Synopsys®, Inc.
690 East Middlefield Road
Mountain View, CA 94043 USA
Website: www.synopsys.com

Synplify Pro® for Microsemi Edition Release Notes

Version L-2016.09M-G5, November 2016

Publication Version 01

Release Note Topics

About the Release

This L-2016.09M-G5 release includes software features and enhancements for the Synplify Pro® Microsemi Edition product. For the complete summary of features and enhancements contained in this release and the previous releases, see Feature and Enhancement Summary below.

Feature and Enhancement Summary

The following table highlights the L-2016.09M-G5 features:

Feature	Description
RAM Primitives Support	RAM1K20 and RAM64x12 are supported for PolarFire. See RAM Primitives Support, on page 4.
DSP Block Inference Support	MACC_PA inference is supported for PolarFire. See DSP Block Inference, on page 5.
Wide MUX inference Support	Wide MUXs are implemented using ARI1 primitives.
Support for URAM Inference for Sequential Shift Registers	Synplify Pro is enhanced to support URAM inference for sequential shift registers. By default, seqshift is implemented using URAM. The syn_srlstyle attribute is used to override the default behavior of seqshift implementation. This support is available for the PolarFire technology only. See URAM Inference for Sequential Shift Registers, on page 5.
SLE Enhancement	Synplify Pro supports the packing of enable signal with higher priority than the reset signal (synchronous) into SLE. The tool is enhanced to support the packing of synchronous reset signal into SLE using the SLn pin, when the fanout limit >= 6.
PolarFire RAM1K20 and RAM64x12 Enhancements	The tool is enhanced to support the packing of enable signal on the read address register into RAM1K20 (A_REN) and RAM64x12 (R_ADDR_EN).
Compiler Enhancements	Compiler enhancements include the following: SystemVerilog support for the `begin_keywords and `end_keywords directives. Specifies a pair of directives—`begin_keywords and `end_keywords—to identify keywords reserved within a block of source code, based on a specific version of IEEE Std 1364 or IEEE Std 1800. *Language Support Reference->Verilog Language Support->Support for Verilog Language Constructs->Compiler Directives* Use the new SYN_COMPATIBLE=DC macro to ensure compatibility of Synopsys tools such as Design Compiler (DC) with the synthesis software. *Command Reference->User Interface Commands->Implementation Options Command->Compiler Directives and Design Parameters*

New HDL Analyst® Tool	*Beta* A new version of the next-generation schematic analysis tool is enabled by default. To go back to the original HDL Analyst tool, click the button in the upper right of the tool window or deselect the option HDL Analyst->Use New HDL Analyst (Beta). This version includes usability improvements, better performance and support for designs that generate large netlists. *User Guide->Analyzing with HDL Analyst->Working in the Schematic (Beta)* *Beta* The HDL Analyst also uses new Tcl and find commands. *Command Reference->Tcl Commands->analyst* and *Tcl Commands->design*
Launch an Independent Help	Launch the help system independently of the tool, by running *installDirectory*/bin/fpga_help.exe. It is recommended that you use help instead of PDFs, because help is designed as an integrated system and includes additional navigational aids. You can double-click on the executable to start it on Windows.
Identify Features	
Identify Graphical User Interface Changes	Minor changes to the graphical user interface include: • The RTL Instrumentor status panel is rearranged and renamed to Control Panel. • The Instrumentor Search dialog box is replaced with the Search panel in the main view, and the Search icon has been removed. The *Identify Instrumentor User Guide* has been updated to reflect these changes.
Identify Debugger Stand-alone Installation Package	The Identify debugger executable is packaged and installed separately.
Identify Device Support	See *Identify Tool Device Support* on page 7.

Feature and Enhancement Descriptions

This section contains a summary of the features and enhancements for this release.

RAM Primitives Support

The tool is enhanced to support RAM primitives in the PolarFire device:

- RAM1K20 (LSRAM) is supported for both inference and instantiation.

