable P registers. 0: Disabled.
P_REG_ USE_CE	0, 1	1: P register uses the CE pin. 0: Disabled.
P_REG_ USE_RST	0, 1	1: P register uses the RST pin. 0: Disabled.
W_REG	0, 1	1: Enable W registers. 0: Disabled.
W_REG_ USE_CE	0, 1	1: W register uses the CE pin. 0: Disabled.
W_REG_ USE_RST	0, 1	1: W register uses the RST pin. 0: Disabled.
O_REG	0, 1	1: Enable output registers. 0: Disabled.
O_REG_ USE_CE	0, 1	1: O register uses the CE pin. 0: Disabled.
O_REG_ USE_RST	0, 1	1: O register uses the RST pin. 0: Disabled.
SHIFTER	0, 1	1: Enables output shifting. When using output shifting: • If you bypass the adder, disable W_REG and enable O_REG • If you use the adder, enable both W_REG and O_REG 0: Disables output shifting (default).
RST_SYNC	0, 1	1: Synchronous. 0: Asynchronous.
SIGNED	0, 1	1: Signed arithmetic. 0: Unsigned.
P_EXT	ALIGN_LEFT, ALIGN_RIGHT	Extends the multiplier output to 24 bits. ALIGN_RIGHT: The MSB is sign-extended or zero-extended based on the MODE and SIGNED value. ALIGN_LEFT: The LSB is zero-extended.
C_EXT	ALIGN_LEFT, ALIGN_RIGHT	This extends the multiplier output to 24-bits. ALIGN_RIGHT: The MSB is sign-extended or zero-extended based on the MODE and SIGNED value. ALIGN_LEFT: The LSB is zero-extended.

Parameter Name	Allowed Values	Description
M_SEL	P, C	Selects the data for the M input to the add/sub block. C: Input C. P: Multiplier output.
N_SEL	CONST0, CONST1, C, P, CASCIN, W, or O	Selects the data for the N input to the add/sub block. CONST0/1: Chooses a constant 0 or 1. C: Input C. P: Multiplier output; CASCIN: CASCIN input; W: Input to the logical shifter. If selected, enable the W register. O: DSP output. If selected, enable the O and W registers.
W_SEL	P, X	Selects the data for the logical shifter. P: Multiplier output. X: Add/sub output. If you bypass the adder, disable W_REG.
CASCOUT_SEL	C, P, W, ABC	Selects the data for the CASCOUT output. ABC: Lower 8-bits of the A, B, and C inputs. C: C input. P: Multiplier output. W: Logical shifter input. When set to W, you cannot bypass the adder.
*<name>*_POLARITY	0, 1	0: Active low. 1: Active high.

EFX_DSP12 Function

The EFX_DSP12 is a signed integer multiplier.

Figure 32: EFX_DSP12 Verilog HDL Instantiation

```
// DSP12 Instantiation Template
EFX_DSP12 #(

    .A_REG               (0),              // enable A-register
    .B_REG               (0),              // enable B-register
    .C_REG               (0),              // enable C-register
    .P_REG               (0),              // enable P-register
    .OP_REG              (0),              // enable OP-register
    .W_REG               (0),              // enable W-register
    .O_REG               (0),              // enable O-register
    .RST_SYNC            (0),              // set sync/async reset
    .SIGNED              (1),              // set signed/unsigned multiply
    .P_EXT               ("ALIGN_RIGHT"),  // left/right alighment for P
    .C_EXT               ("ALIGN_RIGHT"),  // left/right alighment for C
    .M_SEL               ("P"),            // select M-input to the adder
    .N_SEL               ("C"),            // select N-input to the adder
    .W_SEL               ("X"),            // select input to the shifter
    .CASCOUT_SEL         ("W"),            // select cascout
    .CLK_POLARITY        (1),              // clk polarity
    .CE_POLARITY         (1),              // ce polarity
    .RST_POLARITY        (1),              // rst polarity
    .SHIFT_ENA_POLARITY  (1),              // shift_ena polarity
    .A_REG_USE_CE        (1),              // A-register use clock enable
    .B_REG_USE_CE        (1),              // B-register use clock enable
    .C_REG_USE_CE        (1),              // C-register use clock enable
    .OP_REG_USE_CE       (1),              // OP-register use clock enable
    .P_REG_USE_CE        (1),              // P-register use clock enable
    .W_REG_USE_CE        (1),              // W-register use clock enable
    .O_REG_USE_CE        (1),              // Oregister use clock enable
    .A_REG_USE_RST       (1),              // A-register use reset
    .B_REG_USE_RST       (1),              // B-register use reset
    .C_REG_USE_RST       (1),              // C-register use reset
    .OP_REG_USE_RST      (1),              // OP-register use reset
    .P_REG_USE_RST       (1),              // P-register use reset
    .W_REG_USE_RST       (1),              // W-register use reset
    .O_REG_USE_RST       (1)               // O-register use reset
    )
    dsp12_inst (
    .A (A),                               // 4-bit A input
    .B (B),                               // 4-bit B input
    .C (C),                               // 4-bit C input
    .CASCIN(0),                           // 12-bit cascin, hard-wired from another DSP block
    .OP (2'b00),                          // 2-bit operation mode
    .SHIFT_ENA(1'b1),                     // 1-bit shift_ena input
    .CLK (clk),                           // 1-bit clock
    .CE (ce),                             // 1-bit clock enable
    .RST (rst),                           // 1-bit reset
    .O (O),                               // 12-bit output
    .CASCOUT(),                           // 12-bit cascout, hard-wired to another DSP block);
```

Figure 33: EFX_DSP12 VHDL Instantiation

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

entity dsp12_VHDL is
    port
    (
        clk, ce, rst, shift_ena    : in std_logic;
        a,b,c                      : in std_logic_vector(3 downto 0);
        op                         : in std_logic_vector(1 downto 0);
        o                          : out std_logic_vector(11 downto 0)
    );
end entity dsp12_VHDL;

architecture Behavioral of dsp12_VHDL is
begin

 EFX_DSP12_inst : EFX_DSP12
    generic map (
        A_REG => 0,
        B_REG => 0,
        C_REG => 0,
        P_REG => 0,
        OP_REG => 0,
        W_REG => 0,
        O_REG => 0,

