Efinity Software Quantum Primitives User Guide

UG-EFN-PRIMITIVES-4.1 October 2018

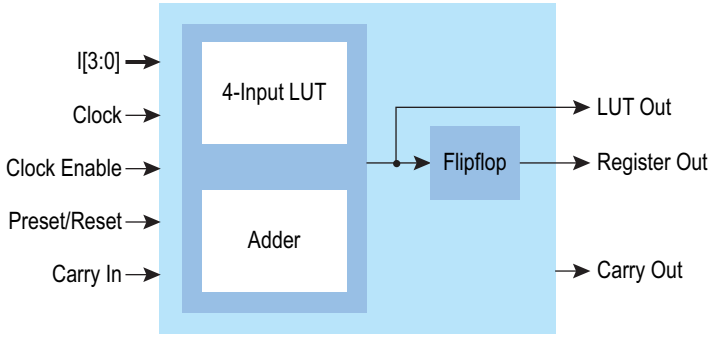
Introduction

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

Logic Cell

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

Figure 1: Logic Cell (Logical View)



Logic cell primitives:

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.

Figure 2: EFX_LUT4 Symbol

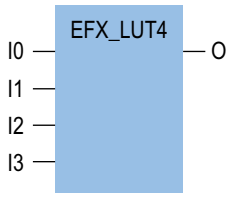


Table 1: EFX_LUT4 Ports

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

Table 2: EFX_LUT4 Parameters

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

Table 3: EFX_LUT4 Function (Part 1 of 2)

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]

Table 3: EFX_LUT4 Function (Part 2 of 2)

Inputs				Output
I3	**I2**	**I1**	**I0**	**O**
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 3: 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
);
```

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.

Figure 4: EFX_ADD Symbol

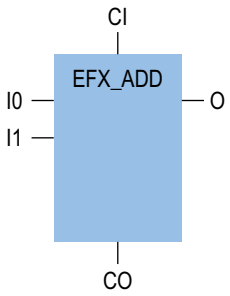


Table 4: EFX_ADD Ports

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

The EFX_ADD parameters control the programmable input inversion.

Table 5: EFX_ADD Parameters

Parameter	Allowed Values	Default	Description
IO_POLARITY	0, 1	1	0 inverting, 1 non-inverting
I1_POLARITY	0, 1	1	0 inverting, 1 non-inverting

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 5: 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
);
```

EFX_FF

D Flipflop with Clock Enable & Set/Reset Pin

The basic EFX_FF primitive is a D flipflop 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 flipflop 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.

Figure 6: EFX_FF Symbol

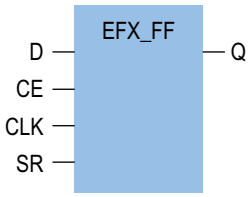


Table 7: EFX_FF Ports

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

Table 8: EFX_FF Parameters

Parameter	Allowed Values	Default	Description
CLK_POLARITY	0, 1	1	0 falling edge, 1 rising edge
CE_POLARITY	0, 1	1	0 active low, 1 active high
SR_POLARITY	0, 1	1	0 active low, 1 active high
D_POLARITY	0, 1	1	0 inverting, 1 non-inverting
SR_SYNC	0, 1	0	0 asynchronous, 1 synchronous
SR_VALUE	0, 1	0	0 reset, 1 set

