



# Efinity<sup>®</sup> Synthesis User Guide

---

UG-EFN-SYNTH-v4.4  
November 2025  
[www.efinixinc.com](http://www.efinixinc.com)



# Contents

<b>Introduction.....</b>	<b>4</b>
SystemVerilog and Verilog HDL Support.....	4
VHDL Support.....	4
Specifying Language Support.....	4
Synthesis Project Settings.....	5
Netlist Tab.....	6
<b>Design Guidelines.....</b>	<b>7</b>
DSP.....	7
Inferring DSP.....	7
Using the DSP Block Effectively.....	10
Timing Considerations.....	11
Flip-Flops.....	13
Flip-Flop Reporting.....	14
Flip-Flop Guidelines.....	14
Latches.....	14
RAM.....	15
Inferring RAM.....	15
Estimating Block RAM Resources.....	20
Inferring Shift Registers.....	22
Tri-State Buffers.....	23
<b>Synthesis Options.....</b>	<b>24</b>
Example: --infer-clk-enable.....	28
Example: --create-onehot-fsms.....	29
Example: --allow-const-ram-index.....	29
Example: --msg_suppression_list.....	29
Retiming.....	30
<b>Synthesis Pragmas.....</b>	<b>30</b>
<b>Synthesis Attributes.....</b>	<b>32</b>
async_reg.....	32
mark_debug.....	32
skip_ram_init.....	33
syn_extract_enable.....	33
syn_keep.....	33
syn_peri_port.....	34
syn_preserve.....	35
syn_ramdecomp.....	35
syn_ramstyle.....	36
syn_romstyle.....	36
syn_srlstyle.....	36
syn_use_dsp.....	37
translate_on, translate_off.....	37
<b>Out-of-Context Synthesis.....</b>	<b>38</b>
Export Project Files as OOC Archive.....	38
Generate OOC Files with IP Manager.....	39
Import OOC Files.....	41
<b>Using VHDL Libraries.....</b>	<b>42</b>
Referencing Efinix VHDL Libraries.....	43
<b>VHDL 2008 Support.....</b>	<b>44</b>
Relational Operators (9.2.1).....	44
Condition Operator (9.2.9).....	44
Vector Aggregates (9.3.3).....	45

Conditional and Sequential Statements (10.5.3, 10.5.4).....	45
Case Statements with Don't Care (10.9).....	45
Sensitivity List (11.3).....	45
Generate Statements (11.8).....	46
Expressions in Port Maps (11.8).....	46
Enhanced String Literals (15.8).....	46
Block Comments (15.9).....	46
Fixed-Point Handling (16.10).....	47
Minimum() and Maximum() Functions (16.3).....	47
<b>VHDL 2019 Support.....</b>	<b>48</b>
VHDL 2019 Interface Usage.....	48
VHDL 2019 Example.....	49
<b>SystemVerilog Support.....</b>	<b>51</b>
Hierarchical Names Support.....	54
<b>Warning and Error Messages.....</b>	<b>57</b>
Synthesis Messages.....	57
Post-Synthesis Check Messages.....	71
<b>Where to Learn More.....</b>	<b>79</b>
<b>Revision History.....</b>	<b>80</b>

# Introduction

The Efinity<sup>®</sup> software is a complete tool for creating RTL designs. The first stage after you complete your RTL design is synthesis. During synthesis, the compiler takes your design and turns it into a gate-level netlist.

The software supports the synthesizable subset of the following languages:

- SystemVerilog and Verilog HDL
- VHDL
- Mixed languages (any combination of the above)

## SystemVerilog and Verilog HDL Support

The Efinity<sup>®</sup> software supports the complete IEEE-1800 standard (2017, 2012, 2009, 2005) and includes regular Verilog (IEEE 1164).

- Supports full SystemVerilog IEEE 1800
- Supports Verilog 2001 and Verilog 1995
- Provides 100% language coverage for analysis
- Supports mixed-language with VHDL

## VHDL Support

The Efinity<sup>®</sup> software supports supports the complete IEEE-1076 standard (2008, 1993, 1987) for analysis.

- Supports all of VHDL IEEE 1076
- Includes specialized packages for mixed -1993 / -2008 support
- Provides 100% language coverage for analysis
- Supports mixed-language with SystemVerilog and Verilog HDL
- Supports VHDL libraries (Efinity<sup>®</sup> software v2020.2 and higher)

## Specifying Language Support

You can set the language support at the project level or at the file level. In both cases, you make this setting in the **Design** tab of the Project Editor dialog box. Open the Project Editor by choosing **File > Edit Project** or by clicking the toolbar button.

### Verilog HDL Choices

verilog\_2k  
 verilog\_95  
 SystemVerilog2005  
 SystemVerilog2009

### VHDL Choices

vhdl\_2008  
 vhdl\_1993

# Synthesis Project Settings

You set project-specific synthesis options in the **Project Editor > Synthesis tab**.

**Table 1: Synthesis Project Settings**

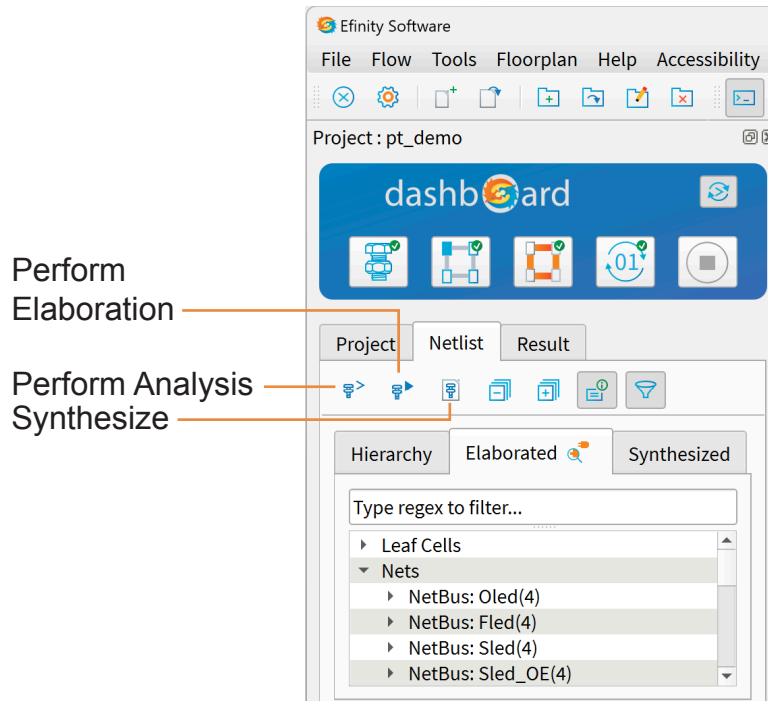
Setting	Description
<b>Work Directory</b>	Specify a custom directory or use the default ( <b>work_syn</b> ).
<b>Generate post synthesis netlist</b>	Choose whether the software should create this netlist. Default: On
<b>Synthesis Options</b>	See <b>Synthesis Options</b> on page 24.
<b>Include Dir</b>	Specify directories to include in your project. If you use the IP Manager to add IP, the <b>ip/&lt;module&gt;</b> directory is listed here. The software searches these locations when you use include statements.
<b>Dynamic Parameter</b>	Use this area to add parameters and values that apply to the top-level module or entity in your project. The value passed into the Dynamic Parameter field must be the same format as that you would use for any variable in VHDL or Verilog HDL. For example, string should be in quotation marks.
<b>Verilog `define Macro</b>	<p>Use this area to add `define macros to your project.</p> <p>Some FPGA EDA tools automatically create a SYNTHESIS macro. If you want to use the same behavior in the Efinity software, you need to create it here. For example, click the Add Verilog `define Macro button and then enter SYNTHESIS in the <b>NAME</b> field and 1 in the <b>Value</b> field. Then if you want to include simulation only code, use this format:</p> <pre style="background-color: #f0f0f0; padding: 10px;">`ifndef SYNTHESIS \$display(...) ... some other simulation directives ... `endif</pre> <p>You can also use the translate_on and translate_off directives to accomplish similar functionality.</p>

## Netlist Tab

The Netlist tab, which is under the Dashboard, shows the design hierarchy and helps you browse through the elaborated design and synthesized netlist. You can only view the synthesized netlist after you have performed synthesis. You can right-click the items in the Netlist tab to open a context-sensitive menu with shortcut actions.

**Tip:** You can resize the Netlist tab. Grab the blank space between the Netlist tab and the Console and drag to resize.

*Figure 1: Using the Netlist Tab*



# Design Guidelines

The following sections provide some general design guidelines.

## DSP

Titanium FPGAs have DSP Blocks that support arithmetic functions such as multiplication, addition, subtraction, accumulation, and 4-bit variable right shifting. The full functionality of the DSP Block is represented with the EFX\_DSP48 primitive. Additionally, the DSP Block supports a concept of *fracturing*, in which the software packs two or more multiplications into a single DSP Block. These fractured blocks are represented with the EFX\_DSP24 and EFX\_DSP12 primitives.



**Note:** Refer to the Quantum® Titanium Primitives User Guide for information on the DSP Block primitives, including the modes and multiplier sizes supported for each primitive.

### Inferring DSP

The Titanium DSP Block supports multiply, add, shift, and cascade functions.

If the signals are registered, use the same clock for all DSP Block inputs and outputs. Additionally, if you use the reset or clock enable, the clocks should be the same. However, not all registers require a reset or clock enable.

- *Example 1*—You use the DSP A\_REG, B\_REG, and W\_REG. They all must be clocked by the same clock signal.
- *Example 2*—A\_REG and C\_REG use a reset signal. B\_REG cannot use a different reset clock, but B\_REG is not required to have a reset.



**Learn more:** Refer to the Trion®, Topaz, and Titanium Quantum primitives user guides for more details on the DSP Block primitives.

The following examples show code to infer them.

#### Figure 2: Inferring Multipliers

When inferring multipliers, synthesis uses different primitives, depending on the width:

- $\leq 4$  infers an EFX\_DSP12
- $\leq 8$  infers an EFX\_DSP24
- $\leq 19, 18$ , infers an EFX\_DSP48

```
`define AWIDTH 4
`define BWIDTH 4
module mult(a, b, x);
  input signed [`AWIDTH-1:0] a;
  input signed [`BWIDTH-1:0] b;
  output signed [`AWIDTH+`BWIDTH-1:0] x;

  assign x = a * b;
endmodule
```

**Figure 3: Inferring Multiply-Accumulate**

This multiply-add can be packed into a single EFX\_DSP48 primitive because the width of C fits into the C port.

```

module dsp_multadd_s(a, b, c, o);
  input signed [17:0] a;
  input signed [17:0] b;
  input signed [17:0] c;

  output signed [35:0] o;

  wire signed [35:0] p;

  assign p = a * b;
  assign o = p + c;

endmodule

```

**Figure 4: Inferring Multiply-Accumulate with Cascading**

Generally, a multiply-add cannot be packed into a single EFX\_DSP48 primitive because the width of C does not fit into the C port. With the `syn_use_dsp` attribute, synthesis uses the A:B:C cascade of another DSP Block to produce a multiply-accumulate.

```

module dsp_multadd_s(a, b, c, o);
  input signed [17:0] a;
  input signed [17:0] b;
  input signed [35:0] c;

  output signed [35:0] o;

  wire signed [35:0] p;
  (* syn_use_dsp = "yes" *) wire [35:0] sum;

  assign p = a * b;
  assign sum = p + c;
  assign o = sum;

endmodule

```

**Figure 5: Inferring Multiply-Accumulate with Output Feeding Back to Accumulate**

The feedback path can take output of the DSP Block and feed it back for accumulation. You enable it with the `syn_use_dsp` attribute.

```

module dsp_multadd_s(a, b, clk, o);
  input signed [17:0] a;
  input signed [17:0] b;
  input clk;

  output signed [35:0] o;

  wire signed [35:0] p;
  (* syn_use_dsp = "yes" *) reg [35:0] sum;

  assign p = a * b;
  always @(posedge clk) begin
    sum <= p + sum;
  end
  assign o = sum;

endmodule

```

### Figure 6: Inferring a Standalone Feedback Accumulate

The DSP block has a feedback path to take the output of the DSP block and feed it back for accumulation. In the following example, input a is wired to the DSP block's C port. The M\_SEL parameter is set to C, the N\_SEL parameter is set to W, and W\_REG is used.

```
module dsp_add_s(a, clk, o);
  input signed [17:0] a;
  input clk;
  output signed [35:0] o;
  (* syn_use_dsp = "yes" *) reg [35:0] sum;
  always @(posedge clk) begin
    sum <= a + sum;
  end
  assign o = sum;
endmodule
```

### Figure 7: Inferring a Shifter

The DSP block has a shift function between the W and O registers. The following code infers the shifter with shift value s connected to C. The SHIFTER parameter is set to 1, and the P\_REG and O\_REG parameters are used. You must also register the shift value with the same clock.

```
`define SIZEA 8
`define SIZEB 8
module dsp_shiftout (a, b, s, clk, x);
  input clk;
  input [`SIZEA-1:0] a;
  input [`SIZEB-1:0] b;
  input [3:0] s;
  output [`SIZEA+`SIZEB-1:0] x;
  reg [`SIZEA+`SIZEB-1:0] p1, p2, p3;
  reg [3:0] sreg;
  always @(posedge clk) begin
    p1 <= a * b;
    p3 <= p1 >> sreg;
    sreg <= s;
  end
  assign x = p3;
endmodule
```

### Figure 8: Inferring a Subtract Operation

In this case, because of a narrower width, the software infers a DSP12 primitive. The first DSP primitive performs the  $p2 = d * c$  operation and the output is fed through the CASCOUT output to the second DSP primitive, which handles the  $p1 = a * b$  and  $x = p1 - p2$  operations (by assigning  $2'b01$  to the OP input).

```
module dsp_cas_sub (a, b, c, d, x);
  input [3:0] a, b, c, d;
  output [7:0] x;
  wire [7:0] p1, p2;
  assign p1 = a * b;
  assign p2 = d * c;
  assign x = p1 - p2;
endmodule
```

## Inferring Wide Multipliers

Multiplication operations wider than 18x17, are decomposed into smaller operations. The software adds the partial sums to form the final product. For example, a 32x32 multiplication is broken down into 4 smaller multiply operations and 3 summations. By default, these partial sums are implemented inside the DSP block using the post-adder feature, and the cascade paths between DSP blocks.

Because long cascade combinational paths can cause slower  $f_{MAX}$ , you can insert pipeline registers in the DSP operation to improve throughput ( $f_{MAX}$ ) at the cost of extra latency.



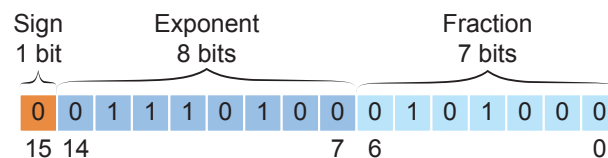
**Warning:** The effect of extra latency in the synthesized netlist must be accounted for in post-synthesis simulation. Otherwise, simulation mismatch may be observed between the RTL and synthesized netlists.

See the `--mult-auto-pipeline` option in [Synthesis Options](#) on page 24.

## Floating-Point Operation

The DSP block supports a fused multiply add (FMA) operation in BFLOAT16 format. BFLOAT16 is mapped to 16 bits as shown in the following figure.

Figure 9: BFLOAT16 Format



Efinity synthesis does not support inferred floating-point operations in Titanium DSP blocks. Instead, To use this function you must instantiate the `EFX_DSP48` primitive and set the `MODE` parameter to `BFLOAT`. In this mode, the DSP block inputs `A`, `B`, and `C` be in BFLOAT16 format, and the resulting output `O` is in 32-bit floating-point format (FP32). The accumulation happens internally in 32-bit floating-point. All pipeline registers (except `O_REG`) must be enabled for correct operation. The following code shows an example instantiation of the primitive:

```
EFX_DSP48 dut (.A(a), .B(b), .C(18'd0), .OP(op), .O(o), .CASCIN({48{1'b0}}),
    .CLK(clk), .RST(rst), .CE(1'b1), .SHIFT_ENA(1'b0));
defparam dut.MODE = "BFLOAT";
defparam dut.M_SEL = "P";
defparam dut.N_SEL = "W";
defparam dut.W_SEL = "X";
defparam dut.A_REG = 1;
defparam dut.B_REG = 1;
defparam dut.C_REG = 0;
defparam dut.P_REG = 1;
defparam dut.OP_REG = 1;
defparam dut.W_REG = 1;
defparam dut.O_REG = 0;
defparam dut.RST_POLARITY = 1;
```

## Using the DSP Block Effectively

Fracturing lets the synthesis tool optimize how it packs operations into the DSP block. The packing density is the percentage of DSP Blocks that are fully occupied with `EFX_DSP24` and/or `EFX_DSP12` primitives. Theoretically, two `EFX_DSP24` primitives or four `EFX_DSP12` primitives pack into one `EFX_DSP48`. In practice, there are some restrictions on how well they can pack: *legality* constraints and *packing quality* constraints.

### Legality Constraints

Synthesis can only pack `EFX_DSP24` and `EFX_DSP12` primitives into the same `EFX_DSP48` if they are the same except for the datapath inputs/outputs. Generally:

- Clock, CE, RST, SHIFT\_ENA, and OP inputs to the `EFX_DSP24` and `EFX_DSP12` to be packed together must be identical nets. You can use `VCC`, `GND`, or disconnected for these ports, as long as they match.
- All DSP primitive Verilog HDL parameters must match, including registered ports (`A_REG`, `B_REG`, `W_REG`) and the mode.

If you are not happy with the DSP Block packing density, you need to modify your design to allow them to pack more effectively.

## Quality Constraints

The software avoids packing random DSP Blocks together because it can have a negative impact on  $f_{MAX}$ . Instead, it tries to pack DSP blocks that are logically related—for example, ones that share neighbouring blocks and input nets—resulting in the best clustering that fits on the FPGA comfortably.