        RST_SYNC => 0,
        SIGNED   => 1,

        P_EXT => "ALIGN_RIGHT",
        C_EXT => "ALIGN_RIGHT",
        M_SEL => "P",
        N_SEL => "C",
        W_SEL => "P",
        CASCOUT_SEL => "W",

        CLK_POLARITY  => 1,
        CE_POLARITY   => 1,
        RST_POLARITY  => 1,
        SHIFT_ENA_POLARITY  => 1,

        A_REG_USE_CE  => 1,
        B_REG_USE_CE  => 1,
        C_REG_USE_CE  => 1,
        OP_REG_USE_CE  => 1,
        P_REG_USE_CE  => 1,
        W_REG_USE_CE  => 1,
        O_REG_USE_CE  => 1,

        A_REG_USE_RST  => 1,
        B_REG_USE_RST  => 1,
        C_REG_USE_RST  => 1,
        OP_REG_USE_RST  => 1,
        P_REG_USE_RST  => 1,
        W_REG_USE_RST  => 1,
        O_REG_USE_RST  => 1
    )
    port map (
        A => a,
        B => b,
        C => c,
        CASCIN => (others => '0'),
        OP => op,
        SHIFT_ENA => shift_ena,
        CLK => clk,
        CE => ce,
        RST => rst,
        O => o,
        CASCOUT => open
    );

end architecture Behavioral;
```

EFX_GBUFCE

Global Clock Buffer

The EFX_GBUFCE logic block represents the global clock buffer driving the global or regional 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.

You can use an EFX_GBUFCE to gate a clock with the clock enable pin.

EFX_GBUFCE Ports

Figure 34: EFX_GBUFCE Symbol
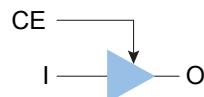
Table 35: EFX_GBUFCE Ports

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

EFX_GBUFCE Parameters

Table 36: 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 37: EFX_GBUFCE Function

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

Figure 35: 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 36: 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 37: Interface Primitives
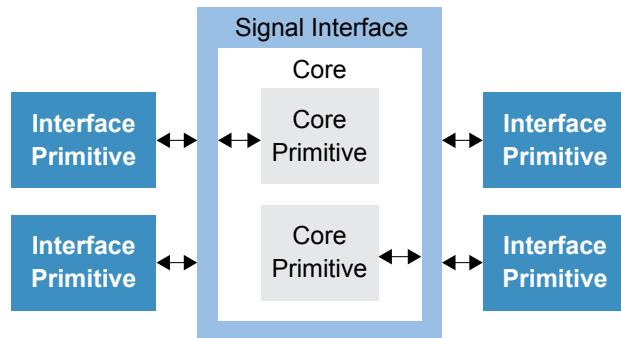
> **Important:** The Efinity software v2024.2 only supports a subset of the possible interface primitives. Additional primitives will be supported in upcoming releases.

Table 38: 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_V3, EFX_PLL_V3, EFX_FPLL_V1, OSC_V3

EFX_IBUF

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

EFX_IBUF Ports

Figure 38: EFX_IBUF Symbol
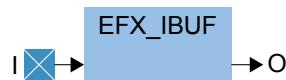
Table 39: EFX_ Ports

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

EFX_IBUF Parameters

Table 40: 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 39: EFX_IBUF Verilog HDL Instantiation

```
EFX_IBUF # (
    .PULL_OPTION("NONE")  // "NONE" (Default), "WEAK_PULLUP" , "WEAK_PULLDOWN"
) EFX_IBUF_inst (
    .I(I), 7/ Buffer input (connect to top-level port)
    .O(O)  // Buffer output
);
```

Figure 40: 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 68 primitive and do not enable the optional input signal.

EFX_OBUF Ports

Figure 41: EFX_OBUF Symbol
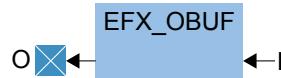
Table 41: EFX_OBUF Ports

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 42: EFX_OBUF Verilog HDL Instantiation

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

Figure 43: 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 44: EFX_IO_BUF Symbol
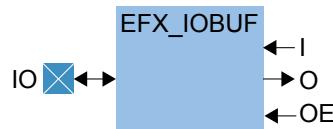
Table 42: EFX_IO_BUF Ports

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 43: 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 45: 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 46: 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 47: EFX_CLKOUT Symbol
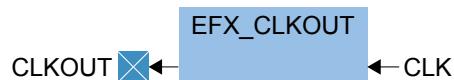
Table 44: EFX_CLKOUT Ports

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

EFX_CLKOUT Parameters

Table 45: 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 48: EFX_CLKOUT Verilog HDL Instantiation

```
EFX_CLKOUT# (
    .IS_CLK_INVERTED(1'b0')   // 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 49: 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 50: EFX_IREG Symbol
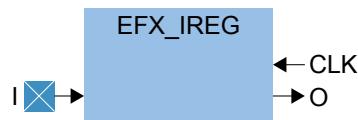
Table 46: EFX_IREG Ports

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 47: 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 51: 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 52: 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 53: EFX_OREG Symbol
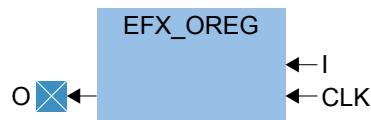
Table 48: EFX_OREG Ports

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 49: 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 54: 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 55: 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 56: EFX_IOREG Symbol
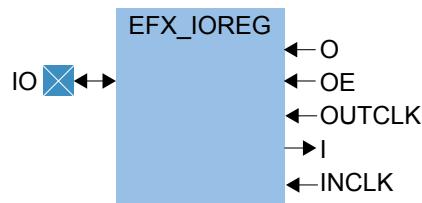
*Table 50: EFX_IOREG Ports*

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 51: 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 57: 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 58: 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 59: EFX_IDDIO Symbol
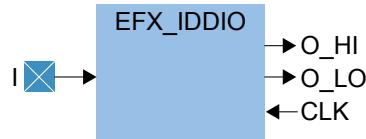
Table 52: EFX_IDDIO Ports

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 53: 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 DDIO_PIPE	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. DDIO_PIPE: Pipeline mode.