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 7: 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
);
```

EFX_RAM_5K

5 Kbit RAM Block

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

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

Table 9: EFX_RAM_5K Allowed Read & Write Mode Combinations

	256 x 16	512 x 8	1024 x 4	2048 x 2	4096 x 1	256 x 20	512 x 10	1024 x 5
256 x 16	✓	✓	✓	✓	✓			
512 x 8	✓	✓	✓	✓	✓			
1024 x 4	✓	✓	✓	✓	✓			
2048 x 2	✓	✓	✓	✓	✓			
4096 x 1	✓	✓	✓	✓	✓			
256 x 20						✓	✓	✓
512 x 10						✓	✓	✓
1024 x 5						✓	✓	✓

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

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

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

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

When implementing an EFX_RAM_5K block:

- You must connect the RAM control ports (WCLK, WE, WCLKE, RCLK, and RE). If your design does not use these ports, connect them to ground or V_{CC} depending on their polarity. The software issues a warning if the read or write clock is connected to ground or V_{CC} (except when implementing a ROM).

  The RAM contains an optional output register that improves t_{CO} at a cost of one latency stage. It uses the same control signals as the read port.

- You can only use the address lines that are valid in the particular mode, and you should connect all of them. Leave all other address lines unconnected. Connect required unused address lines to ground.
- Leave unused data lines unconnected.
- When implementing a ROM connect the WE, WCLK, and WCLKE to ground. Leave WDATA unconnected. Connect WADDR to ground or leave it unconnected based on the write mode you select. The write mode must be compatible with the read mode even though the write ports of the ROM are unused.

When you connect the same clock signal to the read and write clock ports, the WRITE_MODE parameter controls the read port behavior when writing:

- READ_FIRST—Old memory content is read (default).
- WRITE_FIRST—Write data is passed to the read port.
- READ_UNKNOWN—Read and writes are unsynchronized, therefore, the results of the address can conflict.

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

Figure 8: EFX_RAM_5K Symbol

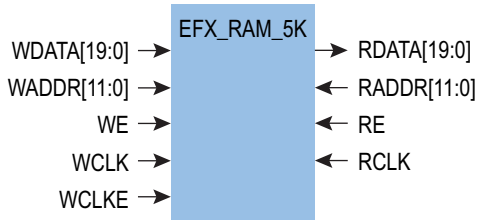


Table 10: EFX_RAM_5K Ports

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

Table 11: EFX_RAM_5K Parameters

Parameter Name	Allowed Values	Default	Description
INIT_<*n*>	256 bit hexadecimal number	0	Initial RAM content
READ_WIDTH WRITE_WIDTH	16	✓	256 x 16
	8		512 x 8
	4		1024 x 4
	2		2048 x 2
	1		4096 x 1
	20		256 x 20
	10		512 x 10
	5		1024 x 5
OUTPUT_REG	0, 1	0	1 enables output register
<*port name*>_POLARITY	0, 1	1	0 active low, 1 active high
WRITE_MODE	READ_FIRST WRITE_FIRST READ_UNKNOWN	READ_FIRST	When using the same clock for RCLK and WCLK, this parameter controls whether the read data is old or new.

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

EFX_RAM_5K Function

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

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

The address at the physical memory array must not conflict.

Figure 9: EFX_RAM_5K Verilog HDL Instantiation

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

EFX_DPRAM_5K

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

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

Table 12: EFX_DPRAM_5K Allowed Read & Write Mode Combinations

	512 x 8	1024 x 4	2048 x 2	4096 x 1	512 x 10 (1)	1024 x 5 (1)
512 x 8	✓	✓	✓	✓		
1024 x 4	✓	✓	✓	✓		
2048 x 2	✓	✓	✓	✓		
4096 x 1	✓	✓	✓	✓		
512 x 10					✓	✓
1024 x 5					✓	✓

Note:
1. 5 Kbits only available in 512 x 10 and 1024 x 5 modes.

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

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

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

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

When connecting the ports, use the following guidelines:

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

- RDATA should be disconnected.

NOTE: Each BRAM output port contains an optional output register to improve t_{CO} at a cost of one stage of latency. It uses the same control signals as the port.

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

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

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

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

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

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

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

When implementing a ROM:

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

Figure 10: EFX_DPRAM_5K Symbol

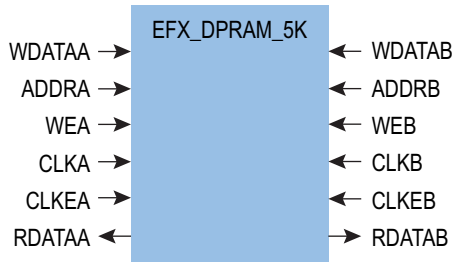


Table 13: EFX_DPRAM_5K Ports

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

Table 14: EFX_DPRAM_5K Parameters

Parameter Name	Allowed Values	Default	Description
INIT_<*n*>	256 bit hexadecimal number	0	Initial RAM content
READ_WIDTH_A READ_WIDTH_B WRITE_WIDTH_A WRITE_WIDTH_B	8	✓	512 x 8
	4		1024 x 4
	2		2048 x 2
	1		4096 x 1
	10		512 x 10
	5		1024 x 5
WRITE_MODE_A WRITE_MODE_B	READ_FIRST	READ_FIRST	Old data is output to RDATA
	WRITE_FIRST		New data is output to RDATA
	NO_CHANGE		RDATA holds last value
OUTPUT_REG_A OUTPUT_REG_B	0, 1	0	1 enables output register
<*port name*>_POLARITY	0, 1	1	0 active low, 1 active high

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

EFX_DPRAM_5K Function

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

Figure 11: EFX_DPRAM_5K Verilog HDL Instantiation

