Synthesis User Guide (UG018)

All Achronix Devices



Copyrights, Trademarks and Disclaimers

Copyright © 2025 Achronix Semiconductor Corporation. All rights reserved. Achronix, Speedcore, Speedster, and ACE are trademarks of Achronix Semiconductor Corporation in the U.S. and/or other countries All other trademarks are the property of their respective owners. All specifications subject to change without notice.

Notice of Disclaimer

The information given in this document is believed to be accurate and reliable. However, Achronix Semiconductor Corporation does not give any representations or warranties as to the completeness or accuracy of such information and shall have no liability for the use of the information contained herein. Achronix Semiconductor Corporation reserves the right to make changes to this document and the information contained herein at any time and without notice. All Achronix trademarks, registered trademarks, disclaimers and patents are listed at http://www.achronix.com/legal.

Achronix Semiconductor Corporation

2903 Bunker Hill Lane Santa Clara, CA 95054 USA

Website: www.achronix.com E-mail : info@achronix.com

Table of Contents

Chapter 1 :	Overview	2
	Synthesis Flows	2
Chapter 2 :	ACE-Driven Integrated Synthesis	3
	Synthesis Project Setup in ACE	3
	Create an ACE Project	3
	Add the Design Files and Set Project Options	4
	Managing Memory Initialization Files in ACE	7
	Synthesis Options Configuration	9
	Running Synthesis to Compile the Design	10
	Synthesis Reports and Messages	
	Opening Synplify Project File in Synplify Pro	14
Chapter 3 :	Synplify-Pro-Driven Integrated Synthesis	16
	Configuring the Synthesis Project in Synplify Pro	16
	Synthesis Project Setup in ACE	16
	Create an ACE Project	
	Add the Design Files and Set Project Options	
	Managing Memory Initialization Files in ACE	20
	Synthesis Options Configuration	21

	Running Synthesis to Compile the Design	22
	Synthesis Reports and Messages	24
	Opening Synplify Project File in Synplify Pro	26
Chapter 4 :	Stand-Alone Synthesis in Synplify Pro	. 27
	Configuring the Synthesis Project in Synplify Pro	27
	Running Synthesis	27
	Adding the Synthesized Netlist to ACE for Place and Route	27
Chapter 5 :	Synthesis Integration with Multiprocess Option Exploration	.29
Chapter 6 :	Managing Projects in Synplify Pro	. 31
	Creating and Setting up a Project	31
	Adding the Synthesis Library Include File	33
	Adding Source Files to the Project	34
	Implementation Options	35
	Verilog	35
	Place and Route	36
	Timing Report	36
	Implementation Results	
	Constraints	
	Options	
Chapter 7 :	Synplify Pro Features	. 41
	Synplify Warnings	41

	Synthesis Hierarchical Report	
	Hierarchical Area Report	
	HDL Analyst Schematics	
	Watch Window	
	Validating Constraints	
	Using Help	
Chapter 8 :	Synthesis Constraints	48
	Timing Constraints	
	create_clock	
	Syntax	
	Command Examples	
	create_generated_clock	
	Syntax	
	Command Examples	
	set_clock_groups	
	Syntax	
	Command Example	
	set_false_path	
	Syntax	
	Command Examples	
	set_input_delay	51
	Syntax	
	Command Examples	
	set_output_delay	
	Syntax	

Chapter

	Command Examples	52
	set_max_delay	
	Syntax	52
	Command Examples	52
	set_multicycle_path	
	Syntax	53
	Command Examples	53
	set_clock_latency	53
	Syntax	53
	Command Example	53
	set_clock_uncertainty	
	Syntax	53
	Command Example	53
	Non-timing Constraints	
	Compile Points	
	Attributes	
9:	Synthesis Optimizations	57
	Preventing Objects from Being Optimized Away	57
	Dangling Nets	
	Dangling Sequential Logic	
	Unconnected Instances	57
	Speedster Output Pad	
	Speedcore Output Pin	
	Prevent ACE Optimizing Objects Away	
	Pipelining	

Retiming	58
Forward Annotation of RTL Attributes to the Netlist	59
Example 1	59
Example 2	59
Example 3	60
Example 4	60
Example 5	61
Example 6	62
Example 7	63
Compile Points	64
Finite State Machines	66
Generating Better Results	66
Debugging the State Machines	66
FSM Encoding	66
In VHDL	67
In Verliog	67
Replication of States with High Fan-ins	67
Fanout Limit	

Chapter 10 : Synthesis User Guide Revision History......69

Chapter 1: Overview

This user guide describes how to synthesize an RTL design to generate a synthesized gate-level netlist for implementation in an Achronix device. Suggested optimization techniques are also included.

A high-level overview of the Achronix design flow is shown in figure below.

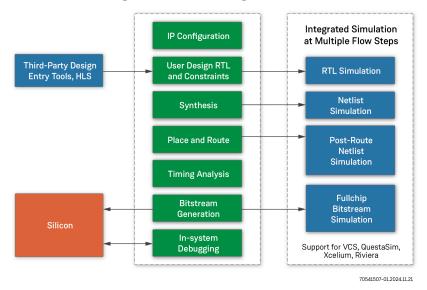


Figure 1 • Achronix Design Flow

Synthesis Flows

There are three main synthesis flows supported by the ACE tools suite:

- ACE-Driven Integrated Synthesis (page 3), where ACE owns and manages the synthesis project definition, and synthesis is run via the built-in ACE flow step.
- Synplify-Pro-Driven Integrated Synthesis (page 16), where Synplify Pro owns and manages the synthesis project definition, and synthesis is run via the built-in ACE flow step.
- Stand-Alone Synthesis in Synplify Pro (page 27), where Synplify Pro is run completely outside of ACE to generate the synthesized gate-level netlist, which is then added to the ACE project. The built-in ACE synthesis flow step is not run in this case.

Synplify Pro does not support batch mode with node-locked licenses In order to run the integrated synthesis flow steps, a floating license setup is required for Synplify Pro since ACE calls Synplify Pro in batch mode.