## Improving Packing Density

If you want better DSP Block packing, there are some things you can try:

- Tweak the design such that more DSP blocks have identical control signals and mode settings, and are thus packable. For example, try to avoid letting synthesis infer unique clock enables for every EFX\_DSP24 and/or EFX\_DSP12 primitive.
- Setting the synthesis options `--dspinout-regs-packing`, `--dsp-output-regs-packing`, and `--dsp-mac-packing` to 0 may improve packing density at a cost of increased flipflop and adder consumption. You set these options in the **Project Editor > Synthesis tab**.

You can also use these reports to help with debugging:

- **place.rpt**—This report includes a DSP packing summary that shows the number of DSP Blocks that would be packable if synthesis ignored control signal and attribute-related legality constraints. This number helps you understand whether tweaking the design to improve the packing density is even feasible. In practice, you can get about 50% packing improvement by tweaking the design. The report also lists the control set and attributes for each DSP Block. To be packed, EFX\_DSP24 and EFX\_DSP12 primitives must have matching control sets and attributes. Look for DSP Blocks that do not share control sets or attributes with other blocks, and then look at `dsp_control_sets.csv` for more detailed information on how to potentially adjust them.
- **dsp\_control\_sets.csv**—This file is a table in `.csv` format that lists every control signal and attribute for every DSP Block. You can use a spreadsheet application to review the data to identify EFX\_DSP24 and EFX\_DSP12 primitives that are inferred with unique settings that make packing illegal.

## Timing Considerations

### Closing Timing with High DSP Block Utilization

If your design has a > 50% of the DSP Blocks implemented with EFX\_DSP24 or EFX\_DSP12 primitives, the  $f_{MAX}$  can vary significantly depending on the placement seed. Therefore, it is a good idea to try 3 or 4 seeds to see if it helps with timing closure, more so than for a typical design.



**Learn more:** Refer to the [Efinity Timing Closure User Guide](#) for information on how to perform seed sweeping.

## Wide Multiplier Handling

When a multiplier is in your design's critical path, you may try different synthesis options to handle the decomposed mult-add structure:

1. `--mult-auto-pipeline` automatically adds pipeline registers in the DSP to reduce the critical path delay but increase latency. See [Synthesis Options](#) on page 24 for a detailed description of this option's behavior.
2. `--mult-decomp-ptime` performs backward retiming to reduce the critical path delay. Extra registers should be added to the output of the wide multiplier to allow for retiming to occur.
3. Setting the maximum cascade connected DSP blocks to 2 (`--max-cc-dsp48=2`) may in some cases reduce the critical path delay.

For wide multipliers that exceed the DSP width by a small amount (e.g., 19x19, 20x20), the expanded partial sums or partial products may be too small to efficiently implemented by DSP or CARRY logic. In this case, some of them may be blasted into LUTs. The bit-blasting may hinder the use of the aforementioned command line options for better  $f_{MAX}$  results. In such a case, you may choose to disable small MULT or ADD blasting by setting the following command line options to 0:

1. `--small-adder-limit` (Default: 4)
2. `--small-mult-limit` (Default: 2)

## Flip-Flops

The Efinity<sup>®</sup> synthesis tool recognizes flip-flops (or registers) while processing the RTL. Flip-flops can have these control signals:

- Rising or falling edge clocks
- Asynchronous set/reset
- Synchronous set/reset
- Clock enable

Figure 10: EFX\_FF Symbol

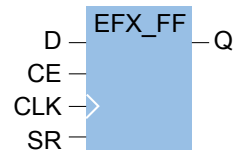


Table 2: EFX\_FF Ports

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

Flip-flops with active-high synchronous resets and active-high clock enables are described as sequential processes in VHDL.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

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

architecture Behavioral of D_FF_VHDL is
begin
  process (clk) is
  begin
    if rising_edge(clk) then
      if (rst='1') then
        q <= '0';
      elsif (ce='1') then
        q <= d;
      end if;
    end if;
  end process;
end architecture Behavioral;
```

They are described with an always statement in Verilog HDL.

```
module D_FF_VERILOG (d, ce, clk, reset, out);
  input d;
  input ce;
  input clk;
  input reset;
  output out;
  reg q;

  always @(posedge clk) begin
    if (reset) q <= 1'b0;
    else if (ce) q <= d;
  end
  assign out = q;
endmodule
```

## Flip-Flop Reporting

The synthesis report (**map.rpt**) shows flip-flop utilization and optimization. For example:

```
Equivalent flip-flop removal reporting:

FF|OPT : Flip-flop optimization by equivalence checking

@ "./zipcpu/idecode.v (350)" removed instance : thecpu/instruction_decoder/dff_195/i7
@ "./zipcpu/idecode.v (350)" representative instance : thecpu/instruction_decoder/dff_196/i7

Clock Enable reporting:

Total number of enable signals: 1199
Enable signal <vcc>, number of controlling flip flops: 256
Enable signal <o_dbg_stall>, number of controlling flip flops: 35

Flip-flop resource summary:

### ### ### Resource Summary (begin) ### ### ###

EFX_FF : 35132
```

## Flip-Flop Guidelines

For best optimization during synthesis, follow these guidelines:

- Avoid using flip-flops with asynchronous set/reset. These structures limit the synthesis tool's ability to optimize the code.
- Avoid flip-flops with both a set and a reset signal. The Trion<sup>®</sup>, Topaz, and Titanium flip-flop only has a single set/reset signal. Therefore, to construct a register with both a set and reset signal, the synthesis must infer additional control logic.

## Latches

If you do not assign an output for all possible conditions in an `if` or `case` statement (that is, incomplete assignment), the software infers a latch. Trion<sup>®</sup>, Topaz, and Titanium FPGAs do not support latches natively in hardware. The Efinity<sup>®</sup> synthesis tool infers look-up tables (LUTs) to provide latch behaviour.

Because latches are turned into LUTs, they can use up resources you could be using for something else. So you want to avoid them when possible.

## RAM

Efnix® FPGAs have embedded RAM blocks that support simple dual-port memory and true dual-port memory. The read and write ports are registered. Asynchronous memory reads (e.g. in asynchronous FIFO or buffer implementation) can be bit-blasted into logic but may cause high device resource utilization. If you do not use the write port, the primitive acts as a ROM.

- *Trion FPGAs*—During synthesis, the memory is mapped to EFX\_RAM\_5K (simple dual port) or EFX\_DPRAM\_5K (true dual port) primitives.
- *Titanium FPGAs*—During synthesis, the memory is mapped to EFX\_RAM10 (simple dual port) or EFX\_DPRAM10 (true dual port) primitives.

The following sections provide code example for inferring these memories.



**Learn more:** Refer to the [Quantum® Trion Primitives User Guide](#) for detailed information on the EFX\_RAM\_5K and EFX\_DPRAM\_5K primitives.

Refer to the [Quantum® Titanium Primitives User Guide](#) for information on the EFX\_RAM10 and EFX\_DPRAM10 primitives.

### Inferring RAM

The following sections provide example for simple and true dual-port inferencing.

#### Simple Dual-Port Memory Examples

The following example infers a 512 x 8 RAM. Because both writes and reads are performed with a blocking statement and the write occurs before the read in the `always` block, the software infers a simple dual ported RAM in `WRITE_FIRST` mode.

**Figure 11: Simple Dual-Port RAM in WRITE\_FIRST Mode**

```
module ram512x8_sp(wdata, addr, clk, we, rdata);
    parameter AWIDTH = 9;
    parameter DWIDTH = 8;
    localparam DEPTH = 1 << AWIDTH;
    localparam MAX_DATA = (1<<DWIDTH)-1;
    input [DWIDTH-1:0] wdata;
    input [AWIDTH-1:0] addr;
    input clk, we;
    output reg [DWIDTH-1:0] rdata;

    reg [DWIDTH-1:0] mem [DEPTH-1:0];

    // Blocking Statement and order indicates write before read
    always@(posedge clk) begin
        if (we) begin
            mem[addr] = wdata;
        end
        rdata = mem[addr];
    end
endmodule
```

If the read and write clocks are different, the software configures the memory primitive as `READ_UNKNOWN`. That is, if you read and write to the same address at the same time, the read data is indeterministic.

**Figure 12: Simple Dual-Port RAM in READ\_UNKNOWN Mode**

```

module ram512x8_sp(wdata, addr, rclk, re, wclk, we, rdata);
    parameter AWIDTH = 9;
    parameter DWIDTH = 8;
    localparam DEPTH = 1 << AWIDTH;
    localparam MAX_DATA = (1<<DWIDTH)-1;
    input [DWIDTH-1:0] wdata;
    input [AWIDTH-1:0] addr;
    input rclk, re, wclk, we;
    output reg [DWIDTH-1:0] rdata;

    reg [DWIDTH-1:0] mem [DEPTH-1:0];

    // different read and write clock, forces READ_UNKNOWN mode
    always@(posedge wclk) begin
        if (we) begin
            mem[addr] = wdata;
        end
    end
    always@(posedge rclk) begin
        if (re) begin
            rdata = mem[addr];
        end
    end
endmodule

```

**Figure 13: Simple Dual-Port RAM with Byte Enable (Verilog HDL) (Titanium)**

```

// 16-bit wide, 512 depth, byte-enabled
// fits into 1 Titanium 10K blockram
module ram10_bel #(
    parameter integer wrAddressWidth = 9, // 512 depth
    parameter integer wrDataWidth = 16, // 16-bit wide
    parameter integer wrMaskWidth = 2,
    parameter integer rdAddressWidth = 9,
    parameter integer rdDataWidth = 16
) (
    input wr_clk,
    input wr_en,
    input [wrMaskWidth-1:0] wr_mask,
    input [wrAddressWidth-1:0] wr_addr,
    input [wrDataWidth-1:0] wr_data,
    input rd_clk,
    input rd_en,
    input [rdAddressWidth-1:0] rd_addr,
    output [rdDataWidth-1:0] rd_data
);

    reg [wrDataWidth-1:0] ram_block [(2*wrAddressWidth)-1:0];
    integer i;
    localparam COL_WIDTH = wrDataWidth/wrMaskWidth;
    always @ (posedge wr_clk) begin
        if (wr_en) begin
            for (i=0; i<wrMaskWidth; i=i+1) begin
                if (wr_mask[i]) begin // byte-enable
                    ram_block[wr_addr][i*COL_WIDTH +: COL_WIDTH] <=
                    wr_data[i*COL_WIDTH +:COL_WIDTH];
                end
            end
        end
    end

    reg [rdDataWidth-1:0] ram_rd_data;
    always @ (posedge rd_clk) begin
        if (rd_en) begin
            ram_rd_data <= ram_block[rd_addr];
        end
    end
    assign rd_data = ram_rd_data;
endmodule

```

**Figure 14: Simple Dual-Port RAM with Byte Enable (VHDL) (Titanium)**

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

-- 512 x 32, with 4 byte-enable signals

entity Memory is
  generic (ADDR_WIDTH: integer := 9);
  Port ( DBOut : out STD_LOGIC_VECTOR (31 downto 0);
        DBIn  : in  STD_LOGIC_VECTOR (31 downto 0);
        AdrBus : in  STD_LOGIC_VECTOR (ADDR_WIDTH-1 downto 0);
        ENA   : in  STD_LOGIC;
        WREN  : in  STD_LOGIC_VECTOR (3 downto 0);
        CLK   : in  STD_LOGIC
        );
end Memory;

architecture Behavioral of Memory is

  constant SIZE : natural := 2**ADDR_WIDTH;
  type tRam is array (0 to SIZE-1) of STD_LOGIC_VECTOR (31 downto 0);
  subtype tWord is std_logic_vector(31 downto 0);

  signal ram : tRam;
  signal DOA,DIA : tWord;
  signal WEA : STD_LOGIC_VECTOR (3 downto 0);

begin

  DIA<=DBIn;
  DBOut<=DOA;
  WEA<=WREN;

  process(clk)
  variable adr : integer;
  begin
    if rising_edge(clk) then
      if ena = '1' then
        adr := to_integer(unsigned(AdrBus));

        for i in 0 to 3 loop
          if WEA(i) = '1' then
            ram(adr)((i+1)*8-1 downto i*8) <= DIA((i+1)*8-1 downto i*8);
          end if;
        end loop;

        DOA <= ram(adr);

      end if;
    end if;
  end process;

end Behavioral;

```

## True Dual-Port Memory Examples

In true dual-port RAM, the two ports have independent read and write functions. Each port supports different write modes. The following example shows how to implement a 512 x 8 RAM block with READ\_FIRST write mode for port A and WRITE\_FIRST mode for port B.

```

module ram512x8_tdp_mix (wdataA, addrA, clkA, weA, rdataA, wdataB, addrB, clkB, weB, rdataB);
    parameter AWIDTH = 9;
    parameter DWIDTH = 8;
    localparam DEPTH = 1 << AWIDTH;
    localparam MAX_DATA = (1<<DWIDTH)-1;

    input [DWIDTH-1:0] wdataA, wdataB;
    input [AWIDTH-1:0] addrA, addrB;
    input  clkA, weA;
    input  clkB, weB;
    output reg [DWIDTH-1:0] rdataA, rdataB;

    reg [DWIDTH-1:0] mem [DEPTH-1:0];

    integer i;
    initial begin
        // The memory is initialized with
        // decreasing values starting from MAX_DATA
        for (i=0; i<DEPTH; i=i+1)
            mem[i] = MAX_DATA - i;
    end

    always@(posedge clkA) begin
        // Use blocking assignments to for read-first
        rdataA = mem[addrA];
        if (weA) begin
            mem[addrA] = wdataA;
        end
    end

    always@(posedge clkB) begin
        // Use blocking assignments to force write-first
        if (weB) begin
            mem[addrB] = wdataB;
        end
        rdataB = mem[addrB];
    end
endmodule

```

## Initializing RAM

Initialize the memory content with the Verilog HDL `$readmemh` or `$readmemb` routines.

**Figure 15: Initializing RAM in Verilog HDL**

```

module ram_256x16 (wdata, waddr, wclk, we, raddr, rclk, re, rdata);
    localparam addr_width = 8;
    localparam data_width = 16;
    input [data_width-1:0] wdata;
    input [addr_width-1:0] waddr, raddr;
    input  wclk, we;
    input  rclk, re;
    output reg [data_width-1:0] rdata;

    reg [data_width-1:0] mem [(1<<addr_width)-1:0];

    integer i;
    initial begin
        // Initialize memory with external file
        $readmemh("ram256x16b.inithex", mem);
    end

    always@(posedge wclk) begin
        if (we)
            mem[waddr] <= wdata;
    end
    always@(posedge rclk) begin
        if (re)
            rdata <= mem[raddr];
    end
endmodule // ram_256x16

```

The memory file should be simple hexadecimal numbers (`$readmemb`) or binary numbers (`$readmemb`) without any comments or prefixes.

**Figure 16: Example Memory File**

```
FE
FD
FC
FB
FA
F9
F8
F7
F6
F5
...
```

## Inferring Output Registers

Synthesis packs registers that immediately follow the read data into the output registers of the BRAM if the control logic is compatible:

- The read clock is the same as the register clock signal.
- Enables:
  - Trion FPGAs—The register must always be enabled (no explicit clock enable control).
  - Titanium FPGAs—The register's clock enable must be the same as the BRAM's read enable signal.
- Resets:
  - Trion FPGAs—The register cannot have a reset signal.
  - Titanium FPGAs—If the read port has a reset signal, and if the register has a reset signal, they must match. Additionally, the Titanium output register only supports asynchronous reset logic.

## Address Enable (Titanium)

The Titanium BRAM supports an address enable feature. However, synthesis does not infer these signals; for inferred BRAM these signals are always tied high. To use the address enable, instantiate the primitive (`EFX_RAM10` and `EFX_DPRAM10`),

## Resetting RAM (Titanium)

Titanium RAM supports a reset option on the RAM output and output register.

**Figure 17: RAM Output with Asynchronous Reset**

```
module ram10_arst (wdata, waddr, wclk, wclke, rclk, we, re, raddr, rdata,
rst);
    parameter AWIDTH = 11; // 2048 depth
    parameter DWIDTH = 4; // 4-bit wide
    localparam DEPTH = 1 << AWIDTH;
    input [DWIDTH-1:0] wdata;
    input [AWIDTH-1:0] waddr, raddr;
    input wclk, wclke, we, rclk, re, rst;
    output reg [DWIDTH-1:0] rdata;

    reg [DWIDTH-1:0] mem [DEPTH-1:0];

    always@(posedge wclk) begin
        if (wclke) begin
            if (we)
                mem[waddr] <= wdata;
            end
        end
    always@(posedge rclk or posedge rst) begin
        if (rst)
            rdata <= 0;
        else if (re)
            rdata <= mem[raddr];
        end
    endmodule
```

## Estimating Block RAM Resources

The Efinity® software v2020.2 (patch 2020.2.299.2.6) and higher includes a Block RAM Resource Estimator that helps you determine how many block RAM resources the software needs for a given memory size. You run this tool at the command line using the `efx_map_ramest` command. The estimator uses these options:

**Table 3: Block RAM Resource Estimator Options**

Option	Description
<code>--help</code>	Display the help.
<code>--family &lt;family name&gt;</code>	Specify the family: Quantum: for Quantum® cores. Trion: for Trion FPGAs. Titanium: for Titanium FPGAs.
<code>--device &lt;FPGA name&gt;</code>	Optional. Specify the FPGA you are targeting.
<code>--mode &lt;mode&gt;</code>	Specify the decomposition mode, speed, area, or power.
<code>--size &lt;memory size&gt;</code>	Specify the memory size as <code>&lt;depth&gt;x&lt;width&gt;</code> .
<code>--size2 &lt;memory size&gt;</code>	If using true dual-port, specify the second port's memory size as <code>&lt;depth&gt;x&lt;depth&gt;</code> .

The following code examples show how to estimate RAM for Trion and Titanium RAM blocks of varying sizes.