  The following configurations are supported for inference:

  – True dual-port configuration

  – Two independent data ports

  – Non-ECC—1Kx20, 2Kx10, 4Kx5, 8Kx2 or 16Kx1 on each port

  – Two-port configuration

  – Read from port A and write to port B

  – Non-ECC—512x40, 1Kx20, 2Kx10, 4Kx5, 8Kx2 or 16Kx1 on each port

  – ECC—512x33 on both ports

  Generates SB_CORRECT and DB_DETECT flags

  – Write operations

  – 3 modes—simple write, write feed-through, read before write

  – Limitations

  – Asymmetric RAM is not supported in this release

  – RAM initialization is not supported in this release

- RAM64x12 (USRAM) is supported for both inference and instantiation.

  The following configurations are supported for inference:

  – The RAM64x12 block contains 768 memory bits and is a two-port memory providing one write port and one read port. Write operations to the RAM64x12 memory are synchronous. Read operations can be asynchronous or synchronous for setting up the address and reading out the data.

  – There is one read-data port and one write-data port.

  – Both read-data and write-data ports are configured to 64x12.

DSP Block Inference

The DSP block inference feature allows the synthesis tool to infer MACC_PA block for PolarFire devices. The following structures are supported:

- Add-mult—adder followed by a multiplier
- Multipliers
- Mult-adds —multiplier followed by an adder
- Mult-subs —multiplier followed by a subtractor
- Mult-acc—multiplier followed by an accumulator
- Wide multiplier inference
- MATH block inferencing across hierarchy
- DOTP Support

  MACC_PA block supports DOTP mode when:

  – Width of the multiplier inputs is less than or equal to 9-bits for signed multiplication.
  – Width of the multiplier inputs is less than or equal to 8-bits for unsigned multiplication.

By default, the synthesis software maps the multiplier to DSP blocks if all inputs to the multiplier are more than 2-bits wide. Otherwise, the multiplier is mapped to logic. You can override this default behavior using the syn_multstyle attribute.

URAM Inference for Sequential Shift Registers

By default, seqshift is implemented using URAM. The syn_srlstyle attribute is used to override the default behavior of seqshift implementation.

syn_srlstyle Values

Value	Description
Registers	seqshifts are inferred as registers
URAM	seqshift is inferred as RAM64x12

syn_srlstyle Syntax

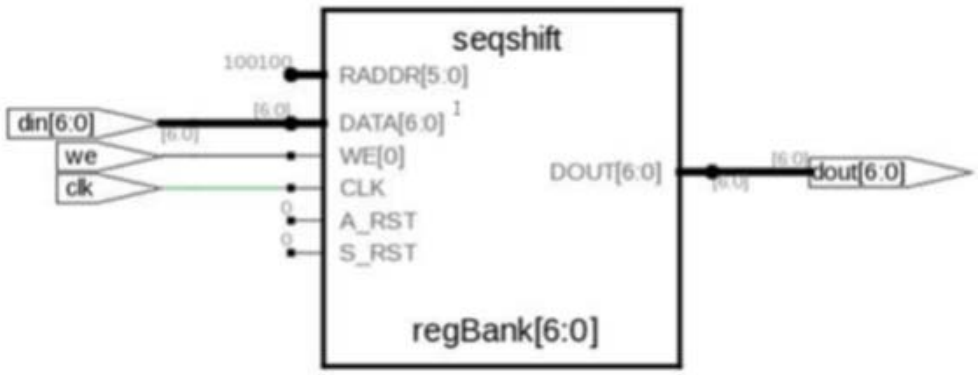
FDC	define_attribute {object} syn_srlstyle {registers|uram } define_global_attribute syn_srlstyle {registers|uram }
Verilog	object /* synthesis syn_srlstyle = "registers | uram " */ ;
VHDL	attribute syn_srlstyle : string; attribute syn_srlstyle of object : signal is "registers | uram ";

Example

The tool infers a seqshift primitive for the given RTL:

```
module p_seqshift(clk, we, din, dout) ;
parameter SRL_WIDTH = 7;
parameter SRL_DEPTH = 37;
input clk, we;
input [SRL_WIDTH-1:0] din;
output [SRL_WIDTH-1:0] dout;
reg [SRL_WIDTH-1:0] regBank[SRL_DEPTH-1:0]/*synthesis srlstyle = "uram"*/ ;
integer i;
always @(posedge clk) begin
if (we) begin
for (i=SRL_DEPTH-1; i>0; i=i-1) begin
regBank[i] <= regBank[i-1];
end
regBank[0] <= din;
end
end
assign dout = regBank[SRL_DEPTH-1];
endmodule
```

The following graphic displays seqshift for the above RTL in the technology view.