Chapter 2 : ACE-Driven Integrated Synthesis

As of ACE version 10.0, synthesis is now a fully integrated flow step in ACE. For designers, the simplest and easiest synthesis flow to use is the ACE-driven integrated synthesis flow. In this flow, end users do not need to leave ACE to configure or run synthesis. Users can stay in ACE and manage all aspects of design synthesis, including synthesis project setup, synthesis options configuration, running synthesis to compile the design, error reporting and log viewing, and report viewing.

In this scenario, ACE is the master of the Synplify Pro project and runs Synplify Pro from within the ACE **Run Synthesis** flow step.

🔒 Caution!

Users should not open the ACE-generated Synplify project file and make changes in Synplify Pro in this flow, because ACE will re-generate the Synplify project file from the ACE project file settings each time synthesis is run, and any changes made in Synplify Pro will be lost. To manage a Synplify project file using Synplify Pro, refer to section. Synplify-Pro-Driven Integrated Synthesis (page 16).

Synthesis Project Setup in ACE

To simplify the download, install, and licensing process, Synplify Pro is now included in the base ACE install package. Users no longer need to find the compatible version of Synplify Pro, and download it separately from ACE.

The ACE installer on Windows, and the ACE installer script on Linux have been updated to automatically install Synplify Pro as part of the ACE installation. Users no longer need to install Synplify Pro separately. In addition, some of the ACE and Synplify Pro license installation and configuration is now automated in the ACE install process. See the *ACE Installation and Licensing Guide* (UG002)¹ for more details.

As of ACE 10.2 users no longer need to set the \$ACX_SYNPLIFY_TOOL_PATH environment variable. ACE now searches for the Synplify Pro installation according to the following order of precedence:

- 1. If ACX_SYNPLIFY_TOOL_PATH is set, use it, otherwise;
- 2. Check if \$SYNPLIFY_HOME is set, and search for it there, otherwise;
- Check to see if synplify_pro is available inside the ACE install at <ace_install>/Synplify/bin/synplify_pro(.exe on Windows), otherwise;
- 4. Check to see if synplify_pro is available on the \$PATH env variable, otherwise;
- 5. Error out

2.3

Now launch ACE to get started.

Create an ACE Project

In the Projects View, click the (📫) Create Project toolbar button. Follow these steps to create the project:

1. In the Create Project Dialog, enter (or browse to) the desired path to the ACE project top-level directory in the Project Directory field.

¹ https://www.achronix.com/documentation/ace-installation-and-licensing-guide-ug002

2. Enter the desired ACE project name in the **Project Name** field and click **Finish**.

The new project will now appear in the Projects view. See "Creating Projects" or "Working with Projects and Implementations" in the *ACE Users Guide* (UG070)² for more details.

Add the Design Files and Set Project Options

In the Projects view, click the project to select it. Follow these steps to add the design source files for synthesis and place and route:

- 1. Click the (🖹) Add Source Files toolbar button and select Add RTL Files.
- 2. In the Add RTL Files dialog, browse to the source RTL directory and select all of the RTL files by holding down the **CTRL** key and clicking each file name.
- 3. Click the **Open** button to add the RTL files to the project. Repeat this process as needed until all the RTL files are added to the project.
- 4. Click the (🖹) Add Source Files toolbar button and select Add Synthesis Constraint Files.
- 5. In the "Add Synthesis Constraint Files" dialog, browse to the constraints directory and select all of the synthesis constraints files by holding down the **CTRL** key and clicking each file name.
- 6. Click the **Open** button to add the synthesis constraint files to the project. Repeat this process as needed until all the synthesis constraints files are added to the project.
- 7. Click the (🖹) Add Source Files toolbar button and select Add Place and Route Constraint Files.

(i) Notes

- This and the following steps to add place-and-route constraint Files are optional and are not required for running synthesis. These instructions only apply to continue running the flow through Place and Route.
- If a previously generated a synthesized gate level netlist exists and has been added it as a placeand-route netlist file in the ACE project, remove the netlist from the ACE project prior to running the integrated synthesis flow in ACE. ACE will automatically add the generated synthesized netlist to the ACE project as part of the Run Synthesis flow step.
- 8. In the "Add Place and Route Constraint Files" dialog, browse to the place-and-route constraints directory and select all of the files by holding down the **CTRL** key and clicking each file name.
- 9. Click the **Open** button to add the place-and-route constraint files to the project. Repeat this process as needed until all the place-and-route constraint files are added to the project.

For instructions on adding simulation files to the ACE project, please see the *Simulation User Guide* (UG072)³ or the "ACE Quickstart Tutorial" in the *ACE Users Guide* (UG070)⁴.

² https://www.achronix.com/documentation/ace-user-guide-ug070

³ https://www.achronix.com/documentation/simulation-user-guide-ug072

⁴ https://www.achronix.com/documentation/ace-user-guide-ug070

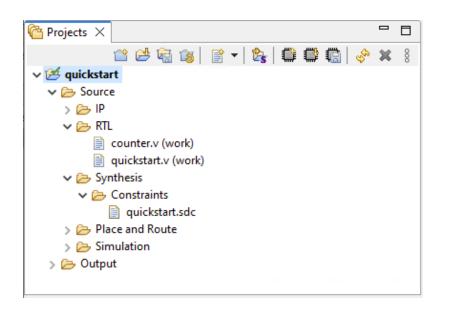


Figure 2 • Synthesis Project Source Files

In the Options View, follow these steps to configure the project options:

- 1. Expand the "Project Options" section and select the target device for the design.
- 2. Optionally, if the project contains VHDL RTL source files (*.vhdl or *.vhd), uncheck **Auto-Select Top Module** option and specify a **Top Module Name**.

(i) Note

The Synthesis flow step will error out if "Auto-Select Top Module" is enabled when using VHDL RTL source files.

3. In the "Project Options" section, scroll down and add include paths using the **HDL Include Path** widget or set the semicolon-separated list for the include paths on the TCL console as follows:

set_project_option -project quickstart hdl_include_path {D:/quickstart/src/rtl;D:/
quickstart/src/tb}

(i) Notes

- The HDL include path applies to both synthesis and simulation.
- The include path does not need to be added to the ACE libraries in <ace_install>/
 libraries. The Run Synthesis flow step will automatically add this to the include path in the
 generated Synplify Pro project file.
- 4. In the "Project Options" section, scroll down and add HDL define symbols using the **HDL Defines** widget or set the space-separated list of any HDL define symbols needed for the design on the TCL console as follows:

set_project_option -project quickstart hdl_defines {ADDR_WIDTH=16 DATA_WIDTH=8}

(i) Note

The HDL defines applies to both synthesis and simulation.