### Simple Dual-Port RAM Example (Trion)

The following example command runs the estimator for a Trion FPGA, 10240 X 16 RAM size, and optimizing for area:

```
efx_map_ramest --family Trion --mode area --size 10240x16
```

The command outputs:

```
Efinix Block Ram Resource Estimator
Version: 2020.2.299
Compiled: Dec 30 2020.

Copyright (C) 2013 - 2020 Efinix Inc. All rights reserved.

FPGA Family      : Trion
Block Ram Size   : 5K
Input Memory Size: 10240x16
Mode             : area

Result          : 33 block rams required to implement the above memory size.
```

## True Dual-Port RAM Example (Trion)

The following example command runs the estimator for a Trion FPGA, 10240 X 16 RAM size, 5120 x 32 RAM size, and optimizing for power:

```
efx_map_ramest --family Trion --mode power --size 10240x16 --size2 5120x32
```

The command outputs:

```
Efinix Block Ram Resource Estimator
Version: 2020.2.299
Compiled: Dec 30 2020.

Copyright (C) 2013 - 2020 Efinix Inc. All rights reserved.

FPGA Family      : Trion
Block Ram Size   : 5K
Input Memory Size : 10240x16
2nd Memory Port Size : 5120x32
Mode             : power

Result           : 33 block rams required to implement the above memory size.
```

## Simple Dual-Port RAM Example (Titanium)

The following example command runs the estimator for a Titanium FPGA, 10240 X 16 RAM size, and optimizing for area:

```
efx_map_ramest --family Titanium --mode area --size 10240x16
```

The command outputs:

```
Efinix Block Ram Resource Estimator
Version: 2020.2.299
Compiled: Dec 30 2020.

Copyright (C) 2013 - 2020 Efinix Inc. All rights reserved.

FPGA Family      : Titanium
Block Ram Size   : 10K
Input Memory Size : 10240x16
Mode             : area

Result           : 17 block rams required to implement the above memory size.
```

## True Dual-Port RAM Example (Titanium)

The following example command runs the estimator for a Titanium FPGA, 10240 X 16 RAM size, 5120 x 32 RAM size, and optimizing for area:

```
efx_map_ramest --family Titanium --mode area --size 10240x16 --size2 5120x32
```

The command outputs:

```
Efinix Block Ram Resource Estimator
Version: 2020.2.299
Compiled: Dec 30 2020.

Copyright (C) 2013 - 2020 Efinix Inc. All rights reserved.

FPGA Family      : Titanium
Block Ram Size   : 10K
Input Memory Size : 10240x16
2nd Memory Port Size : 5120x32
Mode             : area

Result           : 17 block rams required to implement the above memory size.
```

## Inferring Shift Registers

Efinity<sup>®</sup> synthesis can infer shift register functions that use the XLR cell's 8-bit shift register function. You do not need to set any synthesis options. For example, synthesis infers the following code as a simple shift register:

```
// 12-bit shift register example
module srl12 (CLK, SI, DO);
input CLK, SI;
output DO;
localparam DATAWIDTH = 12;
reg [DATAWIDTH-1:0] data;

// initializing the shift register content
initial begin
    data = 12'h800;
end

always @(posedge CLK)
begin
    data <= {data[DATAWIDTH-2:0], data[DATAWIDTH-1]};
end
assign DO = data[0];

endmodule
```

The shift register reset is constructed with extra flipflops.

The following example shows a shift register with a reset:

```
// 12-bit shift register with reset example
module srl12_rst (CLK, DI, RST, DO);
input CLK, DI, RST;
output DO;
localparam DATAWIDTH = 12;
reg [DATAWIDTH-1:0] data;
initial begin
    data = 12'h800;
end

always @(posedge CLK)
begin
    if (RST)
        data <= 0;
    else
        data <= {data[DATAWIDTH-2:0], DI};
end
assign DO = data[DATAWIDTH-1];

endmodule
```

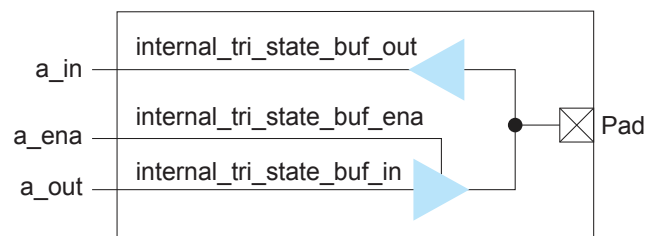
## Tri-State Buffers

Typically, you infer a tri-state buffer in your RTL code. For example:

```
module tri_state_buf_original (a);
    inout a;
    wire b;
    wire oe;
    assign a = (oe ? b : 1'bZ);
endmodule
```

In the Efinity<sup>®</sup> software, however, tri-state buffers are implemented as GPIO blocks in the Interface Designer. The following figure shows a tri-state buffer model in the Efinity<sup>®</sup> software. The Interface Designer promotes all of the internal signals of tri-state buffer to input and output ports of the RTL design (a\_ena, a\_in, and a\_out).

**Figure 18: Tri-State Buffer**



To target Trion<sup>®</sup>, Topaz, and Titanium FPGAs, you map the inout port to three internal nodes (a\_ena, a\_in, and a\_out), export all signals to the top-level module, and then assign the ports to GPIO. The following Verilog HDL code creates the tri-state buffer:

```
module tri_state_buf (a_in, a_out, a_ena,
    internal_tri_state_buf_in,
    internal_tri_state_buf_out,
    internal_tri_state_buf_ena);

    // Split the initial inout port a to three wrapper ports of tri-state buffer
    input a_in;
    output a_out;
    output a_ena;

    // Act as the internal tri state buffer signals.
    output internal_tri_state_buf_out;
    input internal_tri_state_buf_in;
    input internal_tri_state_buf_ena;

    // Modification from traditional way of inferring

    assign internal_tri_state_buf_out = a_in;
    assign a_out = internal_tri_state_buf_in;
    assign a_ena = internal_tri_state_buf_ena;

endmodule
```



**Download:** To download a code example, go to the [How do I create a Tri-State Buffer?](#) topic in the Support Center Knowledgebase.

# Synthesis Options

You can set project-wide synthesis options to control the design flow. You set these options in the Project Editor's **Synthesis** tab. Most options apply to all FPGA families; however, some are specific to Trion, Topaz, or Titanium FPGAs as shown in the following tables.

**Table 4: Synthesis Options (All Families)**

Name	Choices	Description
--allow-const-ram-index	0, 1	Infer RAM if an array is accessed through constant indices. This option can be useful if memory is written such that a constant index refers to each segment (e.g., in a byte-enable read/write). <b>See example.</b> 0: Default. Do not infer. 1: Infer.
--blackbox-error	0, 1	Generate an error when synthesis encounters an undefined instance or entity. 0: No error. 1: Default. Generate error.
--blast_const_operand_adders	0, 1	If one of the operands to an arithmetic operation is constant, implement it as logic instead of adders. 0: Disable. 1: Default. Enable.
--bram_output_regs_packing	0, 1	Enables the software to pack registers into the output of BRAM. 0: Disable. 1: Default. Enable
--bram-push-tco-outreg	0, 1	Retime output registers after address decomposition of BRAM to allow output register packing 0: Default. Disable 1: Enable
--create-onehot-fsms	0, 1	Create onehot encoded state machine when appropriate. Synthesis can only create these state machines if the state variables do not have explicit encoding in the HDL. If a state machine is coded using onehot encoding, a new section in the map report ( <b>&lt;project&gt;.map.rpt</b> ) shows the encoding information. <b>See example.</b> 0: Default. Disabled. 1: Enabled.
--enable_mark_debug	0, 1	Write the <code>mark_debug</code> output to <code>&lt;project dir&gt;/outflow/debug_profile.mark_debug.json</code> to save debug nets for auto debug probe insertion. 0: Disable. 1: Enable (default)
--fanout-limit	0 to <i>n</i>	If something is high fanout, the tool duplicates the fanout source. 0: Default. Disable. <i>n</i> : Indicate the fanout limit at which to begin duplication.

Name	Choices	Description
--hdl-compile-unit	0, 1	When considering multiple source files, resolve `define or parameters independently or across all files. This option only works with SystemVerilog files. 0: Across all files. 1: Default. Independently.
--infer-clk-enable	0, 1, 2, 3, 4	Infer flip-flop clock enables from control logic. <b>See examples.</b> 0: disable. 1, 2, 3, 4: Effort levels.
--infer-sync-set-reset	0, 1	Infer synchronous set/reset signals. 0: Disable. 1: Default. Enable.
--max_ram	-1, 0, <i>n</i>	-1: Default. There is no limit to the number of RAM blocks to infer. 0: Disable. <i>n</i> : Any integer.
--max_mult	-1, 0, <i>n</i>	-1: Default. There is no limit to the number of multipliers to infer. 0: Disable. <i>n</i> : Any integer.
--max_threads	-1, <i>n</i>	Choose how many threads that the synthesis tool can launch. -1: Default. The tool uses the maximum number of available processors. <i>n</i> : Any integer.
--min-sr-fanout	0, <i>n</i>	Infer the flipflop's synchronous set/reset signal from control logic if the set/reset signal fanout is greater than <i>n</i> . This option is useful if the design has a lot of small fanout set/reset signals that may create routing congestion. 0: Default. Disable. <i>n</i> : Signal fanout.
--min-ce-fanout	0, <i>n</i>	Infer the flipflops clock enable from control logic if the clock enable signal fanout is greater than <i>n</i> . 0: Default. Disable. <i>n</i> : Signal fanout.
--mode	speed, area, area2	speed: Default. Optimizes for fastest $f_{MAX}$ . area: Optimizes for smallest area. area2: Uses techniques that help to optimize large multiplexer trees.
--msg_suppression_list	File path	Suppress specific INFO/WARNING messages from synthesis by message IDs. The suppression list file should contain line separated message IDs. Comments are supported. The same suppression list file may be used for both synthesis and place and route. Unrecognized IDs are ignored.

Name	Choices	Description
--mult-auto-pipeline	Integer	<p>Performs automatic pipelining for wide multipliers to increase performance at the cost of extra latency. Inserts pipeline registers at the output of partial multiplies and partial sums. In Titanium and Topaz FPGAs, these pipeline registers can be packed into the DSP48 blocks as W registers. Additional registers are inserted at the input and output of the multiplier to balance latency issues caused by the insertion of the previous registers.</p> <p>The value of this option determines the number of cycles of latency added as a result of inserting pipeline registers. When the value is set to 1, 1 set of pipeline registers are inserted into the wide-multiplier DSP chain. The pipeline register is inserted after the partial adder such that the longest path within the wide-multiply DSP chain is minimized.</p> <p>If the value of this option is set to a number greater than the number of partial adders in the wide-multiply DSP chain, pipeline registers are inserted after the multiplier within the DSP block.</p> <p>Setting this option to a higher value reduces the longest path of the wide-multiplier at the cost of higher latency.</p> <p>Note that these registers may not always be packed into the DSP48 blocks. The maximum value of --mult-auto-pipeline is equivalent to the number of partial adders in the wide-multiply DSP chain + 1. If a number greater than the maximum value is set, synthesis generates a warning message and the maximum value is used instead.</p> <p>0: Default. Disable.</p>
--mult-decomp-rttime	0, 1	<p>Perform retiming after decomposition of a wide multiplier to improve performance.</p> <p>0: Default. Disable. 1: Enable.</p>
--max-bit-blast-mem-size	0 - 99999	<p>Sets the maximum size for memory that is mapped into BRAMs. Memories exceeding the specified size trigger an error message.</p> <p>Default: 10240</p>
--peri-syn-inference	0, 1	<p>Enable unified netlist inference flow. See <a href="#">syn_peri_port</a> on page 34 for the synthesis attribute that you use with this option.</p> <p>0: Default. Disable. 1: Enable.</p>
--peri-syn-instantiation	0, 1	<p>Enable unified netlist instantiation flow. See <a href="#">syn_peri_port</a> on page 34 for the synthesis attribute that you use with this option.</p> <p>0: Default. Disable. 1: Enable.</p>
--operator-sharing	0, 1	<p>Extract shared operators</p> <p>0: Default. Disable 1: Enable</p>
--optimize-adder-tree	0, 1	<p>Optimize skewed adder trees</p> <p>0: Default. Disable 1: Enable</p>

Name	Choices	Description
--optimize-zero-init-rom	0, 1	Optimize ROMs that are initialized to zero. 0: Disable 1: Default. Enable
--retiming	0, 1, 2	Perform retiming optimization. Software moves registers to balance the combinational delay path. 0: Disable. 1: Default. Enable. 2: Advanced algorithm that can benefit some designs.
--seq_opt	0, 1	Turn on sequential optimization. This option can reduce LUT usage but may impact $f_{MAX}$ . 0: Disable. 1: Default. Enable.
--seq-opt-sync-only	0, 1	Sequential synthesis only considers synchronous reset flipflops. 0: Default. Consider all flipflops. 1: Consider synchronous flipflops only.
--suppress_info_msgs	off, on	Suppress INFO messages from synthesis. Off: Default. On: Suppress.
--suppress_warning_msgs	off, on	Suppress WARNING messages from synthesis. Off: Default. On: Suppress.
--use-logic-for-small-mem	0 to $n$	Set the size limit of small RAM blocks implemented in LEs. 0: Disable. 64: Default.
--use-logic-for-small-rom	0 to $n$	Set the size limit of small ROM blocks implemented in LEs. The number is the maximum number of LEs used. 0: Disable. 64: Default.

**Table 5: Synthesis Options (Titanium and Topaz Only)**

Name	Choices	Description
--dsp-input-regs-packing	0, 1	Allow packing of DSP input registers. 0: Disable. 1: Default. Enable.
--dsp-output-regs-packing	0, 1	Allow packing of DSP output registers. 0: Disable. 1: Default. Enable.
--dsp-mac-packing	0, 1	Allow multiplier packing, the software packs adder pairs using multipliers (Trion) or DSP Blocks. 0: Disable. 1: Default. Enable.

Name	Choices	Description
--insert-carry-skip	0, 1	Enable carry-skip optimization for long adders. This option can be useful for designs that have long carry chains. It implements the carry chain with carry skip instead of ripple carry, which can improve performance at the cost of increased area by splitting the carry chains into shorter ones. 0: Default. Disabled. 1: Enable.
--pack-luts-to-comb4	0, 1, 2	Pack compatible LUTs into COMB4 primitives. 0: Default. Disable 1: Effort level 1 2: Effort level 2

**Table 6: Synthesis Options (Trion Only)**

Name	Choices	Description
--mult-input-regs-packing	0, 1	Allow packing of multiplier input registers. 0: Disable. 1: Default. Enable.
--mult-output-regs-packing	0, 1	Allow packing of multiplier output registers. 0: Disable. 1: Default. Enable.

## Example: --infer-clk-enable

The **--infer-clk-enable** synthesis option infers the flip-flop clock enable signal from control logic. This option has three effort levels, or you can disable it.

For example, if you choose effort level 1, this code:

```
always @(posedge clk) begin
    if (e1 | e2) q <= d;
end
```

infers `or (e1, e2)` as the CE pin of a flop with `d` in the D pin and `q` in the Q pin.

In a more complex case, this code:

```
always @(posedge clk) begin
    if (e1) q <= d1;
    else (e2) q <= d2;
end
```

does not result in an inferred clock enable.

With the effort level 3 option, which is the default, the synthesis tool traces multiplexer connections to look for a loop back to the Q pin of the flip-flop. If the tracing is successful, the software extracts the condition pins of the multiplexer path to form the clock enable signal. So in our complex example previously, the software infers `or (e1, e2)` inferred as the clock enable. The level 3 option reduces the number of LUTs by about 4-5% compared to the level 1 option.

## Example: --create-onehot-fsms

The following code snippet shows an example of the FSM encoding section in the **map.rpt** file.

```
### ### Finite State Machine Report (begin) ### ### ###
Module apb3_slave
-----
Recognized state machine 'busState' with states :
"00" : IDLE
"01" : SETUP
"10" : ACCESS

Module axi4_slave
-----
Recognized state machine 'busState' with states :
"000" : IDLE
"001" : PRE_WR
"100" : PRE_RD
"010" : WR
"011" : WR_RESP
"101" : RD
```

## Example: --allow-const-ram-index

The following code snippet demonstrates code that can benefit from the **--allow-const-ram-index** option. Lines 13, 15, 23, and 25 use a constant index to address the first dimension of the memory:

```
subtype t_dim1 is std_logic_vector(7 downto 0);
type t_dim1_vector is array(natural range <>) of t_dim1;
subtype t_dim2 is t_dim1_vector(0 to 511);
type t_dim3_vector is array(natural range <>) of t_dim2;
subtype t_dim3 is t_dim3_vector(0 to 3);

signal RAM : t_dim3 := (others => (others => (others => '0')));
...
RAM3Proc_t : process(Clk)
begin
if(rising_edge(Clk)) then
if(WriteEn = '1') then
RAM(3) (to_integer(unsigned(Addr))) <= WriteData3_t;
end if;
ReadData3_t <= RAM(3) (to_integer(unsigned(Addr)));
end if;
end process RAM3Proc_t;

RAM2Proc_t : process(Clk)
begin
if(rising_edge(Clk)) then
if(WriteEn = '1') then
RAM(2) (to_integer(unsigned(Addr))) <= WriteData2_t;
end if;
ReadData2_t <= RAM(2) (to_integer(unsigned(Addr)));
end if;
end process RAM2Proc_t;

...
```

Setting the **--allow-const-ram-index** option to 1 (enable) instructs the synthesis tool to infer the code as a RAM block.

## Example: --msg\_suppression\_list

The following code snippet shows an example of a message suppression list file.

```
#Syn
```

VERI-1209

```
#PnR
DBMsg::NetlistRemovedLUTBuffers
```

The suppression list file should contain line separated message IDs. To identify the message ID for a given message, look for the name between brackets at the end of the message. Comments are supported.