```
EFX_DPRAM_5K #
(
  .READ_WIDTH_A(8),            // 8 512x8
  .WRITE_WIDTH_A(8),           // 8 512x8
  .OUTPUT_REG_A(1'b0),         // 1 add pipe-line read register
  .CLKA_POLARITY(1'b1),        // 0 falling edge, 1 rising edge
  .WEA_POLARITY(1'b1),         // 0 active low, 1 active high
  .CLKEA_POLARITY(1'b1),       // 0 falling edge, 1 rising edge
  .WRITE_MODE_A("READ_FIRST"), // Output "old" data
  .READ_WIDTH_B(8),            // 8 512x8
  .WRITE_WIDTH_B(8),           // 8 512x8
  .OUTPUT_REG_B(1'b0),         // 1 add pipe-line read register
  .CLKB_POLARITY(1'b1),        // 0 falling edge, 1 rising edge
  .WEB_POLARITY(1'b1),         // 0 active low, 1 active high
  .CLKEB_POLARITY(1'b1),       // 0 falling edge, 1 rising edge
  .WRITE_MODE_B("READ_FIRST"), // Output "old" data
  .INIT_0(256'h0000000000000000000000000000000000000000000000000000000000000000),
  .INIT_1(256'h0000000000000000000000000000000000000000000000000000000000000000),
  .INIT_2(256'h0000000000000000000000000000000000000000000000000000000000000000),
  .INIT_3(256'h0000000000000000000000000000000000000000000000000000000000000000),
  .INIT_4(256'h0000000000000000000000000000000000000000000000000000000000000000),
  .INIT_5(256'h0000000000000000000000000000000000000000000000000000000000000000),
  .INIT_6(256'h0000000000000000000000000000000000000000000000000000000000000000),
  .INIT_7(256'h0000000000000000000000000000000000000000000000000000000000000000),
  .INIT_8(256'h0000000000000000000000000000000000000000000000000000000000000000),
  .INIT_9(256'h0000000000000000000000000000000000000000000000000000000000000000),
  .INIT_A(256'h0000000000000000000000000000000000000000000000000000000000000000),
  .INIT_B(256'h0000000000000000000000000000000000000000000000000000000000000000),
  .INIT_C(256'h0000000000000000000000000000000000000000000000000000000000000000),
  .INIT_D(256'h0000000000000000000000000000000000000000000000000000000000000000),
  .INIT_E(256'h0000000000000000000000000000000000000000000000000000000000000000),
  .INIT_F(256'h0000000000000000000000000000000000000000000000000000000000000000),
  .INIT_10(256'h0000000000000000000000000000000000000000000000000000000000000000),
  .INIT_11(256'h0000000000000000000000000000000000000000000000000000000000000000),
  .INIT_12(256'h0000000000000000000000000000000000000000000000000000000000000000),
  .INIT_13(256'h0000000000000000000000000000000000000000000000000000000000000000)
  ) EFX_DPRAM_5K_inst (
    .RDATAA(RDATAA), // Read data output A
    .ADDRA(ADDRA),   // Address input A
    .CLKA(CLKA),     // Clock input A
    .CLKEA(CLKEA),   // Clock-enable input A
    .WEA(WEA),       // Write-enable input A
    .WDATAA(WDATAA), // Write data input A
    .RDATAB(RDATAB), // Read data output B
    .ADDRB(ADDRB),   // Address input B
    .CLKB(CLKB),     // Clock input B
    .CLKEB(CLKEB),   // Clock-enable input B
    .WEB(WEB),       // Write-enable input B
```