EFX_IDDIO Function

These examples show how to instantiate the input DDIO register.

Figure 60: 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", "DDIO_PIPE"
) 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 61: 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 62: EFX_ODDIO Symbol
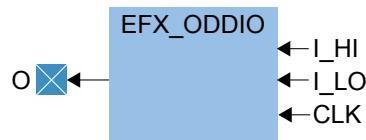
Table 54: EFX_ODDIO Ports

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 55: 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 63: 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 64: 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 65: EFX_JTAG_CTRL Symbol
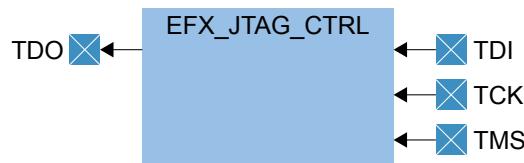
Table 56: EFX_JTAG_CTRL Ports

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 57: EFX_JTAG_CTRL Parameters

Parameter	Allowed Values	Description
DEVICE	Ti60W64, Ti60F100, Ti60F100S3F2, Ti60F225, Ti60F256, Ti35F100, Ti35F100S3F2, Ti35F225, Ti35F256, Ti180M484, Ti180L484, Ti120L484, Ti90L484, Ti180J484, Ti120J484, Ti90J484, Ti180J361, Ti120J361, Ti90J361, Ti180G529, Ti120G529, Ti90G529, Ti180G400, Ti120G400, Ti90G400, Ti375N1156, Ti240N1156, Ti165N1156, Ti375C529, Ti240C529, Ti165C529, Ti375N484, Ti240N484, Ti165N484, Ti135N676, Ti85N676	The FPGA and package.

EFX_JTAG_CTRL Function

Use these examples to instantiate the JTAG pins.

Figure 66: EFX_JTAG_CTRL Verilog HDL Instantiation

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

Figure 67: 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 68: EFX_JTAG_V1 Symbol
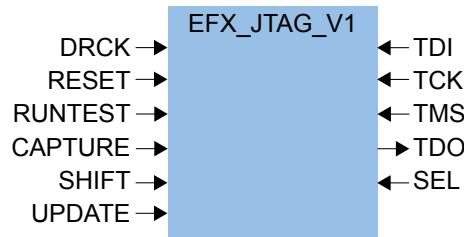
Table 58: EFX_JTAG_V1

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 59: EFX_JTAG_V1 Parameters

Parameter	Allowed Values	Description
RESOURCE	JTAG_USER1, JTAG_USER2, JTAG_USER3, JTAG_USER4	The JTAG user TAP interface name.

EFX_JTAG_V1 Function

Use these examples to instantiate the JTAG pins.

Figure 69: 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 70: 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_V3

HVIO and HSIO Used as GPIO

This primitive represents the functionality of the HVIO and HSIO when used as GPIO. The HVIO support a subset of the ports and parameters described in this primitive.

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

EFX_GPIO_V3 Ports

Figure 71: EFX_GPIO_V3 Symbol
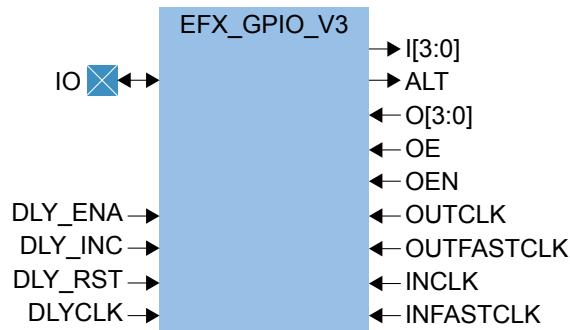
Table 60: EFX_GPIO_V3 Ports

Port	Direction	Description
I[3:0]	Output	Input data from the 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). When using the deserializer, the first bit is on I[0] and the last bit is on I[3].
ALT	Output	Alternative input connection for GCLK, PLL_CLKIN, RCLK, PLL_EXTFB, and VREF. (In the Interface Designer, **Register Option** is **none**).
O[3:0]	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). When using the serializer, the first bit is on O[0] and the last bit is on O[3].
OE/OEN	Input	Output enable from core fabric to the I/O block. Can be registered. OEN is used in differential mode. Drive it with the same signal as OE.

Port	Direction	Description
DLYCLK	Input	Delay clock for dynamic delay, sampled on the negative edge. In serializer mode, this clock must be the same clock as INCLK.
DLY_ENA	Input	(Optional) Enable the dynamic delay control.
DLY_INC	Input	(Optional) Dynamic delay control. When DLY_ENA = 1, 1: Increments 0: Decrements The updated delay count takes effect approximately 5 ns after the rising edge of the clock.
DLY_RST	Input	(Optional) Reset the delay counter.
OUTCLK	Input	Core clock that controls the output and OE registers. This clock is not visible in the user netlist.
OUTFASTCLK	Input	Core clock that controls the output serializer.
INCLK	Input	Core clock that controls the input registers. This clock is not visible in the user netlist.
INFASTCLK	Input	Core clock that controls the input serializer.

Table 61: EFX_GPIO_V3 Pads

Signal	Direction	Description
IO (P and N)	Bidirectional	GPIO pad.

EFX_GPIO_V3 Parameters

Table 62: EFX_GPIO_V3 Parameters

Parameter	Allowed Values	Description
MODE	INPUT, OUTPUT, INOUT, CLKOUT	The GPIO direction or clock mode.
OUT_REG	BYPASS, REG, DDIO, DDIO_RESYNC, SERIAL	Bypass (default), register, inverted register, or use DDIO for the output.
IN_REG	BYPASS, REG, DDIO, DDIO_RESYNC, DDIO_PIPE, SERIAL	Bypass (default), register, or use DDIO for 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 input clock is inverted. 0: Not inverted (default) 1: Inverted