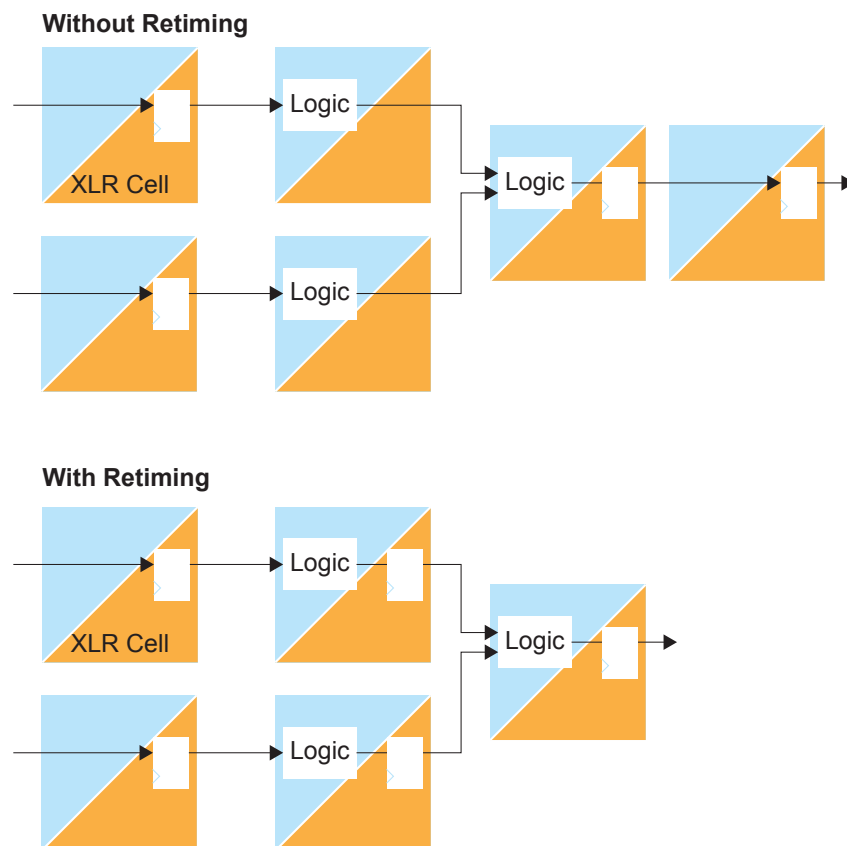
The same suppression list file may be used for both synthesis and place and route. Unrecognized IDs are ignored.

## Retiming

The Efinity® software includes an option for retiming. You enable it by setting the `--retiming` option to 1 in **Project Editor > Synthesis tab**.

When this option is turned on, the software moves registers forward or backward to improve the design's performance. Because the XLR cell comprises both logic and routing, the software can efficiently relocate registers with fine granularity.

*Figure 19: Retiming with XLR Cells*



## Synthesis Pragmas

The Efinity software supports these synthesis pragmas. Put the pragma in a comment, preceded by the `synthesis` keyword.

## synthesis on, synthesis off

This attribute directs synthesis to compile or skip a section of the RTL

## full\_case

This attribute directs synthesis to interpret `case` statements as `full_case` (similar to the Synopsys `full_case` pragma). If you use a `full_case` pragma, synthesis assumes that the listed cases are the *only* possible conditions. All other input combinations are *don't care* and, therefore, synthesis does not generate logic for them

In the following example, the `full_case` pragma directs synthesis to treat the condition `sel = 2'b11` as don't care. The net effect is that the logic used to implement this case statement is simpler.

```
always @(a or b or c or sel) // synthesis full_case
  case (sel)
    2'b00: y = a;
    2'b01: y = b;
    2'b10: y = c;
  endcase
```

## parallel\_case

This attribute directs synthesis to interpret `case` statements as `parallel_case` (similar to the Synopsys `parallel_case` pragma). If you use the `parallel_case` pragma, synthesis does *not* assume that the case conditions are mutually exclusive, that is, they can be true at the same time.

In the following example, if `sel = 3'b111`, all three of the conditions match, and `a`, `b`, and `c` are assigned to `1'b1`.

```
always @(sel)
  begin
    {a, b, c} = 3'b0;
    casez (sel) // synthesis parallel_case
      3'b1??: a = 1'b1;
      3'b?1?: b = 1'b1;
      3'b??1: c = 1'b1;
    endcase
  end
```

# Synthesis Attributes

You use synthesis attributes to guide the Efinity® software to perform (or not perform) certain actions during the synthesis step. The following sections describe the attributes the software supports.

<a href="#">async_reg</a>	<a href="#">syn_skip_ram_init</a>	<a href="#">syn_ramdecomp</a>
<a href="#">syn_extract_enable</a>	<a href="#">syn_keep</a> on page 33	<a href="#">syn_ramstyle</a>
<a href="#">syn_srlstyle</a>	<a href="#">syn_preserve</a>	<a href="#">syn_romstyle</a>
<a href="#">mark_debug</a>	<a href="#">translate_on</a>	<a href="#">syn_use_dsp</a>
	<a href="#">translate_off</a>	

## async\_reg

This attribute applies to register outputs; it designates registers as synchronizers.

When `async_reg` is `true`, synthesis does not perform optimization to reduce, merge, or duplicate these registers. During place and route, the software keeps these registers close together to improve synchronization between asynchronous clock domains.

*Verilog HDL:*

```
(* async_reg = "true" *) reg [1:0] x;
```

*VHDL:*

```
attribute async_reg: boolean;
attribute async_reg of x : signal is true;
```

## mark\_debug

You use this attribute to mark debug nets for auto debug probe insertion. When set to `true` or `1`, the synthesis tool writes out the selected signal to a default file (`<project dir>/outflow/debug_profile.mark_debug.json`).

The `mark_debug` attribute behavior is controlled by **enable-mark-debug**:

- 0—Write an empty **mark\_debug.json** json file
- 1—Synthesis writes the selected signal to the default json file

The attribute is supported in the Efinity software v2025.1 and higher.

*Verilog HDL:*

```
(* mark_debug = "true" *) wire x;
```

*VHDL:*

```
attribute mark_debug: boolean;
attribute mark_debug of x : signal is true;
```

## skip\_ram\_init

This attribute applies to an instantiated block RAM instance. When set to 1 or `true`, synthesis instructs the bitstream generation module to skip including default RAM initialization values to reduce the bitstream size. When the FPGA is configured, the RAM content is not initialized and must be written before being read.

*Verilog HDL:*

```
(*skip_ram_init = 'true'*) EFX_RAM10 dut (...);
```

*VHDL:*

```
attribute skip_ram_init : boolean;
attribute skip_ram_init of u0 : label is true;
```

## syn\_extract\_enable

You use this signal attribute on register outputs. To use this attribute, set it to (`false`, `no`, or `0`). During synthesis, the software will not infer an active clock enable (except when there is a gated clock) on registers with this attribute set.

*Verilog HDL:*

```
(* syn_extract_enable="false" *) reg [3:0] cnt;
```

*VHDL:*

```
attribute syn_extract_enable: boolean;
attribute syn_extract_enable of cnt : signal is false;
```

## syn\_keep

This attribute applies to signals or wires. It is similar to `syn_preserve` except it keeps more than just the signal itself. When it is set to `true`, `yes`, or `1`, the synthesis tool keeps the driver and the loads of the signal through optimization (synthesis does not minimize or remove them).

*Verilog HDL:*

```
(* syn_keep = "true" *) wire x;
```

*VHDL:*

```
attribute syn_keep: boolean;
attribute syn_keep of x : signal is true;
```



**Note:** A signal with `syn_keep` usually has its name preserved through synthesis flow. However, if the signal is connected directly to a top-level port, the name in the `map.v` netlist may be changed to that of the top-level port name.

## syn\_peri\_port

You apply the `syn_peri_port` attribute to top-module ports in your RTL. This option allows you control how inference is done on each I/O port. In the default inference flow, synthesis infers I/O registers or I/O buffers on any I/O port if it is possible. The tool prioritizes inferring I/O registers over I/O buffers (i.e., if an `EFX_IBUF` or `EFX_IREG` can be inferred for a given input port, an `EFX_IREG` will be inferred).



**Learn more:** Refer to the Trion®, Topaz, and Titanium Quantum primitives user guides for primitive definitions.

The attribute has the following settings:

- 0—Do not infer any interface blocks
- 1—Infer buffers only (`EFX_IBUF`, `EFX_OBUF`, `EFX_IO_BUF`)
- 2—Infer registers only (`EFX_IREG`, `EFX_OREG`)
- 3—Infer buffers and registers (default value in inference flow)

This attribute is supported in the Efinity software v2024.2 and higher.

*Verilog HDL:*

```
module irectst_port_attr (clk,i1,o1,i2,o2,i3,o3,i4,o4,i5,o5);
  (* syn_peri_port = 0 *) input clk;
  (* syn_peri_port = 0 *) input i1;
  (* syn_peri_port = 3 *) output o1;
  (* syn_peri_port = 2 *) input i2;
```

*VHDL:*

```
entity top is
port (
  pll_resetrn : out std_logic;
  pll_lock : in std_logic;
  clk : in std_logic;
  clk50 : in std_logic;
);
ATTRIBUTE syn_peri_port : INTEGER;
ATTRIBUTE syn_peri_port OF pll_resetrn : SIGNAL IS 0;
ATTRIBUTE syn_peri_port OF pll_lock : SIGNAL IS 0;
ATTRIBUTE syn_peri_port OF clk : SIGNAL IS 0;
ATTRIBUTE syn_peri_port OF clk50 : SIGNAL IS 0;
```

## syn\_preserve

This attribute applies to signals. When it is set to `true`, `yes`, or `1`, synthesis keeps the signal through optimization, that is, synthesis does not minimize or remove the signal. This attribute can be helpful when you want to simulate or view a signal in the Debugger. Although the signal is kept, synthesis may still choose to implement downstream functions that depend on this signal independent of this preserved signal.

In the Efinity software v2022.2 and higher, the `syn_preserve` attribute is supported on a user hierarchy instance. The effect is equivalent to tagging all boundary signals of the instance with `syn_preserve`.

*Verilog HDL:*

```
(* syn_preserve = "true" *) wire x;
```

*VHDL:*

```
attribute syn_preserve: boolean;
attribute syn_preserve of x : signal is true;
```



**Note:** A signal with `syn_preserve` usually has its name preserved through synthesis flow. However, if the signal is connected directly to a top-level port, the name in the `map.v` netlist may be changed to that of the top-level port name.

## syn\_ramdecomp

This attribute applies to a RAM or ROM signal and controls how synthesis decomposes the RAM or ROM.

- When this attribute is not set, synthesis always chooses data-width decomposition for better performance.
- When set to `area`, synthesis decomposes the RAM or ROM for minimum area (least number of RAM block primitives), but it favors data-width decomposition.
- When set to `power`, synthesis decomposes the RAM or ROM for minimum area, but it favors address decomposition.

*Verilog HDL:*

```
(* syn_ramdecomp = "area" *) reg [DWIDTH-1:0]          mem [DEPTH-1:0];
(* syn_ramdecomp = "power" *) reg [DWIDTH-1:0]        mem [DEPTH-1:0];
```

*VHDL:*

```
attribute syn_ramdecomp: string;
attribute syn_ramdecomp of mem : signal is "power";
```



**Note:** You can use the Block RAM Resource Estimator to explore the number of blocks synthesis will use for various settings. Refer to [Estimating Block RAM Resources](#) on page 20 for details.

## syn\_ramstyle

You apply this attribute to RAM signals:

- The `block_ram` value assigns the signals to block RAM.
- The `registers` value assigns the signals to registers.

*Verilog HDL:*

```
(* syn_ramstyle = "block_ram" *) reg [DWIDTH-1:0] mem [DEPTH-1:0];
(* syn_ramstyle = "registers" *) reg [DWIDTH-1:0] mem [DEPTH-1:0];
```

*VHDL:*

```
attribute syn_ramstyle: string;
attribute syn_ramstyle of mem : signal is "block_ram";
```

## syn\_romstyle

You apply this attribute to ROM signals:

- The `block_rom` value assigns the signals to block ROM.
- The `logic` value assigns the signals to logic.

*Verilog HDL:*

```
(* syn_romstyle = "block_rom" *) reg [DWIDTH-1:0] mem [DEPTH-1:0];
(* syn_romstyle = "logic" *) reg [DWIDTH-1:0] mem [DEPTH-1:0];
```

*VHDL:*

```
attribute syn_romstyle: string;
attribute syn_romstyle of mem : signal is "block_rom";
```

## syn\_srlstyle

This attribute, when applied to a register signal, directs the synthesis inference step to choose between shift register (`srl`), simple register (`registers`), or first register + shift register (`reg_srl`).



**Note:** Shift registers are only available in the Titanium family; therefore, you should only use this attribute when targeting Titanium FPGAs.

*Verilog HDL:*

```
(* syn_srlstyle = "registers" *) reg [WIDTH-1:0] d;
```

*VHDL:*

```
attribute syn_srlstyle: string;
attribute syn_srlstyle of d : signal is "registers";
```

## syn\_use\_dsp

You use this attribute on multiplier output signals.

- When set to `true`, `yes`, or `1`, the synthesis tool implements the multiply function using hard multipliers (that is, the `EFX_MULT` primitive).
- When set to `false`, `no`, or `0`, the synthesis tool generates adders and logic for the multiply function.

In Titanium FPGAs, applying this attribute to the output of an adder that is driven on one side by a multiplier tells synthesis to try to pack the adder into the DSP Block by:

- Using an extra DSP Block to feed the other operand through the `cascin/cascout` path.
- If possible, pack a feedback loop through the `N-SEL` path so that the multiply-accumulate is implemented in a single DSP Block.

*Verilog HDL:*

```
(* syn_use_dsp = "yes" *) signed [27:0] x;
```

*VHDL:*

```
attribute syn_use_dsp: boolean;
attribute syn_use_dsp of x : signal is true;
```

## translate\_on, translate\_off

You use these directives to tell the synthesis tool to ignore the code within them. You should use these together; a `translate_off` should have a corresponding `translate_on`.

For example, the FPGA's flipflop powers up to a 0 value. For simulation, you need to initialize the registers to 0 for the simulation to match this behavior. Using this directive allows the synthesis tool to demonstrate the FPGA's default behavior.

*Verilog HDL:*

```
module in_shifter #(parameter N=2)
(
input data,
input clk,
output reg [N-1:0] out
);

reg [N-1:0] shift_reg;

// synthesis translate_off
initial begin
shift_reg = 0;
out = 0;
end
// synthesis translate_on

always @(posedge clk)
begin
shift_reg <= {shift_reg[N-2:0], data};
out <= shift_reg;
end
endmodule // in_shifter
```

# Out-of-Context Synthesis

Out-of-context (OOC) synthesis is a way to perform synthesis on portions of your design separately and then stitch them together in your top-level design. When you synthesize the top-level design, the software treats the OOC portion(s) as a black box and does not re-synthesize. This flow has a number of uses:

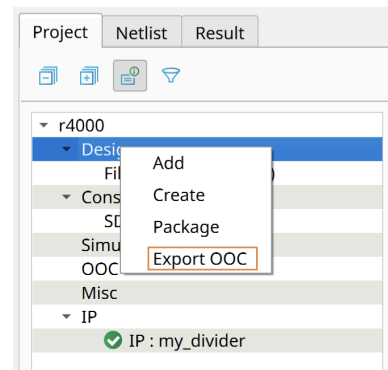
- If you have a large design, you can preserve and re-use the synthesis results from one or more portions of it.
- You can re-use the OOC design in multiple other designs without re-synthesizing.
- You can iterate on the OOC portion of the design independently, saving overall development time.
- If you are working on a team, team members can work on their own OOC designs separately and stitch them together at the end.

The Efinity® software v2025.2 and higher supports OOC synthesis for IP cores and user-created designs.

## Export Project Files as OOC Archive

You can export design files and IP cores as an OOC archive. Right-click the **Design** category or the IP core name in the **Project** pane and choose **Export OOC** from the pop-up menu.

Figure 20: Export OOC Archive



The software synthesizes the design using the default synthesis options, and creates the following files:

- `<module name>.ooc`—Archive file that contains the synthesized design and related files. Use this file to import the synthesized design into another project.
- `<module name>.ooc.vdb`—Synthesized design or IP file.
- `<module name>.stub.v`—Defines the boundaries of the synthesized circuit. The software uses this file when synthesizing your top-level design that includes the OOC archive.

The software saves these files in the:

- **outflow** directory for design files
- `<IP core>/outflow` directory for IP cores



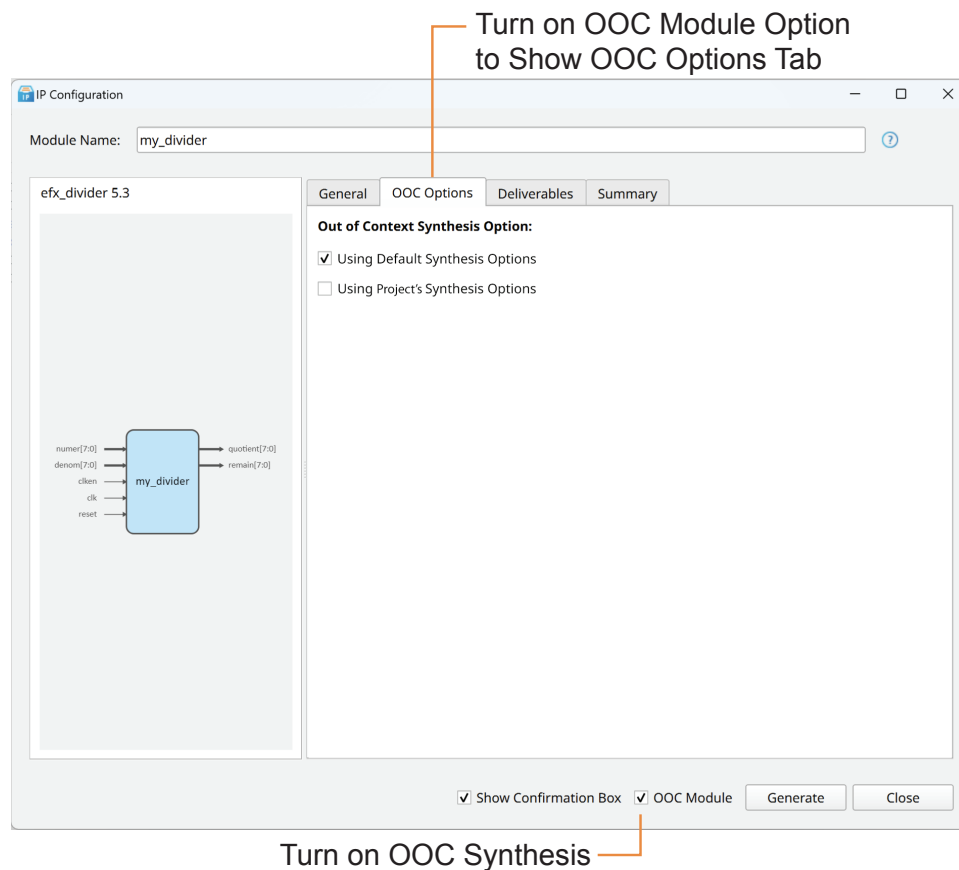
**Important:** To be exported as an OOC archive, your design cannot have parameters in the top module.