Limitations

- URAM inference for seqshift is not supported, if the output is taken from a dynamic stage.
- Seqshifts with synchronous reset or asynchronous reset are inferred as registers.
- Seqshifts with both synchronous reset and asynchronous reset are inferred as registers.
- Seqshifts with both reset and set are inferred as registers.
- Seqshifts with enable signal having higher priority than synchronous set or synchronous reset are inferred as registers.

Identify Tool Device Support

The Identify tool supports the device families shown in the table below. You must select devices from the synthesis tool, which get passed to the Identify Instrumentor in the synthesis project file. If you specify a library from the synthesis tool that is not supported in the Identify tool, then this results in a *device not supported* message when launching the Identify Instrumentor.

Microsemi
Fusion
IGLOO
IGLOOe
IGLOO PLUS
IGLOO2
ProASIC
ProASIC3
ProASIC3E
ProASIC3L
SmartFusion
SmartFusion2

Recommended Versions of Compatible Tools

The FPGA design tools are tested with specific versions of other compatible Synopsys and third-party tools. The recommended versions of these tools are listed below.

Compatible Versions of Synopsys Tools

The table lists the recommended version for VCS:

Tool	Recommended Version
VCS	L-2016.06

Platforms

This section includes platform support for the Synopsys FPGA synthesis product. The software is supported on the platforms and operating systems listed below:

Windows	• Windows 10 Professional or Enterprise (64-bit) • Windows 8.1 Professional or Enterprise (64-bit) • Windows 7 Professional or Enterprise (32 or 64-bit)[1] • Windows Server 2008 R2 (64-bit) • Windows Server 2012 R2 (64-bit)
Linux	All Linux platforms require 32-bit compatible libraries. • Red Hat Enterprise Linux 5[2]/6/7 (64-bit) • SUSE Linux Enterprise 11/12 (64-bit)

1. Support for Windows 7 32-bit will be discontinued as of release L-2016.09-SP1. Therefore, version L-2016.09 is the last release for which Windows 7 32-bit will be supported.
2. Support for Linux Red Hat Enterprise 5 64-bit will be discontinued after release L-2016.09-SP1.

Documentation

The following documents are included with the Synopsys FPGA synthesis product.

Document	Access
User Guide	Online help, PDF
Reference Manual	Online help, PDF
Attribute Reference Manual	Online help, PDF
Command Reference Manual	Online help, PDF
Language Support Reference Manual	Online help, PDF
Messages Reference Manual	Online help
Identify Instrumentor User Guide	Online help, PDF
Identify Debugger User Guide	Online help, PDF
Identify Debugging Environment Reference Manual	Online help, PDF

Known Problems and Solutions

The current known problems in the tool include the *Identify Tool Known Problems and Solutions* on page 9.

Identify Tool Known Problems and Solutions

The following problems are specific to the Identify instrumentor and Identify debugger tools.

Incremental Instrumentation from Previous Release Cannot be Used

Attempting to open an incremental instrumentation created from a previous Identify release results in an assertion error.

Solution: The incremental instrumentation from the previous release cannot be used, and a new instrumentation must be redefined using this new release.

Unable to Launch the GDKWave Viewer from the Debugger

Occasionally when launching the GTKWave viewer from Linux, the viewer fails to open with the following message:

ERROR: couldn't execute "*installPath*/bin/gtkwave/gtkwave": no such file or directory

Solution: Restart both the Synplify tool and the debugger. This problem is scheduled to be addressed in a future release.