```
.WDATAB(WDATAB)  // Write data input B
);
```

EFX_RAM_10K

10 Kbit RAM Block for Quantum Cores

The EFX_RAM_10K primitive represents a configurable 10K bit RAM block that supports a variety of widths and depths. All inputs have programmable inversion, allowing positively or negatively triggered control signals.

The memory read and write ports have 8 modes (512 x 16, 1024 x 8, 2048 x 4, 4096 x 2, 8192 x 1, 512 x 20, 1024 x 10, and 2048 x 5) for addressing the memory. The read and write ports support independently configured data widths.

Table 15: EFX_RAM_10K Allowed Read & Write Mode Combinations

	512 x 16	1024 x 8	2048 x 4	4096 x 2	8192 x 1	512 x 20	1024 x 10	2048 x 5
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 and each parameter represents 256 bits of memory. The memory space covered by each INIT_N parameter uses the formula:

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

When implementing an EFX_RAM_10K block:

- You must connect the RAM control ports (WCLK, WE, WCLKE, RCLK, and RE). If your design does not use these ports, connect them to ground or V_{CC} depending on their polarity. The software issues a warning if the read or write clock is connected to ground or V_{CC} (except when implementing a ROM).

  The RAM contains an optional output register that improves t_{CO} at a cost of one latency stage. It uses the same control signals as the read port.

- You can only use the address lines that are valid in the particular mode, and you should connect all of them. Leave all other address lines unconnected. Connect required unused address lines to ground.
- Leave unused data lines unconnected.
- When implementing a ROM connect the WE, WCLK, and WCLKE to ground. Leave WDATA unconnected. Connect WADDR to ground or leave it unconnected based on the write mode you select. The write mode must be compatible with the read mode even though the write ports of the ROM are unused.

When you connect the same clock signal to the read and write clock ports, the WRITE_MODE parameter controls the read port behavior when writing:

- READ_FIRST—Old memory content is read (default).
- WRITE_FIRST—Write data is passed to the read port.
- READ_UNKNOWN—Read and writes are unsynchronized, therefore, the results of the address can conflict.

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

Figure 12: EFX_RAM_10K Symbol

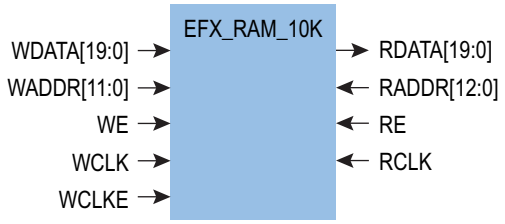


Table 16: EFX_RAM_10K Ports

Port Name	Direction	Description
WDATA[19:0]	Input	Write data
WADDR[12:0]	Input	Write address
WE	Input	Write enable
WCLK	Input	Write clock
WCLKE	Input	Write clock enable
RDATA[19:0]	Output	Read data
RADDR[12:0]	Input	Read address
RE	Input	Read enable
RCLK	Input	Read clock

Table 17: EFX_RAM_10K Parameters

Parameter Name	Allowed Values	Default	Description
INIT_<*n*>	256 bit hexadecimal number	0	Initial RAM content
READ_WIDTH WRITE_WIDTH	16	✓	512 x 16
	8		1024 x 8
	4		2048 x 4
	2		4096 x 2
	1		8192 x 1
	20		512 x 20
	10		1024 x 10
	5		2048 x 5
OUTPUT_REG	0, 1	0	1 enables output register
<*port name*>_POLARITY	0, 1	1	0 active low, 1 active high
WRITE_MODE	READ_FIRST WRITE_FIRST READ_UNKNOWN	READ_FIRST	When using the same clock for RCLK and WCLK, this parameter controls whether the read data is old or new.

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

EFX_RAM_10K Function

The EFXBRAM is physically implemented as a 512 x 40 memory array with decoder logic that maps to the read and write modes. The read and write ports are independent:

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

The address at the physical memory array must not conflict.

Figure 13: EFX_RAM_10K Verilog HDL Instantiation