Parameter	Allowed Values	Description
CONNECTION_TYPE	NONE, NORMAL, GCLK, GCTRL, PLL_CLKIN, PLL_EXTFB, MIPI_CLKIN, VREF	The alternate connection type. Used with the ALT port. NONE: Default
SCHMITT_TRIGGER	0, 1	Specifies whether to use the Schmitt trigger. 0: Do not use (default) 1: Use
DRIVE_STRENGTH	2, 4, 6, 8, 10, 12, 16	Indicates drive strength level. 2: Default
SLEW_RATE	0, 1	Specifies whether to turn on slew rate. 0: Turn off (default) 1: Turn on
BUS_HOLD	0, 1	Specifies whether to turn on bus hold. 0: Turn off (default) 1: Turn on
INDELAY	0 - 63	Static input delay setting. Single ended: 0 - 15 only. Approximately 60 ps of delay per step. Differential: 0 - 63. Approximately 25 ps of delay per step.
OUTDELAY	0 - 15	Output delay setting. Each step adds approximately 60 ps of delay.
INDELAY_DYN_MODE	0, 1	Specifies whether to turn on dynamic delay. 0: Turn off (default) 1: Turn on
IO_STANDARD	1.2_V_Differential_HSTL 1.2_V_Differential_SSTL 1.2_V_HSTL 1.2_V_LVCMOS 1.2_V_SSTL 1.5_V_Differential_HSTL 1.5_V_Differential_SSTL 1.5_V_HSTL 1.5_V_LVCMOS 1.5_V_SSTL 1.8_V_Differential_HSTL 1.8_V_Differential_SSTL 1.8_V_HSTL 1.8_V_LVCMOS 1.8_V_SSTL 2.5_V_LVCMOS 3.0_V_LVCMOS 3.0_V_LVTTL 3.3_V_LVCMOS 3.3_V_LVTTL	Indicates the I/O standard for the GPIO.

EFX_GPIO_V3 Function

These examples show how to instantiate the GPIO V3 primitive.

Figure 72: EFX_GPIO_V3 Verilog HDL Instantiation

```
EFX_GPIO_V3 # (
    .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"),
    .SCHMITT_TRIGGER(0),
    .DRIVE_STRENGTH(4),
    .SLEW_RATE(0),
    .BUS_HOLD(0),
    .INDELAY(0),
    .OUTDELAY(0),
    .INDELAY_DYN_MODE(0),
    .IO_STANDARD("1.8_V_LVCMOS")
) EFX_GPIO_V3_inst (
    .I(I),
    .ALT(ALT),
    .O(O),
    .OE(OE),
    .OEN(OEN),
    .PULL_UP_ENA(PULL_UP_ENA),
    .DLY_ENA(DLY_ENA),
    .DLY_INC(DLY_INC),
    .DLY_RST(DLY_RST),
    .OUTCLK(OUTCLK),
    .INCLK(INCLK),
    .DLYCLK(DLYCLK),
    .OUTFASTCLK(OUTFASTCLK),
    .INFASTCLK(INFASTCLK),
    .IO(IO)                        // Connect to top-level port
);
```

Figure 73: EFX_GPIO_V3 VHDL Instantiation

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

entity EFX_GPIO_V3_VHDL is
port ( i : in std_logic (3 downto 0);
       alt : out std_logic;
       o : out std_logic (3 downto 0);
       oe : in std_logic;
       oen : in std_logic;
       pull_up_ena : in std_logic;
       dly_ena : in std_logic;
       dly_inc : in std_logic;
       dly_rst : in std_logic;
       outclk : in std_logic;
       inclk : in std_logic;
       dlyclk : in std_logic;
       outfastclk : in std_logic;
       infastclk : in std_logic;
       io : inout std_logic;
end entity EFX_GPIO_V3_VHDL;

architecture Behavioral of EFX_GPIO_V3_VHDL is
 begin
 EFX_GPIO_V3_inst : EFX_GPIO_V3
    generic map (
       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",
```