## Generate OOC Files with IP Manager

The IP Manager can generate synthesized files for use in OOC flows. When you configure an IP core, you can generate a synthesized file by turning on the **OOC Module** option. When you turn on the option the **OOC Options** tab displays, which lets you choose how to perform synthesis:

- *Using Default Synthesis Options*—The software performs synthesis using the default synthesis options.
- *Using Project's Synthesis Options*—The software performs synthesis using the options specified in the **Project Editor > Synthesis tab > Synthesis Options** box.

Figure 21: Turn On OOC Synthesis for IP Core

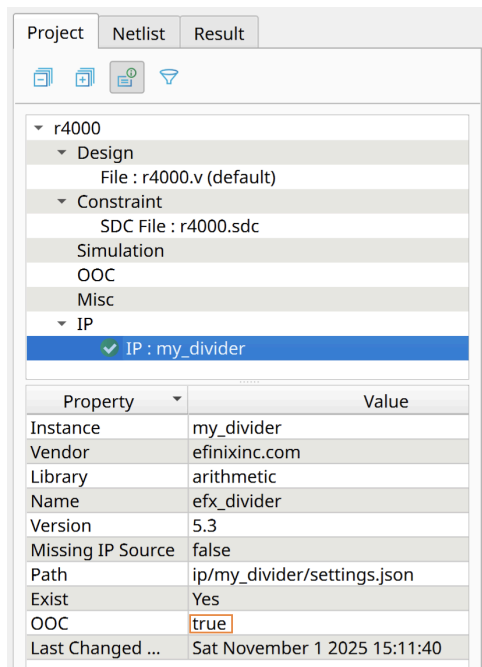


When you generate the IP core, the IP Manager creates these files in the **ip/<IP instance>/outflow** directory:

- **<module name>.ooc.vdb**—Synthesized IP core.
- **<module name>.stub.v**—Defines the boundaries of the synthesized circuit. The software uses this file when synthesizing your top-level design that includes the OOC archive.

When the **OOC Module** is turned on, the IP Manager creates a new version of the **.ooc.vdb** and **.stub.v** every time you re-generate or upgrade the IP core. When you select the IP core in the **Project** pane, the **OOC** property is set to `true`.

Figure 22: OOC Property in Project Pane



To create an OOC archive that you can import into another project, right-click the IP core name in the **Project** pane and choose **Export OOC** from the pop-up menu.



**Note:** If the **OOC Module** option is turned on and you export the core for OOC, the software does not re-synthesize the IP core. It simply creates the OOC archive from the already synthesized IP core (<IP instance>.ooc.vdb).

If the **OOC Module** option is turned off, the software re-synthesizes the design before creating the OOC archive and related files.

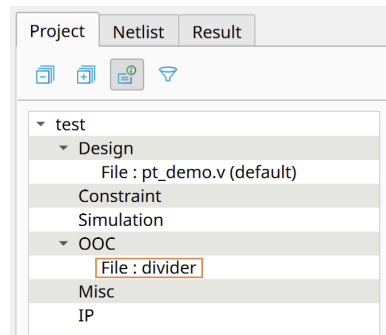
## Import OOC Files

You can import an OOC archive into another project:

1. Ensure that your project's device family and ordering code are the same as the ones used in the OOC design. The software cannot import an OOC archive if the device family and ordering code do not match.
2. Right-click **Project pane** > **OOC** and choose **Import OOC** from the pop-up menu.
3. In the **Import OOC** dialog box, browse to the the OOC archive and click **Open**.

The software extracts the archive and displays it in the **Project pane** > **OOC** category.

*Figure 23: Import OOC Archive*



# Using VHDL Libraries

In the Efinity® software v2020.2 and higher, you can use VHDL libraries to organize and reference commonly used packages and entities.

## Create a Library

To create a library for your project:

1. Open the Project Editor.
2. Click the **Design** tab.
3. Add the design file(s) that have the packages you want to use. You can add multiple files.
4. Double click the cell under **Library**.
5. In the drop-down menu, choose **Add New**.
6. Enter the library name and click **OK**.



**Note:** In VHDL, the **work** library refers to the current library in the design. When assigning a library name to a VHDL design file, you are encouraged not to use the word **work** as the library name only (instead use a variable like name, example: **my\_work**). Doing so will cause an error in synthesis. Leave it blank (or default) if the file is part of the current library in the design project.

7. (Optional) If you add more than one library file to your project, double-click in the **Library** cell for each file and either choose the library name or add a new one.

Library names are saved across projects.

## Add a File to a Library

You add a file to a library in the **Project Editor > Design** tab. Double-click the Library cell for the file and choose the name from the drop-down list.

## Reference a Library

You use the `library` and `use` VHDL language constructs to reference your new library. The following simple code example shows a new library file for the package `mylibrary`:

### Example: mylibrary.vhd

```
--! Use standard library
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

package mylibrary is
  --! factor width
  constant DF_WIDTH : integer := 12;
end package mylibrary;
```

After you add this file to your project and create a library for it, you can refer to the file in your code:

### Example: Referring to the Package

```
--! Use standard library
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

--! Use costum library
library mylibrary;
use mylibrary.mylibrary.all;
```

```

--! Multiplier entity brief description

--! Detailed description of this
--! multiplier design element.
entity multiplier is
  port (
    a    : in  signed (DF_WIDTH-1 downto 0);    --! Multiplier first factor
    b    : in  signed (DF_WIDTH-1 downto 0);    --! Multiplier second factor
    result : out signed (2*DF_WIDTH-1 downto 0) --! Multiplier result
  );
end entity;

```

## Reference Trion and Titanium Primitive Libraries

The Efinity® software includes VHDL libraries for Trion and Titanium primitives. You use the `library` and use VHDL language constructs to reference these libraries:

```

library efxphysicallib;
use efxphysicallib.efxcomponents.all;

```



**Learn more:** The following documents provide example code for these libraries:

[Quantum® Titanium Primitives User Guide](#)

[Quantum® Trion Primitives User Guide](#)

## Referencing Efinix VHDL Libraries

The Efinity® software includes VHDL libraries for Trion and Titanium primitives. You use the `library` and use VHDL language constructs to reference these libraries:

```

library efxphysicallib;
use efxphysicallib.efxcomponents.all;

```

The following code shows how to reference the Efinix library:

### Example: Referring to the Efinix Library

```

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;

```

# VHDL 2008 Support

The Efinity<sup>®</sup> software supports the synthesis subset of the VHDL 2008 standard. The following sections highlight support for new features from an Efinity<sup>®</sup> perspective. This list is not exhaustive.

## Relational Operators (9.2.1)

The software supports these relational operators.

Operator	Description	Usage	Type Returned
=	a is equal to b	a = b	Boolean
/=	a not equal to b	a /= b	Boolean
<	a less than b	a < b	Boolean
<=	a is less than or equal to b	a <= b	Boolean
>	a is greater than b	a > b	Boolean
>=	a is greater than or equal to b	a >= b	Boolean
?=	a is equal to b	a ?= b	Bit or std_logic
?/=	a is not equal to b	a ?/= b	Bit or std_logic
?<	a is less than b	a ?< b	Bit or std_logic
?<=	a is less than or equal to b	a ?<= b	Bit or std_logic
?>	a is greater than b	a ?> b	Bit or std_logic
?>=	a is greater than or equal to b	a ?>= b	Bit or std_logic

In this example, the output is 1 if A(14) is equal 1 and B(14) is equal to 1; otherwise the output is 0.

```
output <= A(14) ?= '1' and B(14);
```

## Condition Operator (9.2.9)

The software supports the condition operator, as shown in the following code example.

```
process(ALL) -- simplify sensitivity list VHDL 2008;
begin
  if A(0) then -- converts std_logic (A(0)) 1,H to boolean True, other will be False
    C(0) <= '1'
  else
    C(0) <= '0';
  end if;
end process;

C(1) <= '1' when A(1) else '0';
```

## Vector Aggregates (9.3.3)

In VHDL 2008, you can use slices in an array aggregation, and you can use aggregates as targets.

```
c(15 downto 8) <= (others =>'1')          when A(8) else --means "11111111"
                  ('1','1','0','1', others =>'0') when A(9) else --means "1101000"
                  (11=>'1', others =>'0')       when A(10) else --means "00001000"
                  ("1101" , others =>'0')       ;               --means "1101000"

(C(39),C(40))<= A(1 downto 0);          -- C(39) becomes A(1) , C(40) becomes A(0)
```

## Conditional and Sequential Statements (10.5.3, 10.5.4)

VHDL 2008 supports conditional and selected sequential statements.

```
DFFwithReset_inst: Process(All)
begin
  if clk'event and clk='1' then
    c(26)<= '0' when reset else A(11); -- c(26) is Q and A(11) is D
  end if;
end process;
```

## Case Statements with Don't Care (10.9)

VHDL 2008 permits a don't care, -, in a case? statement.

```
process(ALL)
begin
  case? A(3 downto 0) is
    when "1---" => c(24) <= '1';
    when "01--" => c(25) <= '1';
    when others => null;
  end case?;
end process;
```

## Sensitivity List (11.3)

You use sensitivity lists to specify a set of signals on which to act. To simplify the sensitivity list and be more comparable with simulation, you can add the keyword ALL to the sensitivity list. The ALL setting adds all signals to the sensitivity list.

```
process(ALL)
```

## Generate Statements (11.8)

In VHDL 2008 you can use case statements in generate statements, and `else/elsif` in if statements.

```
generate_test: case sel generate -- sel must be a constant
  when "00" =>
    compl: entity work.test1(behavior)
      port map(B(13),A(13),C(27));
  when "01" =>
    compl: entity work.test2(behavior)
      port map(B(13),A(13),C(27));
  when others =>
end generate;
```

## Expressions in Port Maps (11.8)

VHDL 2008 allows expressions in port maps.

```
inst1 : entity work.test1(behavior)
  port map (A=>(B(13) AND A(13)), B=> B(15),C=>C(41));
```

## Enhanced String Literals (15.8)

VHDL-2008 enhances bit string literals:

- They may have an explicit width
- They may be declared as signed or unsigned
- They may include meta-values ('U', 'X', etc.)

```
sig(5 downto 0) <= 6x"0F"  when A(2)      else --means 6 bit value "001111"
                  6x"0F"  when A(3)      else --means "001111"
                  6Sx"F"   when A(5)      else --means "111111"      -- sign extension
                  6Ux"F"   when A(6)      else --means "001111"      -- zero extension
                  6SB"11"  when A(7)      else --means "111111"      -- binary format
                  6uO"7";                --means "000111"      -- octal format
```

## Block Comments (15.9)

VHDL you can use single-line comment or delimited comments.

- *Single-line*—These comments begin with two adjacent hyphens `--`, and the comment extends to the end of the line.
- *Delimited*—These comments consist of text surrounded with delimiters. The comment begins with `/*` and continues until `*/`

```
Begin
if A(0) then -- here is a single line comment
  C(0)<='1'
else
  C(0)<='0';
end if;
end process;
/* here is a delimited comment
   that spans two lines */
```

## Fixed-Point Handling (16.10)

VHDL 2008 has fixed-point handling.

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.fixed_pkg.all;

Entity test is
Port (
    s1    : in ufixed(21 downto -2);
    s2    : in ufixed(20 downto -3);
    sum   : out ufixed(22 downto -3);
    prod  : out ufixed(42 downto -5)
);
end test;

Architecture behave of test is
Begin
sum <= s1 +S2;
prod <= S1*S2;
end behave;
```

## Minimum() and Maximum() Functions (16.3)

The software supports the new functions `minimum()` and `maximum()`, which were added in VHDL 2008.

```
min_val <= minimum(a,b) -- if a > b, b is returned
max_val <= maximum(a,b) -- if a > b, a is returned
```

# VHDL 2019 Support

The Efinity<sup>®</sup> software supports the interface feature in the VHDL 2019 standard.

Interfaces play a crucial role in hardware design, serving as essential components. Numerous standardized interfaces exist, such as I<sup>2</sup>C, AXI, or VGA, and user designs incorporate internally developed interfaces to establish connections between different system components. Unfortunately, modeling these interfaces using previous versions of VHDL can be challenging. In most cases, they lack explicit definitions and instead rely on repetitive descriptions within each entity. The only way to identify them is through some naming convention, e.g., all ports of the slave AXI interface could be prefixed with `slave_axi_0_`.

Due to the absence of a centralized definition and reliance on naming conventions, identifying interfaces within complex entities can be challenging. Not only is the interface repeated in every entity that utilizes it, but it is also replicated in each instantiation. Maintaining this duplicated code becomes burdensome. For instance, modifying the type or name of an interface element requires edits in numerous files, even in architectures that solely pass the interface to an instantiation. Because the VHDL language prioritizes strong typing and early bug detection, the interface feature has been added to VHDL-2019.

## VHDL 2019 Interface Usage

You can define VHDL 2019 interfaces using `record` and `view` definitions. In the following example, a record `streaming_bus` is defined in the `interfaces` package. This record defines all the names and types of the elements.

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

package interfaces is
  -- record definition
  type streaming_bus is record
    data_1 : std_logic_vector(3 downto 0);
    data_2 : std_logic_vector(7 downto 0);
  end record;
end;
```

An interface is created for the record `streaming_bus` using the `streaming_master` and `streaming_slave` mode view. You need to define a port mode for each record element. To maximize code reuse, one record can have many mode views.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.interfaces.all; -- USING PACKAGE HERE!

-- Create an interface for the record streaming_bus in the streaming_master mode view
view streaming_master of streaming_bus is
  data_1, data_2 : out;
end view;

-- Create an interface for the record streaming_bus in the streaming_slave mode view
alias streaming_slave is streaming_master'converse;

-- streaming_slave is equivalent to:
-- view streaming_slave of streaming_bus is
--   data_1, data_2 : in;
-- end view;
```

Finally, you can use the `streaming_master` view for the port declarations.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.interfaces.all; -- USING PACKAGE HERE!

-- source uses "master_interface" as the streaming_master interface
entity source is
  --short form
  port(clk, rst : in std_logic; master_interface : view streaming_master);
  -- long form
  -- port(clk, rst : in std_logic;
  --   master_interface : view streaming_master of streaming_bus);
end;
```

## VHDL 2019 Example

The following code illustrates the use of records and views. There are 4 files, **top.vhd**, **interface.vhd**, **source.vhd**, and **sink.vhd**.