 Optionally, in the "Project Options" section, add synthesis HDL parameters using the Synthesis HDL Parameters widget or specify a list of semicolon-separated HDL parameters needed to set on the top user design module as follows:

```
set_project_option -project quickstart syn_hdl_params {{"size=32"};WIDTH 4;initializ
ation_file {"table2"}}
```

(i) Note

The synthesis HDL parameters are passed down to Synplify Pro PRJ file as-is i.e. the HDL parameters in the above example will be generated as follows:

Options ×			
oject: quicksta	art		
nplementation: impl_1			
Project Options			
Target Device	AC7t1500	\sim	
Package	F53	\sim	
Speed Grade	C2	\sim	
Core Voltage	0.85	\sim	
Junction Temperature	0	~	
Flow Mode	Evaluation	\sim	
Enable Industrial M	ode		
Auto-Select Top M	odule		
Incremental Compile			
Enable Incrementa	al Compile		
Export All Partition	5		
Enable Final Timing) Checks		
HDL Include Path		()	
D:/quickstart/src/rtl		×.	
D:/quickstart/src/tb		20	
		Û	
		4	
HDL Defines			
ADDR_WIDTH=16		-	
DATA_WIDTH=8		×	
		Ŷ	
		4	
Synthesis HDL Parame	ters	-	
{"size=32"}	L	-A	
WIDTH 4 initialization_file {"tab	ble2"}	×	
(u	,	Ŷ	
		4	
Memory Initialization	File Base Directory D:/quickstart/src/mem_init_files/base_dir		
Use Default Project			
	g Design Generation Path		
Simulation	g g ·		

Figure 3 • Synthesis Project Options

Managing Memory Initialization Files in ACE

If a design contains ROM, BRAM or LRAM IP, it may require memory initialization. One way to initialize memory is by using mem_init files. Each of the aforementioned IP can have their own mem_init_file.

To ensure that these files are correctly loaded during simulation, synthesis, place and route, and bitstream generation, the file path to the memory initialization file must be specified as either:

- An absolute file path, or
- · A relative file path.

In later case, specify the **Memory Initialization File Base Directory** (mem_init_file_base_directory) in the ACE project options.

E Options ×		
		^
Synthesis HDL Parameters	-	
	×	
	Û	
	Ţ.	
Memory Initialization File Base Directory D:/quickstart/src/mem_init_files/base_dir		
☑ Use Default Project Output Path		
✓ Use Default I/O Ring Design Generation Path		
Simulation		
Synthesis		

Figure 4 • Setting Memory Initialization File Base Directory

All memory initialization file paths used in the user design or testbench RTL files (either specified in mem_init_file parameters, or within \$readmemb or \$readmemh statements) must be relative to this single base directory path in order for all the tools (synthesis, simulation, place and route, bitstream) to find the correct file. The same rule applies to mem_init_file paths specified in ACXIP IP Configuration files.

(i) Notes

- When running simulation on Linux, ACE creates a symbolic link to the memory initialization base directory in the simulation working directory.
- When running simulation on Windows, the memory initialization base directory structure is copied into the simulation working directory. Verilog, VHDL, and other file types are filtered out of the copy operation.

(i) Note

The memory initialization file base directory applies to both synthesis and Simulation. For example, if the memory initialization file base directory is set to:

D:/quickstart/src/mem_init_files/base_dir

And if the path to mem_init_file is:

D:/quickstart/src/mem_init_files/base_dir/mem_init_file.txt

Then, the relative path set in RTL or specified in ACXIP IP configuration files must be as follows:

./base_dir/mem_init_file.txt

Hence, to maintain consistency across all tools (simulation and synthesis), the base directory name must be part of the relative mem_init_file path.

(i) Note

Since the base directory name is part of the relative mem_init_file path, ACE adds the parent directory of the memory initialization file base directory to the end of the HDL include paths so that Synplify Pro knows where to search for mem_init files. For the above example, the following include paths are automatically set in ACE-generated Synplify PRJ:

HDL Include Path and Defines
set_option -include_path {D:/build/allanlobo/main/libraries;D:/quickstart/src/
rtl;D:/quickstart/src/tb;D:/quickstart/src/mem_init_files}

Synthesis Options Configuration

Once the source files are added and the project options are set, the synthesis implementation options must also be set. In "Options View", scroll down to the "Synthesis" section and click to expand the section to show the synthesis implementation options. Ensure that the option the **ACE-Driven Synthesis** is checked.

E Options ×			Π
▼ Synthesis			
ACE-Driven Synthesis	>		
Route Delay Model	acx_custom_route_delay_1	~	
Fanout Limit	200		
Enable Retiming			
Advanced Synplify Options			
Default Frequency (MHz)	200		
- Synthesis Constraint Files			
-	Full Path		
	D:\quickstart\src\constraints\quickstart.sdc		
quickstartistic	n quickstair (sie (constraints (quickstairtisde		
Place and Route			
Advanced Place and Route	2		
Timing Analysis			
Report Generation			~

Figure 5 • Synthesis Implementation Options

Caution!

In order to run the ACE-driven integrated synthesis flow, the **ACE-Driven Synthesis** option must be checked (syn_ace_driven_synthesis project option is set to 1). If it is not checked, then project is using the **Synplify-Pro-Driven Integrated Synthesis** (page 16) flow instead.

Configure the remaining implementation options as needed for the design. Any Synplify Pro options that are not directly exposed in the ACE GUI can be set using the "Advanced Synplify Options" field. Simply enter a TCL formatted list of option-value pairs, for example:

{{option1 value1} {option2 value2}}

Synthesis implementation options can be explored automatically to find the best options for the design by using the ACE multiprocess feature as described in Synthesis Integration with Multiprocess Option Exploration (page 29).