```
EFX_RAM_10K #
(
.READ_WIDTH(20), // 20 512x20
.WRITE_WIDTH(20), // 20 512x20
.OUTPUT_REG(1'b0), // 1 add pipe-line read register
.RCLK_POLARITY(1'b1), // 0 falling edge, 1 rising edge
.RE_POLARITY(1'b1), // 0 active low, 1 active high
.WCLK_POLARITY(1'b1), // 0 falling edge, 1 rising edge
.WE_POLARITY(1'b1), // 0 active low, 1 active high
.WCLKE_POLARITY(1'b1), // 0 active low, 1 active high
.WRITE_MODE("READ_UNKNOWN"), // Output "unknown" data
.INIT_0(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_1(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_2(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_3(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_4(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_5(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_6(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_7(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_8(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_9(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_A(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_B(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_C(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_D(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_E(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_F(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_10(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_11(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_12(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_13(256'h0000000000000000000000000000000000000000000000000000000000000000),
.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)) EFX-
_RAM_10K_inst (
.RDATA(RDATA), // Read data output
.RADDR(RADDR), // Read address input
.RCLK(RCLK), // Read clock input
.RE(RE), // Read-enable input
.WDATA(WDATA), // Write data input
.WADDR(WADDR), // Write address input
.WCLK(WCLK), // Write clock input
.WE(WE), // Write-enable input
.WCLKE(WCLKE) // Write clock-enable input
);
```

EFX_MULT

18 x 18 Multiplier

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

When implementing an EFX_MULT block:

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

The command line option `--num_mult_18` controls the maximum number of multiplier blocks that the software can infer.

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

Figure 14: EFX_MULT Symbol

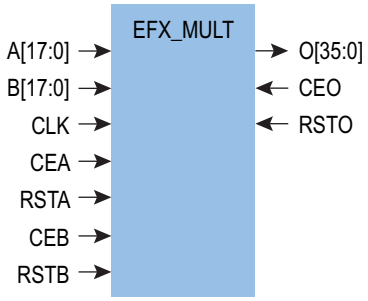


Table 18: EFX_MULT Ports (Part 1 of 2)

Port Name	Direction	Description
A[17:0]	Input	Operand A
B[17:0]	Input	Operand B
CLK	Input	Clock
CEA	Input	Clock enable A
RSTA	Input	Set/reset A
CEB	Input	Clock enable B

Table 18: EFX_MULT Ports (Part 2 of 2)

Port Name	Direction	Description
RSTB	Input	Set/reset B
O[35:0]	Output	Multiplier output
CEO	Input	Clock enable O
RSTO	Input	Set/reset O

Table 19: EFX_MULT Parameters

Parameter Name	Allowed Values	Default	Description
WIDTH	18	18	18 x 18 multiplier
A_REG	0, 1	0	1 enable A registers
B_REG	0, 1	0	1 enable B registers
O_REG	0, 1	0	1 enable output registers
RSTA_SYNC	0, 1	0	0 asynchronous 1 synchronous on A registers
RSTA_VALUE	0, 1	0	0 reset 1 set on A registers
RSTB_SYNC	0, 1	0	0 asynchronous 1 synchronous on B registers
RSTB_VALUE	0, 1	0	0 reset 1 set on B registers
RSTO_SYNC	0, 1	0	0 asynchronous 1 synchronous on output registers
RSTO_VALUE	0, 1	0	0 reset 1 set on output registers
<*port name*>_POLARITY	0, 1	1	0 active low 1 active high

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

EFINIX™
900 Lafayette St, Suite 406
Santa Clara, CA 95050
www.efinixinc.com

EFX_MULT Function

The EFX_MULT is a signed integer multiplier.

Figure 15: EFX_MULT Verilog HDL Instantiation