```
        SCHMITT_TRIGGER => 0,
        DRIVE_STRENGTH => 4,
        SLEW_RATE => 0,
        BUS_HOLD => 0,
        INDELAY => 0,
        OUTDELAY => 0,
        INDELAY_DYN_MODE => 0,
        IO_STANDARD => "1.8_V_LVCMOS"
    )
    port map (
        I => o,
        ALT => alt,
        O => i,
        OE => oe,
        OEN => oen,
        PULL_UP_ENA => pull_up_ena,
        DLY_ENA => dly_ena;
        DLY_INC => dly_inc;
        DLY_RST => dly_rst
        OUTCLK => outclk,
        INCLK => inclk,
        .DLYCLK => dlyclk,
        .OUTFASTCLK => outfastclk,
        .INFASTCLK => infastclk,
        IO => io
    );
end architecture Behavioral;
```

EFX_PLL_V3

Full-Featured PLL

This PLL is available in Ti35, Ti60, Ti90, Ti120, and Ti180 FPGAs.

📖 **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_V3 Ports

Figure 74: EFX_PLL_V3 Symbol
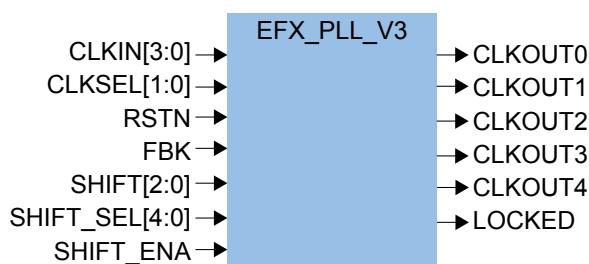
Table 63: EFX_PLL_V3 Ports

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 is in when the PLL is not in internal feedback mode.
CLKOUT0 CLKOUT1 CLKOUT2 CLKOUT3 CLKOUT4	Output	PLL output. You can route these signals as input clocks to the core's GCLK network. **:** CLKOUT4 can only feed the top or bottom regional clocks. **:** All PLL outputs lock on the negative clock edge. The Interface Designer inverts the clock polarity on the leaf cells by default (Invert Output Clock option unchecked). Check the option if you are using the clock to drive core logic. You can use CLKOUT0 only for clocks with a maximum frequency of 4x (integer) of the reference clock. If all your system clocks do not fall within this range, you should dedicate one unused clock for CLKOUT0.
LOCKED	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 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.

Signal	Direction	Description
SHIFT[2:0]	Input	(Optional) Dynamically change the phase shift of the output selected to the value set with this signal. Possible values from 000 (no phase shift) to 111 (3.5 F_{PLL} cycle delay). Each increment adds 0.5 cycle delay.
SHIFT_SEL[4:0]	Input	(Optional) Choose the output(s) affected by the dynamic phase shift.
SHIFT_ENA	Input	(Optional) When high, changes the phase shift of the selected PLL(s) to the new value.

EFX_PLL_V3 Parameters

Table 64: EFX_PLL_V3 Parameters

Parameter	Allowed Values	Description
N	1, 2, 4	Pre-divider.
M	1, 2, 4	Multiplier.
O	1, 2, 4, 8, 16, 32, 64, 128	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.
CLKOUT3_DIV	1 - 128	Output divider for CLKOUT3.
CLKOUT4_DIV	1 - 128	Output divider for CLKOUT4.
CLKOUT0_PHASE_STEP	0 - 7	Output phase for CLKOUT0.
CLKOUT1_PHASE_STEP	0 - 7	Output phase for CLKOUT1.
CLKOUT2_PHASE_STEP	0 - 7	Output phase for CLKOUT2.
CLKOUT3_PHASE_STEP	0 - 7	Output phase for CLKOUT3.
CLKOUT4_PHASE_STEP	0 - 7	Output phase for CLKOUT4.
FEEDBACK_CLK	CLK0, CLK1, CLK2, CLK3, CLK4	Indicates which clock is the feedback clock.
FEEDBACK_MODE	LOCAL, CORE, EXTERNAL	Indicates the feedback mode.
REFCLK_FREQ	16 - 800	Reference clock frequency.
IS_CLKOUT0_INVERTED	0, 1	Invert output clock for CLKOUT0.
IS_CLKOUT1_INVERTED	0, 1	Invert output clock for CLKOUT1.
IS_CLKOUT2_INVERTED	0, 1	Invert output clock for CLKOUT2.
IS_CLKOUT3_INVERTED	0, 1	Invert output clock for CLKOUT3.
IS_CLKOUT4_INVERTED	0, 1	Invert output clock for CLKOUT4.
CLKOUT3_CONN_TYPE	GCLK, RCLK	Specifies local or regional connection for CLKOUT3. (Not applicable to Ti35 or Ti60)
CLKOUT4_CONN_TYPE	GCLK, RCLK	Specifies local or regional connection for CLKOUT4. (Not applicable to Ti35 or Ti60)
CLKOUT0_DYNPHASE_EN	0, 1	Enable dynamic phase shift control for CLKOUT0.
CLKOUT1_DYNPHASE_EN	0, 1	Enable dynamic phase shift control for CLKOUT1.

Parameter	Allowed Values	Description
CLKOUT2_DYNPHASE_EN	0, 1	Enable dynamic phase shift control for CLKOUT2.
CLKOUT3_DYNPHASE_EN	0, 1	Enable dynamic phase shift control for CLKOUT3.
CLKOUT4_DYNPHASE_EN	0, 1	Enable dynamic phase shift control for CLKOUT4.

EFX_PLL_V3 Function

These examples show how to instantiate the PLL V3 primitive.

Figure 75: EFX_PLL_V3 Verilog HDL Instantiation

```
EFX_PLL_V3 # (
    .M(1),
    .N(1),
    .O(1),
    .CLKOUT0_DIV(1),
    .CLKOUT1_DIV(1),
    .CLKOUT2_DIV(1),
    .CLKOUT3_DIV(1),
    .CLKOUT4_DIV(1),
    .CLKOUT0_PHASE(0),
    .CLKOUT1_PHASE(0),
    .CLKOUT2_PHASE(0),
    .CLKOUT3_PHASE(0),
    .CLKOUT4_PHASE(0),
    .FEEDBACK_CLK("CLK0"),
    .FEEDBACK_MODE ("LOCAL"),
    .REFCLK_FREQ(25.0),
    .IS_CLKOUT0_INVERTED(0),
    .IS_CLKOUT1_INVERTED(0),
    .IS_CLKOUT2_INVERTED(0),
    .IS_CLKOUT3_INVERTED(0),
    .IS_CLKOUT4_INVERTED(0),
    .CLKOUT3_CONN_TYPE("GCLK"),
    .CLKOUT4_CONN_TYPE("GCLK"),
    .CLKOUT0_DYNPHASE_EN(0),
    .CLKOUT1_DYNPHASE_EN(0),
    .CLKOUT2_DYNPHASE_EN(0),
    .CLKOUT3_DYNPHASE_EN(0),
    .CLKOUT4_DYNPHASE_EN(0)
) EFX_PLL_V3_inst (
    .CLKOUT0(CLKOUT0),
    .CLKOUT1(CLKOUT1),
    .CLKOUT2(CLKOUT2),
    .CLKOUT3(CLKOUT3),
    .CLKOUT4(CLKOUT4),
    .LOCKED(LOCKED),
    .RSTN(RSTN),
    .FBK(FBK),
    .SHIFT_ENA(SHIFT_ENA),
    .CLKIN(CLKIN),
    .CLKSEL(CLKSEL),
    .SHIFT(SHIFT),
    .SHIFT_SEL(SHIFT_SEL)
);
```

Figure 76: EFX_PLL_V3 VHDL Instantiation

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

entity EFX_PLL_V3_VHDL is
port ( clkout0 : out std_logic;
       clkout1 : out std_logic;
       clkout2 : out std_logic;
       clkout3 : out std_logic;
       clkout4 : out std_logic;
       locked  : out std_logic;
       fbk : in std_logic;
       shift_ena : in std_logic;
       rstn : in std_logic;
       clkin : in std_logic(3 downto 0);
       clksel : in std_logic (1 downto 0);
       shift : in std_logic (2 downto 0);
       shift_sel : in std_logic (4 ddownto 0));
end entity EFX_PLL_V3_VHDL;

architecture Behavioral of EFX_PLL_V3_VHDL is
 begin
 EFX_PLL_V3_inst : EFX_PLL_V3
    generic map (
       M => 1,
       N => 1,
       O => 1,
       CLKOUT0_DIV => 1,
       CLKOUT1_DIV => 1,
       CLKOUT2_DIV => 1,
       CLKOUT3_DIV => 1,
       CLKOUT4_DIV => 1,
       CLKOUT0_PHASE => 0,
       CLKOUT1_PHASE => 0,
       CLKOUT2_PHASE => 0,
       CLKOUT3_PHASE => 0,
       CLKOUT4_PHASE => 0,
       FEEDBACK_CLK => "CLK0",
       FEEDBACK_MODE => "LOCAL"
       IS_CLKOUT0_INVERTED => 0,
       IS_CLKOUT1_INVERTED => 0,
       IS_CLKOUT2_INVERTED => 0,
       IS_CLKOUT3_INVERTED => 0,
       IS_CLKOUT4_INVERTED => 0,
       CLKOUT3_CONN_TYPE => "GCLK",
       CLKOUT4_CONN_TYPE => "GCLK",
       CLKOUT0_DYNPHASE_EN => 0,
       CLKOUT1_DYNPHASE_EN => 0,
       CLKOUT2_DYNPHASE_EN => 0,
       CLKOUT3_DYNPHASE_EN => 0,
       CLKOUT4_DYNPHASE_EN => 0
    )
    port map (
       CLKOUT0 => clkout0,
       CLKOUT1 => clkout1,
       CLKOUT2 => clkout2,
       CLKOUT3 => clkout3,
       CLKOUT4 => clkout4,
       LOCKED => locked,
       RSTN => rstn,
       FBK => fbk,
       SHIFT_ENA => shift_ena,
       CLKIN => clkin,
       CLKSEL => clksel,
       SHIFT => shift,
       SHIFT_SEL => shift_sel
    );
 end architecture Behavioral;
```