The top module is defined as:

```
-- top.vhd top-module
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.interfaces.all; -- USING PACKAGE HERE!

entity top is
  port(clk, rst : in std_logic; data_1_in : in std_logic_vector(3 downto 0);
        data_2_in : in std_logic_vector(7 downto 0);
        data_1_out : out std_logic_vector(3 downto 0);
        data_2_out : out std_logic_vector(7 downto 0));
end;

architecture top_a of top is
  signal bus_signal : streaming_bus;
begin
  producer : entity work.source port map(clk, rst, data_1_in, data_2_in, bus_signal);
  consumer : entity work.sink port map(clk, rst, data_1_out, data_2_out, bus_signal);
end;
```

**interface.vhd** defines the `streaming_master` and `streaming_slave` interfaces.

```
-- interface.vhd
-- Define the streaming_master and streaming_slave interfaces

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

package interfaces is
  -- record definition
  type streaming_bus is record
    data_1 : std_logic_vector(3 downto 0);
    data_2 : std_logic_vector(7 downto 0);
  end record;

  -- Create an interface for the record streaming_bus in the streaming_master mode view
  view streaming_master of streaming_bus is
    data_1, data_2 : out;
  end view;

  -- Create an interface for the record streaming_bus in the streaming_slave mode view
  alias streaming_slave is streaming_master'converse;
end;
```

source receives input data from the top module and sends it to sink using the streaming\_master interface defined in **interface.vhd**.

```
-- source.vhd

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.interfaces.all; -- USING PACKAGE HERE!

-- source uses "master_interface" as the streaming_master interface
entity source is
  -- short form
  port(clk, rst : in std_logic;
        data_1 : in std_logic_vector(3 downto 0);
        data_2 : in std_logic_vector(7 downto 0);
        master_interface : view streaming_master);
end;

architecture source_a of source is
  signal data_1_net : std_logic_vector(3 downto 0) := (others => '0');
  signal data_2_net : std_logic_vector(7 downto 0) := (others => '0');
begin
  process(clk,rst)
  begin
    if rising_edge(clk) then
      if rst='1' then      -- synchronous reset
        data_1_net <= (others => '0');
        data_2_net <= (others => '0');
      else
        data_1_net <= data_1;
        data_2_net <= data_2;
      end if;
      master_interface.data_1 <= data_1_net;
      master_interface.data_2 <= data_2_net;
    end if;
  end process;
end;
```

sink receives data from source through the streaming\_slave interface defined in **interface.vhd**.

```
-- sink.vhd

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.interfaces.all; -- USING PACKAGE HERE!

entity sink is
  port(clk, rst : in std_logic; data_1 : out std_logic_vector(3 downto 0);
        data_2 : out std_logic_vector(7 downto 0);
        slave_interface: view streaming_slave);
end;

architecture sink_a of sink is
  signal data_1_net : std_logic_vector(3 downto 0) := (others => '0');
  signal data_2_net : std_logic_vector(7 downto 0) := (others => '0');
begin
  process(clk,rst)
  begin
    if rising_edge(clk) then
      if rst='1' then      -- synchronous reset
        data_1_net <= (others => '0');
        data_2_net <= (others => '0');
      else
        -- processor receives data_1 and data_2 signal from source and
        -- inverts it before sending to sink
        data_1_net <= not slave_interface.data_1;
        data_2_net <= not slave_interface.data_2;
      end if;
      data_1 <= data_1_net;
      data_2 <= data_2_net;
    end if;
  end process;
end;
```

# SystemVerilog Support

The Efinity<sup>®</sup> software supports the following SystemVerilog constructs:

*Table 7: Support SystemVerilog Constructs*

Data Type	Status
Singular and aggregate types	Supported
Nets and variables	Supported
Variable declarations	Supported
Vector declarations	Supported
2-state (two-value) and 4-state (four-value) data types	Supported
Signed and unsigned integer types	Supported
User-defined types	Supported
Enumerations	Supported
Defining new data types as enumerated types	Supported
Enumerated type ranges	Supported
Type checking	Supported
Enumerated types in numerical expressions	Supported
Enumerated type methods	Supported
Type parameters	Supported
Type operator	Supported
Cast operator	Supported
Const constants	Supported
\$cast dynamic casting	Supported
Real, shortreal, and realtime data types	Supported
<b>Aggregate Data Types</b>	
Structures	Supported
Packed/unpacked structures	Supported
Assigning to structures	Supported
Packed arrays	Supported
Unpacked arrays	Supported
Operations on arrays	Supported
Multidimensional arrays	Supported
Indexing and slicing of arrays	Supported
Array assignments	Supported
Arrays as arguments to subroutines	Supported
Array querying functions	Supported

Data Type	Status
Unpacked unions	Supported
Tagged unions	Not Supported
Packed unions	Supported
<b>Processes</b>	
Combinational logic always_comb procedure	Supported
Implicit always_comb sensitivities	Supported
Latched logic always_latch procedure	Supported
Sequential blocks	Supported
Sequential logic always_ff procedure	Supported
iff event qualifier	Supported
<b>Assignment Statement</b>	
The continuous assignment statement	Supported
Variable declaration assignment (variable initialization)	Supported
Assignment-like contexts	Supported
Array assignment patterns	Supported
Structure assignment patterns	Supported
Unpacked array concatenation	Supported
Net aliasing	Supported
<b>Operators and Expressions</b>	
\$error, \$warning, \$info	Supported only within initial blocks, and can only be used to evaluate constant expressions such as parameters. Can be called from a generate block in SystemVerilog 2009.
Aggregate expressions	Supported
Arithmetic expressions with unsigned and signed types	Supported
Assignment operators	Supported
Assignment within an expression	Supported
Concatenation operators	Supported
Constant expressions	Supported
Increment and decrement operators	Supported
Operations on logic (4-state) and bit (2-state) types	Supported
Wildcard equality operators	Supported
Concatenation of stream_expressions	Supported
Operators with real operands	Supported
Set membership operator	Supported
<b>Procedural Programming Statement</b>	
Loop statements	Supported

Data Type	Status
unique-if, unique0-if, and priority-if	Supported
Assert Statements	Supported
Jump statements	Supported
Pattern matching conditional statements	Supported
unique-case, unique0-case, and priority-case	Supported
<b>Tasks</b>	
Static and automatic tasks	Supported
<b>Functions</b>	
Return values and void functions	Supported
Static and Automatic function	Supported
Constant function	Supported
Virtual functions	Supported
<b>Subroutine Calls and Argument Passing</b>	
Argument binding by name	Supported
Default argument value	Supported
Pass by reference	Supported
Pass by value	Supported
Optional argument list	Supported
<b>Compiler Directives</b>	
Compiler directives	Supported
<b>Modules and Hierarchy</b>	
Module instantiation syntax	Supported
Member selects	Supported
Overriding module parameters	Supported
Top-level modules and \$root	Supported
Binding auxiliary code to scopes or instances	Supported
Hierarchical names	Partially Supported. See <a href="#">Hierarchical Names Support</a> on page 54 for details.
<b>Interfaces</b>	
Interface syntax	Supported
Modport expressions	Supported
Parameterized interfaces	Supported
Ports in interfaces	Supported
Array of interfaces	Supported
Clocking blocks and modports	Supported
Dynamic arrays	Not Supported
Example of exporting tasks and functions	Not Supported
Example of multiple task exports	Not Supported

Data Type	Status
Nested interface	Supported
Virtual interfaces	Supported
<b>Packages</b>	
Package declarations	Supported
Referencing data in packages	Supported
<b>Generate Constructs</b>	
Generate constructs	Supported
<b>Config Statements</b>	
config statements	Supported
<b>Classes</b>	
Instances	Supported
Member and method access	Supported
Constructors	Supported
Static class member and methods	Supported
Access using 'this' and 'super'	Supported
Object assignment	Supported
Inheritance	Supported
Data hiding and encapsulation	Supported
Scope and resolution operator (::)	Supported
Nested classes	Supported
Virtual classes	Supported
Abstract classes	Supported
Assignment with base class object	Not Supported
Object comparison with NULL	Not Supported

## Hierarchical Names Support

Hierarchical names are partially supported in Efinity. This topic lists examples of hierarchical names and their support statuses.

### Named Blocks (Supported)

```

module scope_access (
    input logic [7:0] x,
    output logic [7:0] result
);

    begin : named_scope // named begin-end block
        logic [7:0] scope_var;
        assign scope_var = 8'h33;
    end

    always_comb begin
        result = named_scope.scope_var + x; // hierarchical access
    end

endmodule

```

## Nested Named Blocks (Supported)

```

module nested_scope (
    input logic [7:0] x,
    output logic [7:0] result
);

    begin : outer_scope
        logic [7:0] outer_var;
        assign outer_var = 8'h11 + x;

        begin : inner_scope
            logic [7:0] inner_var;
            assign inner_var = 8'h22 + x + x;
        end
    end

    always_comb begin
        result = outer_scope.outer_var + outer_scope.inner_scope.inner_var;
    end

endmodule

```

## Named Generate Block (Supported)

```

module generate_access (
    input logic [1:0] mult,
    output logic [7:0] result
);

    logic [7:0] sum;

    genvar i;
    generate
        for (i = 0; i < 4; i++) begin : gen_block
            logic [7:0] gen_data;
            assign gen_data = 8'h10 * mult + i;
        end
    endgenerate

    always_comb begin
        // Access generated block variables
        sum = gen_block[0].gen_data +
            gen_block[1].gen_data +
            gen_block[2].gen_data +
            gen_block[3].gen_data;
        result = sum;
    end

endmodule

```

## Named Fork-Join Blocks (Not Supported)

Hierarchical names to named fork-join blocks are not supported.

```

module fork_join_access (
    output logic [7:0] result
);

    initial begin
        // First fork-join block
        fork : mod_1
            begin
                logic [7:0] x;
                mod_2.x = 8'h25;
            end
        join

        // Second fork-join block
        fork : mod_2
            begin
                logic [7:0] x;
                mod_1.x = 8'h35;
            end
        join
    end

    always_comb begin
        result = mod_1.x + mod_2.x;
    end

end

```

```
endmodule
```

## Absolute Path Names (Not Supported)

Absolute path names using `$root` or `$unit` are not supported.

```
module deep_child();
    logic [7:0] deep_data;
    assign deep_data = 8'hAA;
endmodule

module mid_level();
    deep_child u_deep();
endmodule

module absolute_path (
    output logic [7:0] result
);

    logic [7:0] top_data;
    assign top_data = 8'hBB;

    mid_level u_mid();

    always_comb begin
        //Not Supported
        result = $root.absolute_path.u_mid.u_deep.deep_data +
            $root.absolute_path.top_data;
    end

endmodule
```

## External Net Access (Not Supported)

Accessing nets in other modules is not supported.

```
module child_mod();
    logic [7:0] child_data;
    assign child_data = 8'h55;
endmodule

module child_module_var (
    output logic [7:0] result
);

    child_mod u_child();

    always_comb begin
        result = u_child.child_data; // not supported
        // result = child_mod.child_data // also not supported
    end

endmodule
```

## Access Functions or Tasks in Other Modules (Partially Supported)

Accessing functions and tasks in other modules using hierarchical names is supported. However, they can only be accessed by using the module name, not the instance name.

```
module func_provider();
    function automatic logic [7:0] multiply_by_two(
        input logic [7:0] in_val
    );
        return in_val << 1;
    endfunction
endmodule

module function_access (
    input logic [7:0] input_val,
    output logic [7:0] result
);

    func_provider u_func_provider();

    always_comb begin
        result = func_provider.multiply_by_two(input_val); // Supported
        // result = u_func_provider.multiply_by_two(input_val); // Not Supported
    end

endmodule
```

```
end
endmodule
```

## Warning and Error Messages

When you synthesize your design the software performs checks and applies design rules. The following sections list warning and error messages that the software may display and explains how to fix them.

### Synthesis Messages

The software may issue the following messages during synthesis.

#### EFX-0002 (ERROR)

**Message** Illegal command line.  
**To fix** Check the command line help for usage.

#### EFX-0003 (ERROR)

**Message** Unknown Command Line Option : '<option>'.  
**To fix** Check the command line help for usage.

#### EFX-0004 (ERROR)

**Message** Invalid value for option '<option>' in command line.  
**To fix** Check the command line help for usage.

#### EFX-0005 (ERROR)

**Message** Invalid syntax: '<cmd>'.  
**To fix** Check the command line help for usage.

#### EFX-0006 (WARNING)

**Message** Command line option: '<option>' is deprecated and will be ignored.  
**To fix** Option has no effect or is deprecated, refer to the Efinity Synthesis User Guide for expected behaviour.

#### EFX-0007 (WARNING)

**Message** Option value for '<option>' is out of range and will be ignored.  
**To fix** Check the command line help for usage.

#### EFX-0008 (WARNING)

**Message** Top module not specified. Using module '<top>' as top module for the design.  
**To fix** Specify the top module/entity name in the project settings (Project Editor, Design tab, Top Module/Entity box).

## EFX-0010 (VERIFIC\_ERROR)

**Message**            <name> <file: line number>  
**To fix**             Please check and correct the RTL according to the error message.

## EFX-0011 (VERIFIC\_WARNING)

**Message**            <name> <file: line number>  
**To fix**             Please check and correct the RTL according to the error message.

## EFX-0013 (WARNING)

**Message**            Mode '<mode>' is not supported, default to 'speed'.  
**To fix**             Check the command line help for usage.

## EFX-0015 (ERROR)

**Message**            Input source file is not specified.  
**To fix**             Check command line or project settings to include design files.

## EFX-0016 (WARNING)

**Message**            Input source file '<filename>' does not exist and will be skipped.  
**To fix**             Check command line or project settings for the correct design file name and path.

## EFX-0017 (WARNING)

**Message**            Include dir '<directory>' does not exist and will be skipped.  
**To fix**             Check command line or project settings for the correct include path.

## EFX-0018 (WARNING)

**Message**            -f file '<filename>' does not exist and will be skipped.  
**To fix**             Check command line or project settings for the correct -f file name and path.

## EFX-0019 (ERROR)

**Message**            Analysis of source file '<file>' failed. Please check previous error messages.  
**To fix**             Please check and correct the RTL according to the error message.

## EFX-0020 (ERROR)

**Message**            Elaboration of design failed. Please check previous error messages.  
**To fix**             Please check and correct the RTL according to the error message.

## EFX-0021 (ERROR)

**Message**            Elaboration of module '<module>' failed. Please check previous error messages.  
**To fix**             Please check and correct the RTL according to the error message.

## EFX-0022 (WARNING)

- Message** Top module is not specified. Top level parameters will be ignored.
- To fix** Check command line or project settings for correct top module and parameter specifications.

## EFX-0023 (WARNING)

- Message** Top level parameters list must have an even number of elements. (Ignoring parameters)
- To fix** Check command line or project settings to ensure you specified the correct parameter names/values.

## EFX-0024 (ERROR)

- Message** Cannot instantiate unknown module '<instance name>'. <file: line number>
- To fix** Ensure that the module's source code is included in your design files.

## EFX-0025 (ERROR)

- Message** Illegal instantiation of empty module <instance name>. EFX\_PRAGMA\_\_IS\_USER\_DECLARED\_BLACK\_BOX pragma should be set for empty modules. <file: line number>
- To fix** Check whether an empty module is intended; if so, use a pragma on the module.

## EFX-0026 (ERROR)

- Message** Black box instantiation '<instance name>' with inout ports is not supported. <file: line number>
- To fix** Synthesis does not support a black box with an 'inout' port.

## EFX-0027 (WARNING)

- Message** Specified map library '<library>' already exists in lib containers, please specify new library (skipping mapping).
- To fix** Map library with the same name is already loaded. Please use a different map library.

## EFX-0029 (WARNING)

- Message** Settings\_file '<filename>' does not exist and will be ignored.
- To fix** Check command line or project settings for the correct settings file name and path.

## EFX-0030 (ERROR)

- Message** Inferred latch '<instance name>' is not supported. <file: line number>
- To fix** Check your RTL to fix the unsupported latch type in the design.

## EFX-0032 (WARNING)

- Message** primitive '<instance name>' has high impedance (1'bz) input, rewiring to GND. <file: line number>
- To fix** Check whether the unconnected signal or unspecified input to the instance is intended.

## EFX-0033 (WARNING)

- Message** Project XML file specified. Command line option --f files are ignored.
- To fix** The file list in project settings takes precedence over command line --f files.

## EFX-0034 (WARNING)

- Message** Project XML file specified. Command line --v source files are ignored.
- To fix** The file list in project settings takes precedence over command line --v files.

## EFX-0035 (INTERNAL\_ERROR)

- Message** Component '<instance name>' (prim\_mux) is not fully connected. <file: line number>
- To fix** Consult Efinix Support for INTERNAL ERROR messages.

## EFX-0037 (ERROR)

- Message** Cell '<cell>' has compiled netlist, but it does not have matching ports. <errmsg>
- To fix** Please check and correct the RTL according to the error message.

## EFX-0039 (INTERNAL\_ERROR)

- Message** Fail to parse elaborated netlist to find clock domains for MARK\_DEBUG signal(s). Debugger profile will not be generated.
- To fix** Consult Efinix Support on INTERNAL ERROR messages.

## EFX-0100 (INTERNAL\_ERROR)

- Message** I/O mapping failed for <direction> port : '<port name>'
- To fix** Consult Efinix Support on INTERNAL ERROR messages.

## EFX-0101 (INTERNAL\_ERROR)

- Message** I/O mapping failed (<description>).
- To fix** Consult Efinix Support on INTERNAL ERROR messages.

## EFX-0102 (INTERNAL\_ERROR)

- Message** I/O port pre-synthesis failed.
- To fix** Consult Efinix Support on INTERNAL ERROR messages.

## EFX-0103 (INTERNAL\_ERROR)

- Message** Failed to create netlist for logic synthesis.
- To fix** Consult Efinix Support on INTERNAL ERROR messages.

## EFX-0104 (INTERNAL\_ERROR)

- Message** Logic synthesis netlist checking failed.
- To fix** Consult Efinix Support on INTERNAL ERROR messages.

## EFX-0107 (INTERNAL\_ERROR)

- Message** Failed to decompose wide multiplier '<instance name>' (blasting to logic). <file: line number>
- To fix** Consult Efinix Support on INTERNAL ERROR messages.

## EFX-0109 (ERROR)

- Message** Incompatible architecture setting for multipliers.
- To fix** The selected options are incompatible, refer to the Efinity Synthesis User Guide for the expected behaviour.

## EFX-0110 (ERROR)

- Message** Multiplier '<instance name>' illegal instantiation <family> fabric does not support <multiplier size> multiplier. <file: line number>
- To fix** Check your RTL and fix any unsupported or incorrect primitives.

## EFX-0111 (ERROR)

- Message** Incompatible architecture setting for RAM blocks.
- To fix** The selection options are incompatible, refer to the Efinity Synthesis User Guide for the expected behaviour.

## EFX-0112 (WARNING)

- Message** Write mode '<mode name>' for memory block '<instance name>' is not supported in this architecture. READ\_UNKNOWN mode will be used. <file: line number>
- To fix** The targeted device only supports READ\_UNKNOWN mode in memory.

## EFX-0113 (ERROR)

- Message** Memory block '<instance name>' has WRITE\_MODE='WRITE\_FIRST', but read and write widths do not match. <file: line number>
- To fix** The targeted device only supports WRITE\_FIRST mode if the read and write widths match. Change the write mode or use matching read/write widths.

## EFX-0114 (ERROR)

- Message** Memory block '<instance name>' has WRITE\_MODE='WRITE\_FIRST', but read and write address widths do not match. <file: line number>
- To fix** The targeted device only supports WRITE\_FIRST mode if the read and write address widths match. Change the write mode or use matching read/write address widths.

## EFX-0115 (WARNING)

- Message** Memory block '<instance name>' has async read and write ports. WRITE\_MODE will be set to \"READ\_UNKNOWN\". <file: line number>
- To fix** If the read and write clocks are different, the target device's memory only supports READ\_UNKNOWN mode.

## EFX-0116 (ERROR)

- Message** Memory block '<instance name>' has illegal combination of read/write widths: r=<rmode>,w=<wmode>. <file: line number>
- To fix** Read and write data widths of the memory instance are not available in this device.

## EFX-0118 (ERROR)

- Message** Memory block '<instance name>' has WRITE\_MODE set to <mode> but has async read and write ports. Async read/write ports only support WRITE\_MODE=\"READ\_UNKNOWN\". <file: line number>
- To fix** If the read and write clocks are different, the target device's memory only supports READ\_UNKNOWN mode.