Running Synthesis to Compile the Design

To run synthesis from within ACE, ensure that the **Run Synthesis** flow step is enabled (the checkbox is checked):

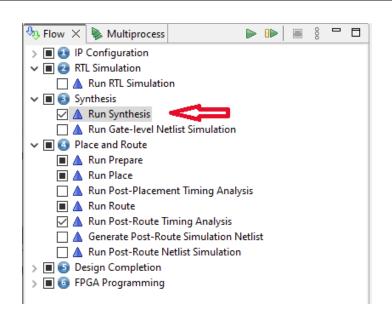


Figure 6 • Enabling the Synthesis Flow Step

To run just the Run Synthesis flow step, perform one of the following:

- Double-click on the Run Synthesis flow step
- Right-click on the Run Synthesis flow step and select Run Selected Flow Step
- Call run -step run_synthesis from the TCL console

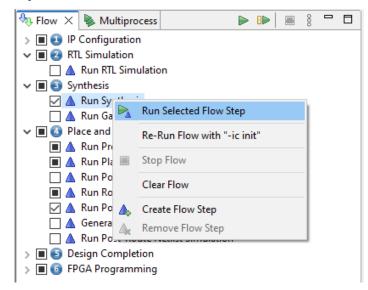


Figure 7 • Running the Run Synthesis Flow Step

UG018

The Run Synthesis flow step can be run from within the context of the overall flow by:

- Clicking on the **Run Flow** toolbar button to run the entire flow
- Call run from the TCL console to run the entire flow

If a subsequent flow step is run, ACE will automatically run all incomplete prerequisite and enabled flow steps between the selected flow step and the last completed flow step. For example, double-clicking on the **Run Post-Route Timing Analysis** flow step and none of the previous steps are complete, ACE will automatically start running the enabled flow steps in order from the beginning of the flow, including Run Synthesis if it is enabled.

The Run Synthesis flow step runs synthesis using the configuration set in the ACE project options. In this flow ACE is the master of the synthesis project (the syn_ace_driven_synthesis project option is set to 1).

The source synthesis project file will be automatically generated from the ACE project settings and managed by ACE in the Project \rightarrow Output \rightarrow (impl) \rightarrow syn directory.

All output from the underlying synthesis tool is streamed to the ACE TCL console and ACE log file. If synthesis fails, ACE will catch the error and will mark the Run Synthesis flow step state as an error with a red X and stop the flow from running any further. If synthesis succeeds, ACE will mark the Run Synthesis flow step as complete with a green check-mark icon.

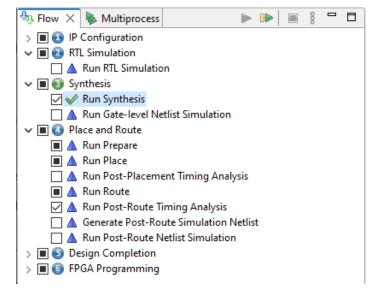


Figure 8 • Synthesis Completed Successfully

Synthesis Reports and Messages

Once synthesis completes, ACE will automatically open any relevant synthesis reports and log files in the ACE GUI Editor Area.

13

-			
ACE - Achronix CAD Environment - Version 10.2 - quickstart->impl_1 (AC7t1500)			- a ×
File Edit Actions Window Help			Q
	E Ontions X		
Construction X Construction Con		syntmp/quickstart_impl_1_srr.htm log f	
V G quickstart	Export All Partitions ^	guickstart_impl_1 (rev_acx)	# Timing report written on Ned Nov 20 14:10:35 2024
> (a) IP	Enable Final Timing Checks	Synthesis - Compiler Report	*
V 🗁 RTL	HDL Include Path	Compiler Constraint	
counter.v (work)	D:/quickstart/src/rtl	Applicator Pre-mapping Report	Top view: quickstart Requested Frequency: 8.0 MHz
guickstart.v (work) work Synthesis	D:/quickstart/src/tb	 <u>Clock Summary</u> <u>Clock Conversion</u> 	Wire load mode: top
✓ ➢ Constraints		Mapper Report	Paths requested: 5 Constraint File(s): U:\ACE RELEASES\latest-rel-v10.2\Achronix-w
quickstart.sdc	<u>Ť</u>	Timing Report RAM Report (14:10.20-	D:\quickstart\src\constraints\quickstart.sd
> 😕 Place and Route	•	Nov)	BN:MT320 : This timing report is an estimate of place and route
> 😂 Simulation	HDL Defines	Resource Utilization Constraint Checker Report	
V 🗁 Reports	ADDR WIDTH=16	(14:10 20-Nov)	<pre>@N:<u>NT322</u> : Clock constraints include only register-to-register</pre>
i multiprocess	DATA_WIDTH=8	Session Log.(14:10.20-Nov)	
> 🛐 synthesis	20		Performance Summary
> 11 timing			renormance summary
> i utilization	8		
> 📋 pins	Use Default Project Output Path		Worst slack in design: 123.953
> 👩 clocks	Use Default I/O Ring Design Generation Path		Requested Estimated Requested Estim
i routing	Simulation		Starting Clock Frequency Frequency Period Perio
others	▼ Synthesis		clk 8.0 MHz 954.8 MHz 125.000 1.047
> @ impl_1	ACE-Driven Synthesis		
Brow X & Multiprocess	Route Delay Model acx custom route delay 1		
> III III P Configuration			
V II 😢 RTL Simulation	Fanout Limit 200		
🗌 🛕 Run RTL Simulation	Enable Retirning		Clock Relationships
✓ ■ ● Synthesis ✓ Run Synthesis	Advanced Synplify Options		
kun symmetric	Default Frequency (MHz) 200		Clocks rise to rise fall to fall
✓ ■ G Place and Route	Synthesis Constraint Files		Starting Ending constraint slack constraint slack
🔳 🛕 Run Prepare	File Full Path		clk clk 125,000 123,953 No paths -
A Run Place A Run Post-Placement Timing Analysis	File Full Path auickstart.sdc D:\auickstart\src\constraints\auickstart.sdc		
A Run Route	quickstart.soc D:\quickstart.src\constraints\quickstart.soc		Note: 'No paths' indicates there are no paths in the design for t 'Diff grp' indicates that paths exist but the starting cloc
🖂 🛕 Run Post-Route Timing Analysis			
Generate Post-Route Simulation Netlist A Run Post-Route Netlist Simulation	Place and Route		
> Design Completion	Advanced Place and Route		Interface Information
> 🔳 🚯 FPGA Programming	Timing Analysis		
	Tcl Console ×		
	The netlist file named: "D:/quickstart/proj/impl_1/syn/rev_acx/quickstart_impl_1.vm" is a	lready a part of the project.	
	Profile run_synthesis Tcpu 51/+42(16+26) Twck 36:23/+50 Mpk 483/+0.0 Mcur 409/+0.0		
	Flow step "run_synthesis" completed in 50 seconds. Peak memory usage is 482 MB. Cputime !	00 seconds.	×
	cmd>		z
			56M of 512M

Figure 9 • Synthesis Reports and Messages

These reports can be found later on in the ACE Projects View under the Project \rightarrow Output \rightarrow Reports \rightarrow synthesis virtual folder. ACE automatically organizes all reports in a central location for easy access.