Instrumentor Can Become Unresponsive While Using Multiple Implementations

When using multiple implementations, selecting a different instrumentation from the Instrumentations Select field at the bottom left of the RTL instrumentor view can cause the instrumentor to become unresponsive.

Solution: Select the desired implementation from the project view in the synthesis tool, then invoke the instrumentor rather than attempting to select a different instrumentation directly from the Instrumentations Select field.

Issue Running Identify Instrumentor and Debugger on Different Platforms

When using real-time debugging with the Identify instrumentor running on a Linux platform and the Identify debugger running on a Windows platform, an error is reported when scanning the logic analyzer stating that the remote copy (RCP) could not be executed.

Solution: Run the Identify debugger from the Linux platform.

Pod Assignments not Displayed After Execution of the Logic Analyzer Command

When using real-time debugging, the iice assignments report command fails to display any pod assignments after successful execution of the assignpod and submit options of the logicanalyzer command.

Solution: This problem is scheduled to be addressed in a future release.

Trigger Position May be Incorrect for Data Compression with Cross-Triggering

When using data compression with cross-triggering enabled, the trigger position is incorrect for both the internal_memory (BRAM) and hapssram (SRAM) buffer type settings (the data is still valid).

Solution: This problem is scheduled to be addressed in a future release.

Limitations

The current limitations in the tool are divided into the following categories:

FPGA Synthesis Limitations

The following limitations apply to supported features in the Synplify Pro product.

GUI Processing Can Fail on Windows 7 for the Synthesis Tool

The synthesis tool GUI might intermittently stop responding on Windows 7.

Solution: To resolve this issue, apply the hotfix from Microsoft by going to suppport.microsoft.com/kb/2718841/.

Crossprobing Source Code Files Created with Third-Party Editors

When using source code files created with third-party editors, you sometimes cannot crossprobe to the correct line number in the source file.

Solution: Open the file in the FPGA synthesis tool text editor.

Editing Externally Created Project (prj) Files

If Tcl commands or script files were used to build your project, you might not be able to save this Project file from the synthesis GUI in downstream tools, because they contain hard-coded file paths.

Solution: Generally, use the same method to save a project as you did to create the project. In this case, save the project file to an external text editor and not in the project GUI.

Identify Tool Limitations

The following limitations are specific to the Identify instrumentor and Identify debugger tools.

Verilog/SystemVerilog Limitations When Importing Signals from Verdi

The following Verilog/SystemVerilog language limitations are present when importing signals directly from the Verdi® platform:

- Enums with syn_enum_encoding attribute are not supported for instrumentation and, if present, can impact data expansion.
- Trigger expression settings for unions are either in the form of a serialized bit vector or hex/integer with the trigger bit width representing the maximum available bit width among all union members. Trigger expressions using enum are not possible.
- Generate statements are not supported.

- A limitation exists in the instrumentation of essential signals generated by the Verdi platform because of the naming convention used to represent certain essential signals by the Verdi tool. Instrumentation of such signals cannot be performed automatically using the Verdi getsignals command. The Identify instrumentor issues a warning message when these type of signals are encountered.
- Instrumentation of Interfaces is not supported.

VHDL Limitations When Importing Signals from Verdi

The following VHDL language limitations are present when importing signals directly from the Verdi platform:

- Boolean vector representation in the Identify-generated FSDB is different from the VCS generated FSDB, but does not have any known impact during the data expansion.
- Record elements are represented in reverse order in the Identify-generated FSDB. This reversal does not have any known impact during data expansion.
- Generate statements are not supported.

External Triggering with Imported Triggers Can Cause Excessive Use of the Internal Block RAM

Use of external triggers via the import trigger mechanism causes an excessive use of internal block RAM due to sampling of the trigger as well as the creation of a look-up table. The problem is most notable when the maximum of eight imported triggers is selected.

Solution: Add an extra input to the top-level RTL code and instrument the input as a trigger only.

Synopsys, Inc.
690 East Middlefield Road
Mountain View, CA 94043 USA