EFX_FPLL_V1

Fractional PLL

This PLL is available in Ti85, Ti135, Ti165, Ti240, and Ti375 FPGAs.

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

EFX_FPLL_V1 Ports

Figure 77: EFX_FPLL_V1 Symbol
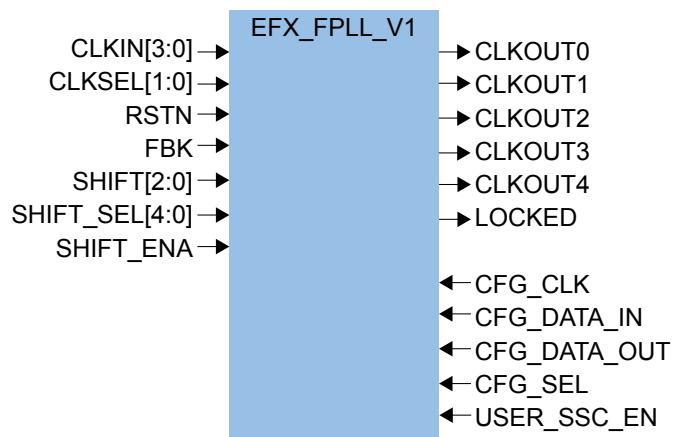
Table 65: EFX_FPLL_V1 Ports

Signal	Direction	Description
CLKIN[3:0]	Input	Reference clocks driven by I/O pads or core clock tree. In dynamic mode, the CLKSEL pin chooses which of these inputs to use.
CLKSEL[1:0]	Input	You can dynamically select the reference clock from one of the clock in pins.
RSTN	Input	(Optional) 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 is not in internal feedback mode. Required when any output is using dynamic phase shift.
CLKOUT0 CLKOUT1 CLKOUT2 CLKOUT3 CLKOUT4	Output	PLL output. You can route these signals as input clocks to the core's GCLK network. The PLL output clock used as the feedback clock can have a maximum frequency of 4x (integer) of the reference clock. If all your system clocks do not fall within this range, you should dedicate one unused PLL output clock for feedback.

Signal	Direction	Description
LOCKED	Output	(Optional) 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 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.
SHIFT[2:0]	Input	(Optional) Dynamically change the phase shift of the output selected to the value set with this signal. Possible values from 000 (no phase shift) to 111 (3.5 F_{PLL} cycle delay). Each increment adds 0.5 cycle delay. Required when any output is using dynamic phase shift.
SHIFT_SEL[4:0]	Input	(Optional) Choose the output(s) affected by the dynamic phase shift. Required when any output is using dynamic phase shift.
SHIFT_ENA	Input	(Optional) When high, changes the phase shift of the selected PLL(s) to the new value. Required when any output is using dynamic phase shift.
CFG_CLK	Input	Configuration clock pin name; used with dynamic configuration.
CFG_DATA_IN	Input	Configuration data input pin name; used with dynamic configuration.
CFG_DATA_OUT	Output	Configuration data output pin name; used with dynamic configuration.
CFG_SEL	Input	Configuration select pin name; used with dynamic configuration.
USER_SSC_EN	Input	User spread-spectrum clocking enable pin name.

EFX_FPLL_V1 Parameters

Table 66: EFX_FPLL_V1 Parameters

Parameter	Allowed Values	Description
N	1, 2, 4	Pre-divider.
M	1, 2, 4	Multiplier.
O	1, 2, 4, 8, 16, 32, 64, 128	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.
CLKOUT3_DIV	1 - 128	Output divider for CLKOUT3.
CLKOUT4_DIV	1 - 128	Output divider for CLKOUT4.
CLKOUT0_PHASE_STEP	0 - 7	Output phase for CLKOUT0.
CLKOUT1_PHASE_STEP	0 - 7	Output phase for CLKOUT1.
CLKOUT2_PHASE_STEP	0 - 7	Output phase for CLKOUT2.
CLKOUT3_PHASE_STEP	0 - 7	Output phase for CLKOUT3.
CLKOUT4_PHASE_STEP	0 - 7	Output phase for CLKOUT4.
FEEDBACK_CLK	CLK0, CLK1, CLK2, CLK3, CLK4	Indicates which clock is the feedback clock.
FEEDBACK_MODE	LOCAL, CORE, EXTERNAL	Indicates the feedback mode.