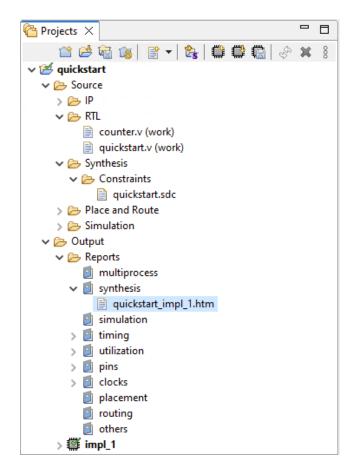


Figure 10 • ACE Project Reports Virtual Folders

Opening Synplify Project File in Synplify Pro

As of ACE 10.1, ACE allows the user to open the generated Synplify Pro project file under Project \rightarrow Output \rightarrow (impl) \rightarrow syn directory in Synplify Pro GUI. To open this file in Synplify Pro, in the Projects View tree, right-click on the ACE-generated Synplify Pro project file, i.e., **Output** \rightarrow **<active_impl>** \rightarrow **syn** \rightarrow **<project_name_impl_name>.prj** and select **Open Project in Synplify**. For more details, refer to the section Opening Synplify in ACE in the *ACE Users Guide* (UG070)⁵.

(i) Note

As of ACE release 10.3, Achronix will no longer publish ACE GUI help in the form of a PDF user guide. The contents are accessible via the the built-in ACE help system.

⁵ https://www.achronix.com/documentation/ace-user-guide-ug070

Caution!

Users can use this feature to view project file settings and schematics in Synplify Pro GUI. The ACEgenerated Synplify Pro project file should not be updated in this flow because ACE will re-generate the Synplify Pro project file from the ACE project file settings each time synthesis is run, and any changes made in Synplify Pro will be lost. To manage a Synplify Pro project file using Synplify Pro, refer to section. Synplify-Pro-Driven Integrated Synthesis (page 16).

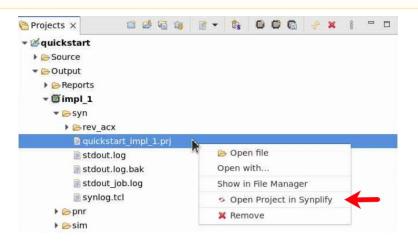


Figure 11 • Open Project in Synplify Pro for ACE-Driven Integrated Synthesis

Chapter 3 : Synplify-Pro-Driven Integrated Synthesis

As of ACE 10.0, synthesis is now a fully integrated flow step in ACE. In this hybrid flow, end users configure and manage their synthesis project in Synplify Pro and run synthesis from inside ACE. This capability enables users who are comfortable using the Synplify GUI to take advantage of the integrated Run Synthesis flow step in ACE and the automated synthesis implementation option exploration offered in the ACE multiprocess feature.

In this scenario, Synplify Pro is the master of the Synplify project file, and ACE calls Synplify Pro from within the ACE Run Synthesis flow step. In this flow, users must disable (uncheck) the **ACE-Driven Synthesis** synthesis implementation option in ACE (syn_ace_driven_synthesis project option is set to 0), and set the **Project Override Path** option to point to the source Synplify project file being managed in Synplify Pro.

When the Run Synthesis flow step is run, ACE reads in the Project Override Path project file, overrides a subset of the implementation options (to enable multiprocess), and generates a local modified copy of the project file to run from within ACE. Users *should not* open the ACE-generated Synplify project file and make changes in Synplify Pro in this flow because ACE will re-generate the Synplify project file from the ACE project file settings each time synthesis is run, and any changes made in Synplify Pro will be lost. To manage a Synplify project file using Synplify Pro, open the Project Override Path project file in Synplify instead. For more details, For more details, refer to the section Opening Synplify in ACE in the *ACE Users Guide* (UG070)⁶.

(i) Note

As of ACE release 10.3, Achronix will no longer publish ACE GUI help in the form of a PDF user guide. The contents are accessible via the the built-in ACE help system.

Configuring the Synthesis Project in Synplify Pro

The first step is to create a new synthesis project and configure the synthesis options as documented in the section, "Managing Projects in Synplify Pro (page 31)".

Synthesis Project Setup in ACE

To simplify the download, install, and licensing process, Synplify Pro is now included in the base ACE install package. Users no longer need to find the compatible version of Synplify Pro, and download it separately from ACE.

The ACE installer on Windows, and the ACE installer script on Linux have been updated to automatically install Synplify Pro as part of the ACE installation. Users no longer need to install Synplify Pro separately. In addition, some of the ACE and Synplify Pro license installation and configuration is now automated in the ACE install process. See the *ACE Installation and Licensing Guide* (UG002)⁷ for more details.

As of ACE 10.2 users no longer need to set the \$ACX_SYNPLIFY_TOOL_PATH environment variable. ACE now searches for the Synplify Pro installation according to the following order of precedence:

1. If ACX_SYNPLIFY_TOOL_PATH is set, use it, otherwise;

⁶ https://www.achronix.com/documentation/ace-user-guide-ug070

⁷ https://www.achronix.com/documentation/ace-installation-and-licensing-guide-ug002

- 2. Check if \$SYNPLIFY_HOME is set, and search for it there, otherwise;
- 3. Check to see if synplify_pro is available inside the ACE install at <ace_install>/Synplify/bin/synplify_pro(.exe on Windows), otherwise;
- 4. Check to see if synplify_pro is available on the \$PATH env variable, otherwise;
- 5. Error out

Now launch ACE to get started.