## EFX-0120 (WARNING)

- Message** Memory block '<instance name>' has mis-matched read/write address. WRITE\_MODE cannot be set to \"WRITE\_FIRST\". It will be set to \"READ\_UNKNOWN\" instead. <file: line number>
- To fix** The targeted device only supports WRITE\_FIRST mode if the read and write address widths match. Change the write mode or use matching read/write address widths.

## EFX-0200 (WARNING)

- Message** Removing redundant signal : <net name>. <file: line number>
- To fix** Check your RTL to confirm the unused signal is intended.

## EFX-0201 (WARNING)

- Message** Re-wiring to GND non-driven net '<net name>'. <file: line number>
- To fix** Check your RTL to confirm the undriven signal is intended.

## EFX-0202 (ERROR)

- Message** Net '<net name>' has multiple drivers. <file: line number>
- To fix** Check your RTL to fix the multiple driver signal.

## EFX-0203 (WARNING)

- Message** Instance <instance name> has unconnected port <port name>. <instance, file: line number>
- To fix** Check your RTL to confirm the undriven port is intended.

## EFX-0204 (ERROR)

- Message** Top design has no ports.
- To fix** Check your RTL to make sure top level ports are declared.

## EFX-0205 (ERROR)

- Message** Primitive instance '<instance name>' net '<net name>' is not driven. <net, file: line number>
- To fix** Check your RTL to make sure required ports are connected.

## EFX-0206 (ERROR)

- Message** Instance '<instance name>' of type '<cell name>' is unsupported. <instance, file: line number>
- To fix** The target device does not support primitives of this type. Check your RTL to make sure it uses the correct primitive.

## EFX-0207 (WARNING)

- Message** '<instance name>' port '<port name>' is wired to wide vector, synthesis will keep only LSB connection. <instance, file: line number>
- To fix** Check your RTL to make sure the trimming is intended.

## EFX-0209 (ERROR)

- Message** Primitive '<instance name>' is not supported. Please check your command line settings. <file: line number>
- To fix** Fabric option is not set to support instantiated primitives, refer to the Efinity Synthesis User Guide for expected behaviour.

## EFX-0210 (ERROR)

- Message** No top design.
- To fix** No top design module/entity available. Check your RTL and project settings.

## EFX-0250 (ERROR)

- Message** Primary output net '<net name>' is not driven by EFX\_IOBUF.
- To fix** Consult Efinix Support on INTERNAL ERROR messages.

## EFX-0251 (ERROR)

- Message** Primary input port '<port name>' must connect to the core only through an IOBUF or a GBUFCE.
- To fix** Consult Efinix Support on INTERNAL ERROR messages.

## EFX-0252 (ERROR)

- Message** Primary input port '<port name>' is driven by IOBUF, but not connected to the 'IO' pin.
- To fix** Consult Efinix Support on INTERNAL ERROR messages.

## EFX-0253 (ERROR)

- Message** Primary input port '<port name>' is connected to an IOBUF, but is not connected to the core.
- To fix** Consult Efinix Support on INTERNAL ERROR messages.

## EFX-0254 (ERROR)

- Message** The primary output port '<output>' is not wired to a net.
- To fix** Check RTL to make sure all primary input/output ports are connected.

## EFX-0255 (ERROR)

- Message** The primary input port '<input>' should not be driven by output '<instancePort>' of module/instance '<instanceName>'.
- To fix** Primary input port is driven by module output. Check your RTL to make sure connections are legal.

## EFX-0256 (WARNING)

- Message** The primary output port '<port name>' wire '<net name>' is not driven.
- To fix** Primary output port is not driven. Check your RTL to make sure connections are legal.

## EFX-0257 (INTERNAL\_ERROR)

- Message** Number of ports do not match in pre/post map netlists (<pre-map number> vs <post-map number>).
- To fix** Consult Efinix Support on INTERNAL ERROR messages.

## EFX-0258 (INTERNAL\_ERROR)

- Message** Number of port-buses do not match in pre/post map netlists (<pre-map number> vs <post-map number>).
- To fix** Consult Efinix Support on INTERNAL ERROR messages.

## EFX-0259 (INTERNAL\_ERROR)

- Message** Primary ports mismatch in pre/post map netlists.
- To fix** Consult Efinix Support on INTERNAL ERROR messages.

## EFX-0260 (INTERNAL\_ERROR)

- Message** Primary port-buses mismatch in pre/post map netlists.
- To fix** Consult Efinix Support on INTERNAL ERROR messages.

## EFX-0261 (INTERNAL\_ERROR)

- Message** Primary connected ports : (<number>) and IOBUFS (<number>) count comparison failure: number of connected primary ports must be greater or equal to the number of IOBUFS (note some primary ports are clock ports and are connected to GBUFCE).
- To fix** Consult Efinix Support on INTERNAL ERROR messages.

## EFX-0262 (ERROR)

- Message** Primary port nets mismatch in pre/post map netlists.
- To fix** Consult Efinix Support on INTERNAL ERROR messages.

## EFX-0263 (INTERNAL\_ERROR)

- Message** Primary port '<port name>' has illegal net name '<net name>'. Net name should match port name.
- To fix** Consult Efinix Support on INTERNAL ERROR messages.

## EFX-0264 (INTERNAL\_ERROR)

- Message** Primary port '<port name>' net has multiple port connections.
- To fix** Consult Efinix Support on INTERNAL ERROR messages.

## EFX-0265 (ERROR)

- Message** Bi-directional port '<net name>' is not supported in this device. Check synthesis options.
- To fix** Bi-directional port is not supported. Check your RTL for correct use of ports.

## EFX-0267 (ERROR)

- Message** Interface block instance '<instance name>' pin '<pin name>' is not connected to dedicated top-level port.
- To fix** The interface block pin should be connected to top level port. Check your RTL for proper connection.

## EFX-0268 (WARNING)

- Message** Interface block instance '<instance name>' pin '<pin name>' is connection to dangling net '<net name>' and the connection to core will be removed.
- To fix** The interface block pin connection is not used. Check your RTL to see if this is intended.

## EFX-0298 (WARNING)

- Message** Instantiated primitive '<instance name>' has unused pin '<pin name>' based on mode. Pin connection will be removed. <file: line number>
- To fix** Primitive has unused connection. Check your RTL to correct the issue.

## EFX-0300 (WARNING)

- Message** Instantiated LUT '<instance name>' has LUTMASK '<lutMask>'. It does not depend on connected input '<port name>'. Connection will be removed. <file: line number>
- To fix** Primitive LUT may have redundant input based on LUTMASK. Check your RTL to see if this is intended.

## EFX-0301 (ERROR)

- Message** Instantiated LUT '<instance name>' has LUTMASK '<lutMask>'. It depends on unconnected input '<port name>'. <file: line number>
- To fix** Primitive LUT is missing required input based on LUTMASK. Check and fix your RTL.

## EFX-0302 (ERROR)

- Message** Instantiated LUT '<instance name>' has LUTMASK '<lutMask>'. It does not fit into a 4-input LUT. <file: line number>
- To fix** Primitive LUT LUTMASK requires more than 4 inputs. Check and fix your RTL.

## EFX-0305 (ERROR)

- Message** Instantiated LUT '<instance name>' has duplicated inputs at ports : '<port 1 name>' and '<port 2 name>'.
- To fix** LUT should not have duplicated inputs. If the primitive is instantiated, check and fix the your RTL.

## EFX-0350 (ERROR)

- Message** EFX\_ADD '<instance name>' CI pin can only be driven by CO pin of another EFX\_ADD.
- To fix** EFX\_ADD primitive CI can only be driven by CO. If the primitive is instantiated, check and fix the your RTL.

## EFX-0351 (ERROR)

- Message** EFX\_ADD '<instance name>' CO pin should only drive CI pin.
- To fix** EFX\_ADD primitive CO can only drive one CI. If the primitive is instantiated, check and fix your RTL.

## EFX-0352 (ERROR)

- Message** EFX\_ADD '<instance name>' CO net drives multiple loads.
- To fix** EFX\_ADD primitive CO can only drive one CI. If the primitive is instantiated, check and fix your RTL.

## EFX-0354 (ERROR)

- Message** Instantiated EFX ADD '<instance name>' port '<port name>' (net '<net name>') has illegal CI/CO connection.
- To fix** EFX\_ADD primitive CI/CO has an illegal connection. If the primitive is instantiated, check and fix your RTL.

## EFX-0375 (ERROR)

- Message** GBUFCE instance '<instance name>' has illegal connections.
- To fix** EFX\_GBUFCE primitive has an illegal connection. If the primitive is instantiated, check and fix your RTL.

## EFX-0376 (ERROR)

- Message** Clock pin '<clock pin name>' of instance '<instance name>' is not driven.
- To fix** Clock pin of the primitive is not driven. If the primitive is instantiated, check and fix your RTL.

## EFX-0377 (ERROR)

- Message** Clock port '<port name>' of instance '<instance name>' is constant. <instance, file: line number>
- To fix** Clock pin of the primitive is tied to constant. If the primitive is instantiated, check and fix your RTL.

## EFX-0378 (WARNING)

- Message** Clock-enable port '<port name>' of instantiated FF '<instance name>' is disabled.
- To fix** Clock enable port of the primitive is always disabled. Check your RTL to see if this is intended.

## EFX-0379 (WARNING)

- Message** Clock-enable port '<port name>' of instance '<instance name>' is tied to VCC, but polarity is active-low.
- To fix** Clock enable port of the primitive is always disabled. Check your RTL to see if this is intended.

## EFX-0382 (ERROR)

- Message** 'I' pin of GBUF/GBUFCE instance '<instance name>' should only be connected to primary input port.
- To fix** EFX\_GBUF primitive has an illegal connection. If the primitive is instantiated, check and fix your RTL.

## EFX-0383 (ERROR)

- Message** Clock pin '<port name>' of instance '<instance name>' is not connected. <instance, file: line number>
- To fix** Clock pin of the primitive is not driven. If the primitive is instantiated, check and fix your RTL.

## EFX-0384 (WARNING)

- Message** Cannot determine clock domain for MARK\_DEBUG signal '<sigName>', skipping it. <net, file: line number>
- To fix** Cannot determine clock domain, choose a different MARK\_DEBUG signal in your RTL.

## EFX-0400 (ERROR)

- Message** Illegal connection of IOBUF '<instance name>' O net to '<instance name>' instance.
- To fix** The EFX\_IOBUF port is not connected correctly. If the primitive is instantiated, check and fix your RTL.

## EFX-0410 (ERROR)

- Message** Signal driving output port '<port name>' is tristated, which is not supported. <net, file: line number>
- To fix** A tri-state signal driving an output port is not supported. If the primitive is instantiated, check and fix your RTL.

## EFX-0449 (ERROR)

- Message** Instance '<instance name>' non-clock pin '<pin name>' is driven by GBUFCE '<name>'.
- To fix** EFX\_GBUFCE can only drive clock pins. If the primitive is instantiated, check and fix your RTL.

## EFX-0471 (ERROR)

- Message** Memory block '<name>' has illegal mode <mode> '<value>'. <file: line number>
- To fix** Memory has an illegal parameter. If the primitive is instantiated, check and fix your RTL.

## EFX-0473 (ERROR)

- Message** <cell name> '<instance name>' is not supported in <family> device. <instance, file: line number>
- To fix** RAM primitive is not supported in the device. Check and fix your RTL.

## EFX-0500 (INTERNAL\_ERROR)

- Message** library model has unexpected port '<library name>'.
- To fix** Consult Efinix Support on INTERNAL ERROR messages.

## EFX-0501 (ERROR)

**Message** For '<cell name>' number of parameters should be : '<numParams>'.  
**To fix** Consult Efinix Support on INTERNAL ERROR messages.

## EFX-0505 (ERROR)

**Message** Instance '<instance name>' library port '<port name>' is missing. <instance, file: line number>  
**To fix** Instance has an unconnected port. If the primitive is instantiated, check and fix your RTL.

## EFX-0506 (WARNING)

**Message** Set/reset port '<port name>' of instance '<instance name>' is always enabled.  
**To fix** Instance set/reset pin is always enabled. Check your RTL if it is intended.

## EFX-0508 (ERROR)

**Message** Instantiated primitive '<instance name>' has illegal parameter '<name>'. <instance, file: line number>  
**To fix** Instance has an illegal parameter. If the primitive is instantiated, check and fix your RTL.

## EFX-0509 (INTERNAL\_ERROR)

**Message** '<instance name>' instance parameter '<name>' value <value> differs from attribute value <value>  
**To fix** Consult Efinix Support on INTERNAL ERROR messages.

## EFX-0510 (ERROR)

**Message** '<instance name>' instance has missing parameter '<name>' when compared with attribute.  
**To fix** Consult Efinix Support on INTERNAL ERROR messages.

## EFX-0511 (ERROR)

**Message** Path '<path name>' does not exist.  
**To fix** Path for file operation is invalid. Check command line and project settings.

## EFX-0512 (ERROR)

**Message** Pre-synthesis failure is encountered in flow.  
**To fix** Consult Efinix Support on INTERNAL ERROR messages.

## EFX-0513 (ERROR)

**Message** Fail to create path '<path name>'.  
**To fix** TODO

## EFX-0650 (ERROR)

**Message** Input port '<port name>' of '<cell name>' instance '<instance name>' should not be connected to constant.  
**To fix** Pin of the primitive should not be tied to constant. If the primitive is instantiated, check and fix your RTL.

## EFX-0651 (WARNING)

- Message** Instance '<instance name>' port '<port name>' is always disabled. <instance, file: line number>
- To fix** Pin of the primitive is disabled. Check your RTL to see if this is intended.

## EFX-0652 (ERROR)

- Message** '<cell name>' instance '<instance name>' port '<port name>' is not disabled but related register is not used. <instance, file: line number>
- To fix** Primitive pin should be disabled based on configuration. If the primitive is instantiated, check and fix your RTL.

## EFX-0654 (WARNING)

- Message** \*\*\* COMBINATIONAL LOOP detected! Signals contributing to the loop are: <net names> <net, file: line number>
- To fix** Efinix does not recommend using combinational loops. Check your RTL to see if this is intended.

## EFX-0655 (WARNING)

- Message** Mapping into logic memory block '<RAM net name>' (<size> bits). <file: line number>
- To fix** Memory will be implemented in logic. Check your RTL to see if this is intended.

## EFX-0656 (WARNING)

- Message** Optimizing into logic zero initialized read-only memory block '<RAM net name>'. <file: line number>
- To fix** ROM is initialized with zeros and will be optimized. Check your RTL to see if this is intended.

## EFX-0657 (WARNING)

- Message** Mapping into logic memory block '<RAM net name>' (<size> bits) <file: line number> because <reason>
- To fix** Memory will be implemented in logic. Check your RTL to see if this is intended.

## EFX-0658 (ERROR)

- Message** Memory '<instance name>' has disconnected port '<port name>'. <instance, file: line number>
- To fix** TODO

## EFX-0659 (ERROR)

- Message** ROM port '<port name>' is unconnected.
- To fix** Memory instance has an unconnected port. If the primitive is instantiated, check and fix your RTL.

## EFX-0662 (INTERNAL\_ERROR)

- Message** Fail to bit blast RAM net '<RAM net name>'. <file: line number>
- To fix** Consult Efinix Support on INTERNAL ERROR messages.

## EFX-0666 (WARNING)

- Message** Mapping into logic multiplier '*<instance name>*'. *<instance, file: line number>*
- To fix** Multiply operator will be implemented in logic. Check your RTL to see if this is intended.

## EFX-0672 (ERROR)

- Message** Memory '*<net name>*' size *<net size>* exceeds maximum bit-blast threshold *<threshold>*. Please check option `--max-bit-blast-mem-size`. *<net, file: line number>*
- To fix** Memory is too large for bit-blasting. Check your RTL if intended or increase the memory blasting threshold.

## EFX-0674 (INTERNAL\_ERROR)

- Message** Logic synthesis failed.
- To fix** Consult Efinix Support on INTERNAL ERROR messages.

## EFX-0675 (INTERNAL\_ERROR)

- Message** Post synthesis netlist creation failed.
- To fix** Consult Efinix Support on INTERNAL ERROR messages.

## EFX-0676 (WARNING)

- Message** Include path is not specified for RAM net '*<net name>*' hex init file *<file>* path is not resolving. *<net, file: line number>*
- To fix** You should specify the memory initialization file search path in the include directory setting. Check your project settings.

## EFX-0678 (WARNING)

- Message** Failed to read memory block '*<net name>*' hex init file '*<file>*'. *<net, file: line number>*
- To fix** Fail to read memory initialization file. Check the file content or location for correctness.

## EFX-0679 (INTERNAL\_ERROR)

- Message** Memory '*<RAM net name>*' pre-synthesis failure. *<file: line number>*
- To fix** Consult Efinix Support on INTERNAL ERROR messages.

## EFX-0680 (ERROR)

- Message** Dual-port memory '*<RAM net name>*' has incompatible control signal, RE ('*<read enable name>*') should also control WE ('*<write enable name>*'). *<file: line number>*
- To fix** The dual-port memory signal RE should also control WE. See the Quantum Primitives User Guide for details.

## EFX-0681 (INTERNAL\_ERROR)

- Message** Memory '*<RAM net name>*' is invalid for inference but cannot be bit-blasted. *<net, file: line number>*
- To fix** Consult Efinix Support on INTERNAL ERROR messages.

**EFX-0683 (WARNING)**

- Message** RAM net '<RAM net name>' has incompatible control or address signals. <net, file: line number>
- To fix** The memory has incompatible control or address signals. Check your RTL to see if this is intended.

**EFX-0684 (WARNING)**

- Message** RAM net '<RAM net name>' cannot be implemented in block RAM or block ROM. <file: line number>
- To fix** The memory cannot be implemented in BRAM blocks and will be blasted to logic. Check your RTL and the Efinity Synthesis User Guide for memory inference examples.

**EFX-0687 (WARNING)**

- Message** 'syn\_preserve' or 'syn\_keep' attribute on net '<net1>' is transferred to primary I/O net '<net2>'.
- To fix** A 'syn\_preserve' attribute on the net is transferred to a primary I/O port.