Parameter	Allowed Values	Description
REFCLK_FREQ	16 - 800	Reference clock frequency.
IS_CLKOUT0_INVERTED	0, 1	Invert output clock for CLKOUT0.
IS_CLKOUT1_INVERTED	0, 1	Invert output clock for CLKOUT1.
IS_CLKOUT2_INVERTED	0, 1	Invert output clock for CLKOUT2.
IS_CLKOUT3_INVERTED	0, 1	Invert output clock for CLKOUT3.
IS_CLKOUT4_INVERTED	0, 1	Invert output clock for CLKOUT4.
CLKOUT3_CONN_TYPE	GCLK, RCLK	Specifies local or regional connection for CLKOUT3. (Not applicable to Ti35 or Ti60)
CLKOUT4_CONN_TYPE	GCLK, RCLK	Specifies local or regional connection for CLKOUT4. (Not applicable to Ti35 or Ti60)
CLKOUT0_DYNPHASE_EN	0, 1	Enable dynamic phase shift control for CLKOUT0.
CLKOUT1_DYNPHASE_EN	0, 1	Enable dynamic phase shift control for CLKOUT1.
CLKOUT2_DYNPHASE_EN	0, 1	Enable dynamic phase shift control for CLKOUT2.
CLKOUT3_DYNPHASE_EN	0, 1	Enable dynamic phase shift control for CLKOUT3.
CLKOUT4_DYNPHASE_EN	0, 1	Enable dynamic phase shift control for CLKOUT4.
DYNAMIC_CFG_EN	0, 1	Enable dynamic reconfiguration.
SCC_MODE	DISABLE, STATIC, DYNAMIC	Spread-spectrum clocking mode.
SCC_FREQUENCY	30 - 33	SSC modulation frequency.
SCC_AMPLITUDE	0 - 0.5	SSC modulation amplitude.
SCC_MODULATION_TYPE	DOWN, UP, CENTER	SSC spread direction.
CLKOUT1_FRAC_EN	0, 1	Enable fractional mode.
CLKOUT1_FRAK_K		Specify fractional coefficient.
CLKOUT1_DC_ODD	0, 1	Enable half VCO shift for CLKOUT1.
CLKOUT1_DC_PDIV	1 - 64	P divider for CLKOUT1.
CLKOUT1_DC_SDIV	1 - 64	S divider for CLKOUT1.
CLKOUT1_PROG_DUTY_CYCLE_EN	0, 1	Enable programmable duty cycle.
CLKOUT1_DUTY_CYCLE	0 - 99	Actual duty cycle for CLKOUT1.

EFX_FPLL_V1 Function

These examples show how to instantiate the fractional PLL V1 primitive.

Figure 78: EFX_FPLL_V1 Verilog HDL Instantiation

```
EFX_FPLL_V1 # (
    .M(1),
    .N(1),
    .O(1),
    .CLKOUT0_DIV(1),
    .CLKOUT1_DIV(1),
    .CLKOUT2_DIV(1),
    .CLKOUT3_DIV(1),
    .CLKOUT4_DIV(1),
    .CLKOUT0_PHASE(0),
    .CLKOUT1_PHASE(0),
```

```
    .CLKOUT2_PHASE(0),
    .CLKOUT3_PHASE(0),
    .CLKOUT4_PHASE(0),
    .FEEDBACK_CLK("CLK0"),
    .FEEDBACK_MODE ("LOCAL"),
    .REFCLK_FREQ(25.0),
    .IS_CLKOUT0_INVERTED(0),
    .IS_CLKOUT1_INVERTED(0),
    .IS_CLKOUT2_INVERTED(0),
    .IS_CLKOUT3_INVERTED(0),
    .IS_CLKOUT4_INVERTED(0),
    .CLKOUT3_CONN_TYPE("GCLK"),
    .CLKOUT4_CONN_TYPE("GCLK"),
    .CLKOUT0_DYNPHASE_EN(0),
    .CLKOUT1_DYNPHASE_EN(0),
    .CLKOUT2_DYNPHASE_EN(0),
    .CLKOUT3_DYNPHASE_EN(0),
    .CLKOUT4_DYNPHASE_EN(0),
    .DYNAMIC_CFG_EN(0),
    .SCC_MODE("NONE"),
    .SCC_FREQUENCY(30),
    .SCC_AMPLITUDE(0),
    .SCC_MODULATION_TYPE("DOWN"),
    .CLKOUT1_FRAC_EN(0),
    .CLKOUT1_FRAK_K(0),
    .CLKOUT1_DC_ODD(0),
    .CLKOUT1_DC_PDIV(0),
    .CLKOUT1_DC_SDIV(0),
    .CLKOUT1_PROG_DUTY_CYCLE_EN(0),
    .CLKOUT1_DUTY_CYCLE(50)
) EFX_FPLL_V1_inst (
    .CLKOUT0(CLKOUT0),
    .CLKOUT1(CLKOUT1),
    .CLKOUT2(CLKOUT2),
    .CLKOUT3(CLKOUT3),
    .CLKOUT4(CLKOUT4),
    .LOCKED(LOCKED),
    .CFG_DATA_OUT(CFG_DATA_OUT),
    .RSTN(RSTN),
    .FBK(FBK),
    .SHIFT_ENA(SHIFT_ENA),
    .CFG_CLK(CFG_CLK),
    .CFG_DATA_IN(CFG_DATA_IN),
    .CFG_SEL(CFG_SEL),
    .USER_SCC_EN(USER_SCC_EN),
    .CLKIN(CLKIN),
    .CLKSEL(CLKSEL),
    .SHIFT(SHIFT),
    .SHIFT_SEL(SHIFT_SEL)
);
```

Figure 79: EFX_FPLL_V1 VHDL Instantiation

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

entity EFX_FPLL_V1_VHDL is
port ( clkout0 : out std_logic;
       clkout1 : out std_logic;
       clkout2 : out std_logic;
       clkout3 : out std_logic;
       clkout4 : out std_logic;
       locked  : out std_logic;
       cfg_data_out : out std_logic;
       fbk : in std_logic;
       shift_ena : in std_logic;
       cfg_clk : in std_logic;
       cfg_data_in : in std_logic;
       cfg_sel : in std_logic;
       user_scc_en : in std_logic;
       rstn : in std_logic;
       clkin : in std_logic(3 downto 0);
       clksel : in std_logic (1 downto 0);
       shift : in std_logic (2 downto 0);
       shift_sel : in std_logic (4 ddownto 0));
end entity EFX_FPLL_V1_VHDL;

architecture Behavioral of EFX_FPLL_V1_VHDL is
 begin
 EFX_FPLL_V1_inst : EFX_FPLL_V1
    generic map (
```