Create an ACE Project

In the Projects View, click the (📫) Create Project toolbar button. Follow these steps to create the project:

- 1. In the Create Project Dialog, enter (or browse to) the desired path to the ACE project top-level directory in the Project Directory field.
- 2. Enter the desired ACE project name in the **Project Name** field and click **Finish**.

The new project will now appear in the Projects view. See "Creating Projects" or "Working with Projects and Implementations" in the *ACE Users Guide* (UG070)⁸ for more details.

Add the Design Files and Set Project Options

In this flow, RTL files or synthesis constraints files do not need to be added to the ACE project since the synthesis project is outside of ACE. Also, the HDL Include Path, HDL Defines do not need to be configured. These settings will all be automatically imported from the Synplify Pro project file specified in the **Project Override Path** when the Run Synthesis flow step is run.

A Caution!

Paths containing environment or TCL variables are not supported as part of the automatic import of settings from the Synplify Pro project file.

In the Projects view, click the project to select it. Follow these steps to add the design source files for synthesis and place and route:

1. Click the (🖹) Add Source Files toolbar button and select Add Place and Route Constraint Files.

(i) Notes

- This and the following steps to add place-and-route constraint Files are optional and are not required for running synthesis. These instructions only apply to continue running the flow through Place and Route.
- If a previously generated a synthesized gate level netlist exists and has been added it as a placeand-route netlist file in the ACE project, remove the netlist from the ACE project prior to running the integrated synthesis flow in ACE. ACE will automatically add the generated synthesized netlist to the ACE project as part of the Run Synthesis flow step.

UG018

⁸ https://www.achronix.com/documentation/ace-user-guide-ug070

- 2. In the "Add Place and Route Constraint Files" dialog, browse to the place-and-route constraints directory and select all of the files by holding down the **CTRL** key and clicking each file name.
- 3. Click the **Open** button to add the place-and-route constraint files to the project. Repeat this process as needed until all the place-and-route constraints files are added to the project.

For instructions on adding simulation files to the ACE project, see the *Simulation User Guide* (UG072)⁹ or the "ACE Quickstart Tutorial" in the *ACE Users Guide* (UG070)¹⁰.

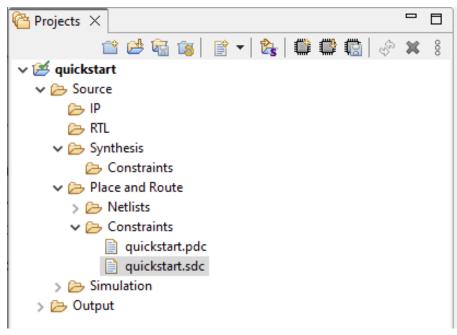


Figure 12 • ACE Project Source Files

In the Options View, follow these steps to configure your project options, expand the "Project Options" section and select the target device for the design.

(i) Note

The HDL Include Path or HDL Defines do not need to be set to run synthesis. These options may need to be configured if running simulation from within ACE. See the *Simulation User Guide* (UG072)¹¹ for details.

⁹ https://www.achronix.com/documentation/simulation-user-guide-ug072 10 https://www.achronix.com/documentation/ace-user-guide-ug070 11 https://www.achronix.com/documentation/simulation-user-guide-ug072

Options ×	start	
plementation: impl		
Project Options		
Target Device	AC7t1500	\sim
Package	F53	\sim
Speed Grade	C2	~
Core Voltage	0.85	\sim
Junction Temperatur	0	\sim
Flow Mode	Evaluation	\sim
Enable Industrial	Mode	
Auto-Select Top	Module	
Incremental Comp	ile	
Enable Increme	ntal Compile	
Export All Partition	ons	
Enable Final Timi	ng Checks	
HDL Include Path		C
		×
		Û
		÷
HDL Defines		4
		E.
		30
		÷
		4
Synthesis HDL Paran	neters	
		e de la companya de l
		30
		Ŷ
		4
Memory Initializatio	n File Base Directory D:/quickstart/src/mem_init_files/base_dir	
🗹 Use Default Proje		
	ting Design Generation Path	
Simulation		

Figure 13 • Synthesis Project Options

Managing Memory Initialization Files in ACE

If a design contains ROM, BRAM or LRAM IP, it may require memory initialization. One way to initialize memory is by using mem_init files. Each of the aforementioned IP can have their own mem_init_file.

To ensure that these files are correctly loaded during simulation, synthesis, place and route, and bitstream generation, the file path to the memory initialization file must be specified as either:

- An absolute file path, or
- A relative file path.

In later case, specify the **Memory Initialization File Base Directory** (mem_init_file_base_directory) in the ACE project options.

E Options ×		
		~
Synthesis HDL Parameters	-	
	×	
	Ŷ	
	Ŷ.	
Memory Initialization File Base Directory D:/quickstart/src/mem_init_files/base_dir		
☑ Use Default Project Output Path		
Use Default I/O Ring Design Generation Path		
Simulation		
Synthesis		

Setting Memory Initialization File Base Directory

All memory initialization file paths used in the user design or testbench RTL files (either specified in mem_init_file parameters, or within \$readmemb or \$readmemh statements) must be relative to this single base directory path in order for all the tools (synthesis, simulation, place and route, bitstream) to find the correct file. The same rule applies to mem_init_file paths specified in ACXIP IP Configuration files.

Notes

- When running simulation on Linux, ACE creates a symbolic link to the memory initialization base directory in the simulation working directory.
- When running simulation on Windows, the memory initialization base directory structure is copied into the simulation working directory. Verilog, VHDL, and other file types are filtered out of the copy operation.

(i) Note

The memory initialization file base directory applies to both synthesis and Simulation. For example, if the memory initialization file base directory is set to:

D:/quickstart/src/mem_init_files/base_dir

And if the path to mem_init_file is:

D:/quickstart/src/mem_init_files/base_dir/mem_init_file.txt

Then, the relative path set in RTL or specified in ACXIP IP configuration files must be as follows:

./base_dir/mem_init_file.txt

Hence, to maintain consistency across all tools (simulation and synthesis), the base directory name must be part of the relative mem_init_file path.