**EFX-0689 (WARNING)**

- Message** SRL8 chain started with '<startBit>' is too long, but cannot be broken up.
- To fix** The software cannot break up a long shift register chain, which may cause place and route issues.

**EFX-0691 (WARNING)**

- Message** Carry skip insertion threshold (<cskip\_t>) is too short, expect minimum <min\_cskip\_t>.
- To fix** The carry skip insertion threshold setting is invalid. Check the Efinity Synthesis User Guide for detail.

**EFX-0702 (INFO)**

- Message** Formal equivalence check of spec(pre-map) '<specNetlistName>' and impl(post-map) '<implNetlistName>' netlists PASS status.
- To fix** TODO

**EFX-0809 (ERROR)**

- Message** The periphery cell <cell name> is not supported on the current device.
- To fix** The periphery primitive is not supported in this device. Check your RTL and fix it.

## Post-Synthesis Check Messages

The software may issue the following messages during post-synthesis checking.

**EFX-9001 (ERROR)**

- Message** Output Port <name> is unconnected and not driven.
- To fix** Check the top level port connections.

**EFX-9002 (ERROR)**

**Message** Port <name> is directly connected to <number> other top-level ports.  
**To fix** Check the top level port connections.

**EFX-9003 (ERROR)**

**Message** Output port <name> is connected to <name>/<name>, but only Input or Inout ports can be connected to an EFX\_GBUFCE I port.  
**To fix** If the instance is an instantiated primitive, please check the port connections.

**EFX-9004 (ERROR)**

**Message** Port <name> is connected to <name>/<name>, but must be connected to an EFX\_IOBUF IO port or an EFX\_GBUFCE I port.  
**To fix** If the instance is an instantiated primitive, please check the port connections.

**EFX-9005 (ERROR)**

**Message** Port <name> is connected to <number> EFX\_GBUFCE instances.  
**To fix** If the instance is an instantiated primitive, please check the port connections.

**EFX-9006 (ERROR)**

**Message** Port <name> is connected to <number> EFX\_IOBUF instances.  
**To fix** If the instance is an instantiated primitive, please check the port connections.

**EFX-9007 (ERROR)**

**Message** Unexpected constant net <name>.  
**To fix** Consult Efinix Support on this ERROR message.

**EFX-9008 (ERROR)**

**Message** Post-synthesis netlist is not legal because it has 0 total nets.  
**To fix** Consult Efinix Support on this ERROR message.

**EFX-9009 (ERROR)**

**Message** Post-synthesis netlist has only vcc and gnd nets because all logic is eliminated. Legal, but will cause router to crash.  
**To fix** The design logic may have been optimized away by synthesis. Check your design for unintended connections or unused outputs.

**EFX-9010 (ERROR)**

**Message** Post-synthesis netlist is not legal because it has <number> GND nets.  
**To fix** Consult Efinix Support on this ERROR message.

**EFX-9011 (ERROR)**

**Message** Post-synthesis netlist is not legal because it has <number> VCC nets.  
**To fix** Consult Efinix Support on this ERROR message.

## EFX-9012 (ERROR)

**Message** Net <name> is not driven.  
**To fix** Consult Efinix Support on this ERROR message.

## EFX-9013 (ERROR)

**Message** Net <name> has multiple drivers.  
**To fix** Consult Efinix Support on this ERROR message.

## EFX-9014 (ERROR)

**Message** Net <name> is driving multiple clock buffers. <name>  
**To fix** If the instance is an instantiated primitive, please check the port connections.

## EFX-9015 (ERROR)

**Message** Clock net <name> drives clock ports and clock buffer <name>.  
**To fix** If the instance is an instantiated primitive, please check the port connections.

## EFX-9017 (ERROR)

**Message** <name> is missing port <name>.  
**To fix** If the instance is an instantiated primitive, please check the port connections.

## EFX-9018 (ERROR)

**Message** <name> is missing portbus <name>.  
**To fix** If the instance is an instantiated primitive, please check the port connections.

## EFX-9019 (ERROR)

**Message** <name> portbus of <name> instance <name> is size <number> but <number> expected.  
**To fix** If the instance is an instantiated primitive, please check the port connections.

## EFX-9020 (ERROR)

**Message** <name> port of <name> instance <name> is disconnected.  
**To fix** If the instance is an instantiated primitive, please check the port connections.

## EFX-9021 (ERROR)

**Message** <name> port of <name> instance <name> is constant.  
**To fix** If the instance is an instantiated primitive, please check the port connections.

## EFX-9022 (ERROR)

**Message** <name> port of <name> instance <name> is not driven by an EFX\_GBUFCE/O port.  
**To fix** If the instance is an instantiated primitive, please check the port connections.

## EFX-9024 (ERROR)

- Message** <name> instance <name> has different read/write address bit <number> in WRITE\_FIRST WRITE\_MODE.
- To fix** If the instance is an instantiated primitive, please check the port connections.

## EFX-9025 (ERROR)

- Message** <name> instance <name> <name> port is disabled, but <name> is not.
- To fix** If the instance is an instantiated primitive, please check the port connections.

## EFX-9026 (ERROR)

- Message** <name> instance <name> <name> port is not disabled in a disabled mode.
- To fix** If the instance is an instantiated primitive, please check the port connections.

## EFX-9027 (ERROR)

- Message** <name> instance <name> input <name> is active, but unused. Check parameter settings.
- To fix** If the instance is an instantiated primitive, please check the correct parameter value and range.

## EFX-9028 (ERROR)

- Message** <name> instance <name> has different read/write address enable in WRITE\_FIRST WRITE\_MODE.
- To fix** If the instance is an instantiated primitive, please check the correct parameter value and range.

## EFX-9029 (ERROR)

- Message** <name> instance <name> with WRITE\_MODE = <name> has different read/write clock nets/polarity.
- To fix** If the instance is an instantiated primitive, please check the correct parameter value and range.

## EFX-9030 (ERROR)

- Message** <name> parameter of <name> instance <name> is not valid.
- To fix** If the instance is an instantiated primitive, please check the correct parameter value and range.

## EFX-9031 (ERROR)

- Message** <name> instance <name> <name> read width <number> is not compatible with write width <number>.
- To fix** If the instance is an instantiated primitive, please check the port connections.

## EFX-9032 (ERROR)

- Message** <name> instance <name> Port A read/write widths <number>/<number> are not compatible with Port B <number>/<number>.
- To fix** If the instance is an instantiated primitive, please check the port connections.

## EFX-9033 (ERROR)

**Message** <name> instance <name> both CLKA port and CLKB ports are disabled.  
**To fix** If the instance is an instantiated primitive, please check the port connections.

## EFX-9034 (ERROR)

**Message** <name> instance <name> WE[1] port is not disabled, but WRITE\_WIDTH is <number>.  
**To fix** If the instance is an instantiated primitive, please check the port connections.

## EFX-9035 (ERROR)

**Message** <name> instance <name> WRITE\_MODE is WRITE\_FIRST but READ\_WIDTH is <number> and WRITE\_WIDTH is <number>.  
**To fix** If the instance is an instantiated primitive, please check the correct parameter value and range.

## EFX-9036 (ERROR)

**Message** <name> instance <name> WRITE\_MODE is WRITE\_FIRST but WCLKE is not permanently enabled.  
**To fix** If the instance is an instantiated primitive, please check the correct parameter value and range.

## EFX-9037 (ERROR)

**Message** <name> instance <name> has parameter SR\_SYNC\_PRIORITY=<number>, but the device only supports a value of <number>.  
**To fix** If the instance is an instantiated primitive, please check the correct parameter value and range.

## EFX-9038 (ERROR)

**Message** <name> instance <name> CLK port is not disabled but no registers are enabled.  
**To fix** If the instance is an instantiated primitive, please check the correct parameter value and range.

## EFX-9039 (ERROR)

**Message** <name> instance <name> SHIFT\_ENA port is not disabled, but SHIFTER parameter is 0.  
**To fix** If the instance is an instantiated primitive, please check the correct parameter value and range.

## EFX-9040 (ERROR)

**Message** <name> instance <name> <name> must be enabled when <name>.  
**To fix** If the instance is an instantiated primitive, please check the correct parameter value and range.

## EFX-9041 (ERROR)

**Message** <name> instance <name> <name> must be <name> when <name>.  
**To fix** If the instance is an instantiated primitive, please check the correct parameter value and range.

## EFX-9042 (ERROR)

- Message**            <name> instance <name> MODE '<name>' is illegal.
- To fix**              If the instance is an instantiated primitive, please check the correct parameter value and range.

## EFX-9043 (ERROR)

- Message**            <name> instance <name> W\_REG may not be enabled when W\_SEL is P.
- To fix**              If the instance is an instantiated primitive, please check the correct parameter value and range.

## EFX-9044 (ERROR)

- Message**            <name> instance <name> CASCIN[<number>] port must be connected to GND or CASCOUT.
- To fix**              If the instance is an instantiated primitive, please check the port connections.

## EFX-9045 (ERROR)

- Message**            <name> instance <name> CASCOUT[<number>] port must be disconnected or connected to CASCIN.
- To fix**              If the instance is an instantiated primitive, please check the port connections.

## EFX-9046 (ERROR)

- Message**            <name> instance <name> CASCIN port must be connected to GND or CASCOUT.
- To fix**              If the instance is an instantiated primitive, please check the port connections.

## EFX-9047 (ERROR)

- Message**            <name> instance <name> CASCOUT port must be disconnected or connected to CASCIN.
- To fix**              If the instance is an instantiated primitive, please check the port connections.

## EFX-9048 (ERROR)

- Message**            <name> instance <name> W\_SEL cannot be P if CASCOUT\_SEL is W.
- To fix**              If the instance is an instantiated primitive, please check the correct parameter value and range.

## EFX-9049 (ERROR)

- Message**            <name> instance <name> CI port must be connected to GND or CO.
- To fix**              If the instance is an instantiated primitive, please check the port connections.

## EFX-9050 (ERROR)

- Message**            <name> instance <name> CO port must be disconnected or connected to CI.
- To fix**              If the instance is an instantiated primitive, please check the port connections.

## EFX-9051 (ERROR)

- Message**            <name> instance <name> <name> port must not be connected in <name> mode.
- To fix**              If the instance is an instantiated primitive, please check the port connections.

## EFX-9052 (ERROR)

**Message** <name> instance <name> unexpected MODE '<name>'.  
**To fix** If the instance is an instantiated primitive, please check the correct parameter value and range.

## EFX-9053 (ERROR)

**Message** <name> instance <name> CO port must be connected to only 1 CIN.  
**To fix** If the instance is an instantiated primitive, please check the port connections.

## EFX-9054 (ERROR)

**Message** <name> instance <name> is driven by net <name> more than once.  
**To fix** If the instance is an instantiated primitive, please check the port connections.

## EFX-9055 (ERROR)

**Message** <name> instance <name> LUTMASK %04X is too large to fit into a 4-input LUT.  
**To fix** If the instance is an instantiated primitive, please check the correct parameter value and range.

## EFX-9056 (ERROR)

**Message** <name> instance <name> LUTMASK %04X does not depend on connected input <number>.  
**To fix** If the instance is an instantiated primitive, please check the port connections.

## EFX-9057 (ERROR)

**Message** <name> instance <name> LUTMASK %04X does depends on unconnected input <number>.  
**To fix** If the instance is an instantiated primitive, please check the port connections.

## EFX-9058 (ERROR)

**Message** <name> instance <name> Q7 port must be disconnected or connected to only an SRL8 D port.  
**To fix** If the instance is an instantiated primitive, please check the port connections.

## EFX-9059 (ERROR)

**Message** I port of <name> instance <name> is constant.  
**To fix** If the instance is an instantiated primitive, please check the port connections.

## EFX-9060 (ERROR)

**Message** Clock net <name> is driving the <name> port of block <name>, it must only drive clock ports.  
**To fix** The GBUFCE output cannot drive a primitive's non-clock port. However, the input can drive it. Use a different signal or drive the non-clock port with the same value as the input to the GBUFCE (which uses regular routing).

## EFX-9062 (ERROR)

**Message** <name> instance <name> input OP is active, but unused (W\_SEL is not set to X). Check parameter settings.

**To fix** If the instance is an instantiated primitive, please check the correct parameter value and range.

## EFX-8001 (WARNING)

**Message** <name> port of <name> instance <name> is permanently enabled

**To fix** If the instance is an instantiated primitive, please check the port connections.

## EFX-8002 (WARNING)

**Message** <name> port of <name> instance <name> is permanently disabled

**To fix** If the instance is an instantiated primitive, please check the port connections.

## EFX-8003 (WARNING)

**Message** Input/inout port <name> is unconnected and will be removed.

**To fix** Check if port is intentionally unconnected.

## EFX-8004 (WARNING)

**Message** <name> port of <name> instance <name> is connected in a disabled mode.

**To fix** If the instance is an instantiated primitive, please check the port connections.

## EFX-8005 (WARNING)

**Message** <name> instance <name> is using deprecated WIDTH parameter <number>.

**To fix** If the instance is an instantiated primitive, please check the correct parameter value and range.

## EFX-8009 (WARNING)

**Message** Net <name> is dangling.

**To fix** The net may be dangling due to 'syn-keep' or 'syn-preserve' property.

## Where to Learn More

The Efinity® software includes documentation as PDF user guides and on-line HTML help. This documentation is provided with the software. You can also access the latest versions of PDF documentation in the [Support Center](#):

- [Efinity Software User Guide](#)
- [Efinity Software Installation User Guide](#)
- [Efinity Synthesis User Guide](#)
- [Efinity Timing Closure User Guide](#)
- [Efinity Trion Tutorial](#)
- [Efinity Debugger Tutorial](#)
- [Efinity Programmer User Guide](#)
- [Efinity IP Packager User Guide](#)
- [Trion Interfaces User Guide](#)
- [Titanium Interfaces User Guide](#)
- [Topaz Interfaces User Guide](#)
- [Efinity Interface Designer Python API](#)
- [Efinity Command-Line Interface User Guide](#)
- [Quantum® Trion Primitives User Guide](#)
- [Quantum® Titanium Primitives User Guide](#)
- [Quantum® Topaz Primitives User Guide](#)

In addition to documentation, Efinity field application engineers have created a series of videos to help you learn about aspects of the software. You can view these videos in the [Support Center](#).

# Revision History

**Table 8: Revision History**

Date	Version	Description
November 2025	4.4	Added information on how to use out-of-context (OOO) synthesis flow. (DOC-2761) Added <b>Hierarchical Names Support</b> on page 54. (DOC-2722)
August 2025	4.3	Added warning note to <b>Inferring Wide Multipliers</b> on page 9. (DOC-2643) Added --suppress_info_msgs, --suppress_warning_msgs, and --msg_suppression_list place and route options. (DOC-2599)
May 2025	4.2	Added mark_debug attribute in Synthesis Attributes. (DOC-2344) Added --enable_mark_debug synthesis option.
February 2025	4.1	Added table of supported SystemVerilog constructs. (DOC-2342) Updated synthesis options. (DOC-2339)
November 2024	4.0	Added additional example for inferring DSP. (DOC-1623) Added error message EFX-9060 to <b>Post-Synthesis Check Messages</b> on page 71. Added --max_threads synthesis option. (DOC-2051) Added --peri-syn-inference and --peri-syn-instantiation synthesis options and syn_peri_port synthesis attribute to support the unified netlist flow. (DOC-2086)
June 2024	3.9	Added description of <b>VHDL-2019</b> support for interfaces. Updated description for --mult-auto-pipeline option. (DOC-1880) Added new advanced option for --retiming. (DOC-1880) Added <b>Warning and Error Messages</b> on page 57.
December 2023	3.8	Added the --use-logic-for-small-mem, --use-logic-for-small-rom, and --mult-auto-pipeline synthesis options. (DOC-1484) Corrected quotation marks in the synthesis attribute examples (now using straight quotes instead of curly). (DOC-1420) Added topic ( <b>Referencing Efinix VHDL Libraries</b> on page 43) describing which Efinix VHDL libraries to reference when using Efinix primitives. (DOC-1455)
June 2023	3.7	Added syn_keep synthesis attribute. Added --mult-decomp-ptime, --optimize-zero-init-rom, and --insert-carry-skip synthesis options. Provided an example for the --allow-const-ram-index synthesis option.
May 2023	3.6	Added description about possibility of bit-blasting certain RAM operations but at the cost of FPGA resource. (DOC-1231)
December 2022	3.5	Described synthesis rules for inferring output registers. (DOC-1020) Updated the synthesis options. (DOC-1020) The syn_preserve attribute is not supported on a user hierarchy instance. (DOC-1020) Added synthesis pragmas. (DOC-1020)
November 2022	3.4	Added note in "Using VHDL Libraries" about the work library. (DOC-957)
August 2022	3.3	Added new synthesis options. (DOC-870)
December 2021	3.2	Added new synthesis options. Added more detail about the Synthesis project settings. (SYN-549)

Date	Version	Description
October 2021	3.1	The default for the <code>-blast_const_operand_adders</code> and <code>-seq_opt</code> synthesis options is 1 for Efinity v2021.1 and higher. (DOC-481)
June 2021	3.0	<p>Updated for Efinity software v2021.1.</p> <p>Added support for Titanium FPGAs.</p> <p>Described how to infer Titanium DSP Blocks, shift registers, and RAM.</p> <p>Added Titanium synthesis options.</p> <p>Added topic on retiming.</p> <p>Added the <code>syn_srlstyle</code> attribute.</p> <p>Added the <code>area2</code> value for the mode synthesis option.</p>
January 2021	2.1	<p>Added information on RAM inferencing.</p> <p>Described the Block RAM Resource Estimator.</p>
December 2020	2.0	<p>Described new support for VHDL libraries.</p> <p>Added <code>async_reg</code>, <code>skip_ram_init</code>, <code>syn_srlstyle</code>, <code>syn_ramdecomp</code>, and <code>syn_srlstyl</code> synthesis attributes.</p>
June 2020	1.0	Initial release.