Efinity® Synthesis User Guide

UG-EFN-SYNTH-v1.0
June 2020
www.efinixinc.com

Contents

Introduction

The Efinity® 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® 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® 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

Specifying Language Support

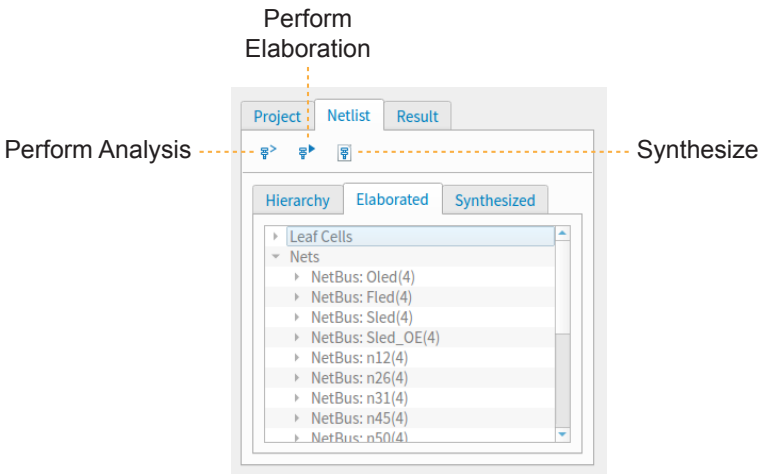
You can set the language support at the project level or at the file level. In both cases, you make this setting in the **Design** tab of the Project Editor dialog box. Open the Project Editor by choosing **File > Edit Project** or by clicking the toolbar button.

Verilog HDL Choices	VHDL Choices
verilog_2k	vhdl_2008
verilog_95	vhdl_1993
SystemVerilog2005	
SystemVerilog2009	

Netlist Pane

The Netlist pane, 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.

Figure 1: Using the Netlist Pane



Design Guidelines

The following sections provide some general design guidelines.

Flip-Flops

The Efinity® 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 2: EFX_FF Symbol

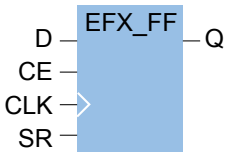


Table 1: EFX_FF Ports

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. Additionally, register stages in block RAM and multipliers are synchronous only.
- Avoid flip-flops with both a set and a reset signal. The Trion® 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® FPGAs do not support latches natively in hardware. The Efinity® 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.

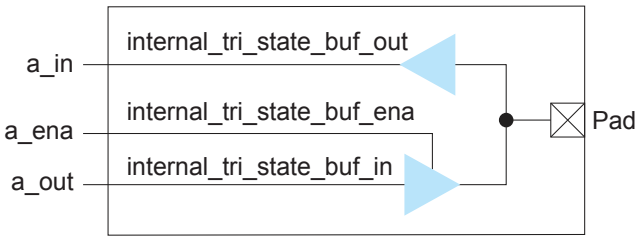
Tri-State Buffers

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® 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® 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 3: Tri-State Buffer



To target Trion® 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
```

Attention: 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.

Table 2: Synthesis Options

Name	Choices	Description
--mode	speed, area	speed: Optimizes for fastest f_{MAX} (default). area: Optimizes for smallest area.
--max_ram	-1, 0, *n*	-1: Default. There is no limit to the number of RAM blocks to infer. 0: Disable. *n*: Any integer.
--max_mult	-1, 0, *n*	-1: Default. There is no limit to the number of multipliers to infer. 0: Disable. *n*: Any integer.
--infer-clk-enable	0, 1, 2, 3	Infer flip-flop clock enables from control logic. 0: disable. 1: Low effort. 2: Medium effort. 3: Default. High effort.
--infer-sync-set-reset	0, 1	Infer synchronous set/reset signals. 0: Disable. 1: Default. Enable.
--fanout-limit	0 to *n*	If something is high fanout, the tool duplicates the fanout source. 0: Default. Disable. *n*: Indicate the fanout limit at which to begin duplication.
--seq_opt	0, 1	Turn on sequential optimization. This option can reduce LUT usage but may impact f_{MAX}. 0: Default. Disable. 1: Enable.

Infer Clock 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.

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.

syn_extract_enable	syn_preserve	syn_ramstyle
syn_romstyle	translate_on translate_off	syn_use_dsp

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_preserve

You use this attribute on signals. When it is set to `true`, `yes`, or `1`, the software keeps the signal through optimization, that is, the software does not minimize or remove the signal. This attribute can be helpful when you want to simulate or view a signal in the Debugger.

Verilog HDL:

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

VHDL:

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

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";
```

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 FPG'A'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
```

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.

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;
```

VHDL 2008 Support

The Efinity® software supports the synthesis subset of the VHDL 2008 standard. The following sections highlight support for new features from an Efinity® 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" =>
                     comp1: entity work.test1(behave)
                            port map(B(13),A(13),C(27));
                  when "01" =>
                     comp1: entity work.test2(behave)
                            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(behave)
        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 extention
                   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 commnents 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
```

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 Synthesis User Guide**
- **Efinity Timing Closure User Guide**
- **Efinity Software Installation User Guide**
- **Efinity Trion Tutorial**
- **Trion Interfaces User Guide**
- **Efinity Interface Designer Python API**
- **Efinity Software Quantum Primitives User Guide**

In addition to documentation, Efinix 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 3: Revision History

Date	Version	Description
June 2020	1.0	Initial release.