(i) Note

Since the base directory name is part of the relative mem_init_file path, ACE adds the parent directory of the memory initialization file base directory to the end of the HDL include paths so that Synplify Pro knows where to search for mem_init files. For the above example, the following include paths are automatically set in ACE-generated Synplify PRJ:

HDL Include Path and Defines
set_option -include_path {D:/build/allanlobo/main/libraries;D:/quickstart/src/
rtl;D:/quickstart/src/tb;D:/quickstart/src/mem_init_files}

Synthesis Options Configuration

Once the source files are added and the project options are set, the synthesis implementation options must also be set. In "Options View", scroll down to the "Synthesis" section and click to expand the section to show the synthesis implementation options. Ensure that the option **ACE-Driven Synthesis** is unchecked.

E Options ×			
Project: quickstart Implementation: impl_1			^
 Project Options 			
Simulation			
▼ Synthesis			
ACE-Driven Synthesis			
Project Override Path	/test/quickstart.prj	Open Project in Synplify	
Route Delay Model	acx_custom_route_delay_1	~	
Fanout Limit	200		
🗹 Enable Retiming			
Advanced Synplify Optio	ns		
Default Frequency (MHz)	200		~

Figure 14 • Opening a Synplify Pro Project File that is Specified as Project Override Path

A Caution!

In order to run the Synplify-driven integrated synthesis flow, the **ACE-Driven Synthesis** option must be unchecked (syn_ace_driven_synthesis project option is set to 0) and have the "Project Override Path" option set to point to the source Synplify project file. If the **ACE-Driven Synthesis** checkbox is checked, then the project is using the **ACE-Driven Integrated Synthesis** (page 3) flow instead.

Configure the remaining implementation options as needed for the design.

Synthesis implementation options can be explored automatically to find the best options for the design by using the ACE multiprocess feature as described in Synthesis Integration with Multiprocess Option Exploration (page 29).

Running Synthesis to Compile the Design

To run synthesis from within ACE, ensure that the **Run Synthesis** flow step is enabled (the checkbox is checked):

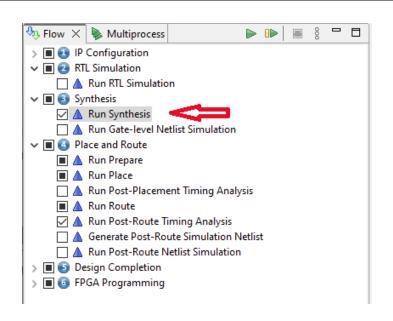


Figure 15 • Enabling the Synthesis Flow Step

To run just the Run Synthesis flow step, perform one of the following:

- Double-click on the Run Synthesis flow step
- Right-click on the Run Synthesis flow step and select Run Selected Flow Step
- Call run -step run_synthesis from the TCL console

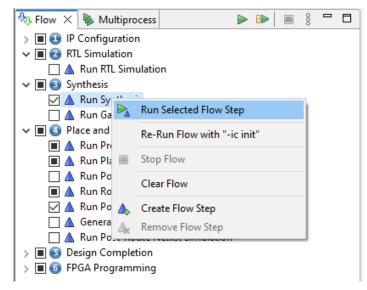


Figure 16 • Running the Run Synthesis Flow Step

UG018

The Run Synthesis flow step can be run from within the context of the overall flow by:

- Clicking on the **Run Flow** toolbar button to run the entire flow
- Call run from the TCL console to run the entire flow

If a subsequent flow step is run, ACE will automatically run all incomplete prerequisite and enabled flow steps between the selected flow step and the last completed flow step. For example, double-clicking on the **Run Post-Route Timing Analysis** flow step and none of the previous steps are complete, ACE will automatically start running the enabled flow steps in order from the beginning of the flow, including Run Synthesis if it is enabled.

The Run Synthesis flow step reads in the Project Override Path project file, overrides a subset of the implementation options (to enable multiprocess), and generates a local modified copy of the project file to run from within ACE. In this flow Synplify Pro is the master of the synthesis project (the syn_ace_driven_synthesis project option is set to 0).

(i) Note

For each implementation, ACE generates a locally modified copy of the synthesis project file in the Project \rightarrow Output->(impl) \rightarrow syn directory.

All output from the underlying synthesis tool is streamed to the ACE TCL console and ACE log file. If synthesis fails, ACE will catch the error and will mark the Run Synthesis flow step state as an error with a red X and stop the flow from running any further. If synthesis succeeds, ACE will mark the Run Synthesis flow step as complete with a green check-mark icon.

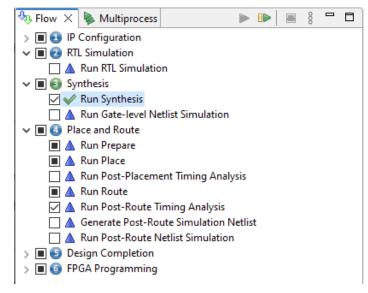


Figure 17 • Synthesis Completed Successfully

Synthesis Reports and Messages

Once synthesis completes, ACE automatically opens any relevant synthesis reports and log files in the ACE GUI Editor Area.