```
EFX_MULT # (
    .WIDTH(18),
    .A_REG(1),
    .B_REG(1),
    .O_REG(1),
    .CLK_POLARITY(1'b1),  // 0 falling edge, 1 rising edge
    .CEA_POLARITY(1'b1),  // 0 falling edge, 1 rising edge
    .RSTA_POLARITY(1'b0),  // 0 falling edge, 1 rising edge
    .RSTA_SYNC(1'b0),      // 0 aynchronous, 1 synchronous
    .RSTA_VALUE(1'b0),     // 0 reset, 1 set
    .CEB_POLARITY(1'b1),  // 0 falling edge, 1 rising edge
    .RSTB_POLARITY(1'b0),  // 0 falling edge, 1 rising edge
    .RSTB_SYNC(1'b0),      // 0 aynchronous, 1 synchronous
    .RSTB_VALUE(1'b0),     // 0 reset, 1 set
    .CEO_POLARITY(1'b1),  // 0 falling edge, 1 rising edge
    .RSTO_POLARITY(1'b0),  // 0 falling edge, 1 rising edge
    .RSTO_SYNC(1'b0),      // 0 aynchronous, 1 synchronous
    .RSTO_VALUE(1'b0)     // 0 reset, 1 set
    ) mult (
        .CLK(CLK),
        .CEA(CEA),
        .RSTA(SRA),
        .CEB(CEB),
        .RSTB(SRB),
        .CEO(CEO),
        .RSTO(SRO),
        .A(A),
        .B(B),
        .O(O)
);
```

EFX_GBUFCE

Global Clock Buffer

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

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

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

Figure 16: EFX_GBUFCE Symbol

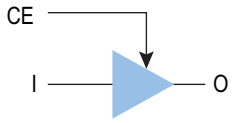


Table 20: EFX_GBUFCE Ports

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

Table 21: EFX_GBUFCE Parameters

Parameter Name	Allowed Values	Default	Description
CE_POLARITY	0, 1	1	0 active low, 1 active high

EFX_GBUFCE Function

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

Table 22: EFX_GBUFCE Function

Inputs		Outputs
CE	I	O
0	X	0
1	0	0
1	1	1

Figure 17: 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
);
```

Revision History

Table 23: Document Revision History

Date	Version	Description
October 2018	4.1	• Added the EFX_RAM_10K primitive. • Removed the EFX_LATCH primitive.

April 2018	4.0	• EFX_RAM_5K—Added description of the READ_UNKNOWN option for the WRITE_MODE parameter.
		• EFX_MULT—Changed the following signals:
		  • SRA to RSTA
		  • SRB to RSTB
		  • SBO to RSTO

November 2017	3.1	• Added EFX_DPRAM_5K primitive. • Updated EFX_RAM_5K description. • Removed EFX_GBUF description.
May 2017	3.0	• Removed OPM family information. • Changed OPH family name to Quantum. • Added GBUF primitive.
May 2016	2.0	Added EFX_RAM_5K and EFX_MULT primitive descriptions.
April 2015	1.0	Initial release.