```
        M => 1,
        N => 1,
        O => 1,
        CLKOUT0_DIV => 1,
        CLKOUT1_DIV => 1,
        CLKOUT2_DIV => 1,
        CLKOUT3_DIV => 1,
        CLKOUT4_DIV => 1,
        CLKOUT0_PHASE => 0,
        CLKOUT1_PHASE => 0,
        CLKOUT2_PHASE => 0,
        CLKOUT3_PHASE => 0,
        CLKOUT4_PHASE => 0,
        FEEDBACK_CLK => "CLK0",
        FEEDBACK_MODE => "LOCAL"
        IS_CLKOUT0_INVERTED => 0,
        IS_CLKOUT1_INVERTED => 0,
        IS_CLKOUT2_INVERTED => 0,
        IS_CLKOUT3_INVERTED => 0,
        IS_CLKOUT4_INVERTED => 0,
        CLKOUT3_CONN_TYPE => "GCLK",
        CLKOUT4_CONN_TYPE => "GCLK",
        CLKOUT0_DYNPHASE_EN => 0,
        CLKOUT1_DYNPHASE_EN => 0,
        CLKOUT2_DYNPHASE_EN => 0,
        CLKOUT3_DYNPHASE_EN => 0,
        CLKOUT4_DYNPHASE_EN => 0,
        DYNAMIC_CFG_EN => 0,
        SCC_MODE => "NONE",
        SCC_FREQUENCY => 30,
        SCC_AMPLITUDE => 0,
        SCC_MODULATION_TYPE => "DOWN",
        CLKOUT1_FRAC_EN => 0,
        CLKOUT1_FRAC_K => 0,
        CLKOUT1_DC_ODD => 0,
        CLKOUT1_DC_PDIV => 0,
        CLKOUT1_DC_SDIV => 0,
        CLKOUT1_PROG_DUTY_CYCLE_EN => 0,
        CLKOUT1_DUTY_CYCLE => 50
    )
  port map (
        CLKOUT0 => clkout0,
        CLKOUT1 => clkout1,
        CLKOUT2 => clkout2,
        CLKOUT3 => clkout3,
        CLKOUT4 => clkout4,
        LOCKED => locked,
        CFG_DATA_OUT => cfg_data_out,
        RSTN => rstn,
        FBK => fbk,
        CFG_CLK => cfg_clk,
        CFG_DATA_IN => cfg_data_in,
        CFG_SEL => cfg_sel,
        USER_SCC_EN => user_scc_en,
        SHIFT_ENA => shift_ena,
        CLKIN => clkin,
        CLKSEL => clksel,
        SHIFT => shift,
        SHIFT_SEL => shift_sel
  );
end architecture Behavioral;
```

EFX_OSC_V3

Titanium Oscillator

This oscillator is available in Titanium FPGAs.

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_V3 Ports

Figure 80: EFX_OSC_V3 Symbol

EFX_OSC_V3
→ CLKOUT
← ENA
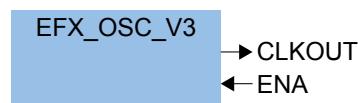
Table 67: EFX_OSC_V3 Ports

Port	Direction	Description
CLKOUT	Output	Oscillator clock output.
ENA	Input	Enable for oscillator.

EFX_OSC_V3 Parameters

Table 68: EFX_IREG Parameters

Parameter	Allowed Values	Description
FREQ	10, 20, 40, 80	Oscillator frequency in MHz.

EFX_OSC_V3 Function

These examples show how to instantiate the OSC V3 primitive.

Figure 81: EFX_OSC_V3 Verilog HDL Instantiation

```
EFX_OSC_V3 # (
    .FREQ(10)
) EFX_OSC_V3_inst(
    .CLKOUT(CLKOUT),    // Output Clock
    .ENA(ENA)           // Clock Enable
);
```

Figure 82: EFX_OSC_V1 VHDL Instantiation

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

entity EFX_OSC_V3_VHDL is
port ( clkout : out std_logic;
       ena : in std_logic);

end entity EFX_OSC_V3_VHDL;

architecture Behavioral of EFX_OSC_V3_VHDL is
 begin
 EFX_OSC_V3_inst : EFX_OSC_V3
    generic map (
       FREQ => 10
    )
    port map (
       CLKOUT => clkout,
       ENA => ena
    );
 end architecture Behavioral;
```

Revision History

Table 69: Revision History

Date	Version	Description
November 2024	2.0	Added interface primitives. (DOC-2086)
June 2024	1.7	For the EXF_DSP12, EXF_DSP24, and EXF_DSP48 primitives, when N_SEL is O the W and O registers must be enabled. (DOC-1863)
November 2023	1.6	Corrected EFX_DPRAM10 Symbol figure signal name. (DOC-1544) Improved Allowed Read and Write Mode Combinations tables for EFX_RAM_5K and EFX_DRRAM_5K. (DOC-1566) Added description about EFX_SRL8 Q7 port and only be connected to the Q of another EFX_SRL8. (DOC-1569)
June 2023	1.5	Updated table EFX_FF Parameters. (DOC-1237) Added in new section for EFX_DSP48, EFX_DSP24, and EFX_DSP12. (DOC-1294)
March 2023	1.4	Updated port map in EFX_DSP48, EFX_DSP 24 and EFX_DSP 12 VHDL Instantiation. (DOC-1110)
August 2022	1.3	The legal values for the EFX_DSP48, EFX_DSP24, and EFX_DSP12 blocks W_SEL parameter are P or X. (DOC-894) Removed the OVFL() signal from the EFX_DSP24 and EFX_DSP12 Verilog HDL examples. These blocks do not have this signal. (DOC-894)
August 2022	1.2	Input output functions at EX4_COMB4 (Arithmetic Mode) at PROP and GEN column. (DOC-848)
April 2022	1.1	The DSP RST_SYNC parameter applies to more than just the A register. (DOC-785)
June 2021	1.0	Initial release.