ACE - Achronix CAD Environment - Version 10.2 - quickstart->impl 1 (AC7(1500))			- 0 ×	
ACE - Actrona CAD Environment - Version 10.2 - quickstert->imp[_1 (AC/t1500) File Edit Actions Window Help			- 5 ×	
			9	
	E Options X	syntmp/quickstart impl 1 srr.htm log fi		
V Projects X III III III III III III III III III			NAMENAR START OF TIMING REPORT REPORT	
V 🔄 quickstart	Export All Partitions	quickstart_impl_1 (rev_acx)	# Timing report written on Wed Nov 20 14:10:35 2024	
> Control P	Enable Final Timing Checks	Compiler Report	* ¹	
V 🗁 RTL	HDL Include Path	Compiler Constraint		
counter.v (work)	D:/quickstart/src/rtl	Applicator	Top view: quickstart	
quickstart.v (work)	D:/quickstart/src/tb	Clock Summary	Requested Frequency: 8.0 MHz Wire load mode: top	
✓ ➢ Synthesis ✓ ➢ Constraints	33	Clock Conversion Mapper Report	Paths requested: 5	
guickstart.sdc	(全) (全)	Timing Report	Constraint File(s): U:\ACE_RELEASES\latest-rel-v10.2\Achronix-w D:\guickstart\src\constraints\guickstart.sd	
> 🗁 Place and Route	8	RAM Report (14:10.20.	D: (duickstart \src\constraints(duickstart.so	
> 🗁 Simulation		 Resource Utilization 	<pre>@N:HT320 : This timing report is an estimate of place and route</pre>	
v 🗁 Output	HDL Defines	Constraint Checker Report	RN:MT322 : Clock constraints include only register-to-register	
Apports Imultiprocess	ADDR_WIDTH=16	(14:10 20-Nev) Session Log (14:10 20-Nev)		
 multiprocess synthesis 	DATA_WIDTH=8			
i simulation			Performance Summary	
> 📓 timing			**********	
> 🧾 utilization				
> 🧾 pins	Use Default Project Output Path		Worst slack in design: 123.953	
> (i) clocks (ii) placement	Use Default I/O Ring Design Generation Path		Requested Estimated Requested Estim	
i routing	> Simulation		Starting Clock Frequency Frequency Period Perio	
i others	~ Synthesis		clk 8.0 MHz 954.8 MHz 125.000 1.047	
> 🕼 impl_1	Synthesis ACE-Driven Synthesis		C1K 8.0 HHZ 954.8 HHZ 125.000 1.04/	
Norman Sector Se	Project Override Path D:/test/quickstart.prj 🗀 Open Project in Synplify			
> III 🔁 IP Configuration	Route Delay Model acx custom route delay 1			
A Run RTL Simulation			Clock Relationships	
✓ ■ Synthesis	Fanout Limit 200		****************	
🖂 🛕 Run Synthesis	Enable Retiming		Clocks rise to rise fall to fall	
A Run Gate-level Netlist Simulation	Advanced Synplify Options			
Im A Run Prepare	Default Frequency (MHz) 200		Starting Ending constraint slack constraint slack	
A Run Place			clk clk 125.000 123.953 No paths -	
🗌 🛕 Run Post-Placement Timing Analysis	Synthesis Constraint Files		Note: 'No paths' indicates there are no paths in the design for t	
🔳 🛕 Run Route	File Full Path		'Diff grp' indicates that paths exist but the starting cloc	
A Run Post-Route Timing Analysis A Generate Post-Route Simulation Netlist	guickstart.sdc D:\quickstart\src\constraints\quickstart.sdc			
A Generate Post-Route Simulation Netlist				
> III S Design Completion	> Place and Route		Interface Information	
> 🔳 🚯 FPGA Programming	Advanced Place and Route			
	^			
ProgressTracker unexpected self progress value 1.000000 greater than remainder of total portion of substeps in step exec_run cmdb set_impl.option -project (quickstart) -impl (impl.1) "sym_ace_driven_symthesis" "0"				
	CHU/			

Figure 18 • Synthesis Reports and Messages

These reports can be found later on in the ACE Projects View under the Project \rightarrow Output \rightarrow Reports \rightarrow synthesis virtual folder. ACE automatically organizes all reports in a central location for easy access.

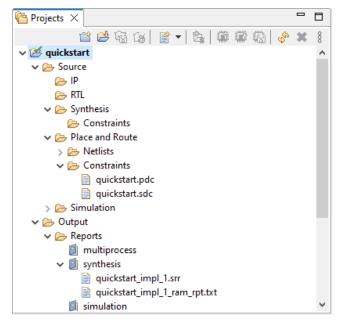


Figure 19 • ACE Project Reports Virtual Folders

Opening Synplify Project File in Synplify Pro

As of ACE 10.1, users can open the path specified as **Project Override Path** (implementation option: syn_project_override_path) by clicking the **Open Project in Synplify** button next to the path text field in the Options view. For more details, refer to the section Opening Synplify in ACE in the *ACE Users Guide* (UG070)¹².

(i) Note

As of ACE release 10.3, Achronix will no longer publish ACE GUI help in the form of a PDF user guide. The contents are accessible via the the built-in ACE help system.

E Options ×				
Project: quickstart		^		
Implementation: impl_1				
Project Options				
Simulation				
▼ Synthesis				
ACE-Driven Synthesis				
Project Override Path	/test/quickstart.prj 🕒 Open Project in	n Synplify		
Route Delay Model	acx_custom_route_delay_1	\sim		
Fanout Limit	200			
☑ Enable Retiming				
Advanced Synplify Optio	ns			
Default Frequency (MHz)	200	~		

Figure 20 • Open Project in Synplify for Synplify-Pro-Driven Integrated Synthesis

¹² https://www.achronix.com/documentation/ace-user-guide-ug070

Chapter 4: Stand-Alone Synthesis in Synplify Pro

In this flow, synthesis is run outside of ACE in Synplify Pro, and the generated gate-level synthesized netlist is added to the ACE project as a source file. In this flow, the Run Synthesis flow step in ACE is disabled (unchecked).

Configuring the Synthesis Project in Synplify Pro

The first step is to create a new synthesis project and configure the synthesis options as documented in the section, Managing Projects in Synplify Pro (page 31).

Running Synthesis

After selecting all the options according to the users design, click **OK**. The user is returned to the Synplify Pro main window to run the synthesis. From this main window, click **RUN** button to start synthesis.

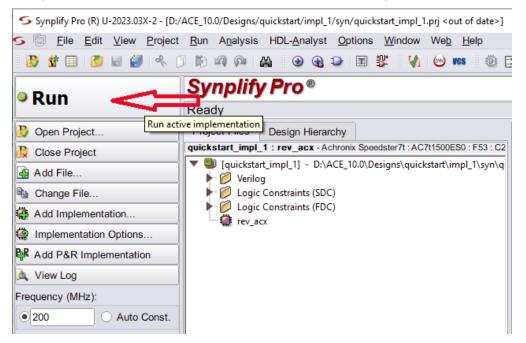


Figure 21 • Running Synthesis in Synplify Pro

Adding the Synthesized Netlist to ACE for Place and Route

Once synthesis has successfully completed, add the generated synthesized netlist to the project in ACE. In the Projects View, Click the () Add Source Files toolbar button and select Add Place and Route Netlist Files. Browse to the Synplify-generated synthesized netlist file and click **Open**.

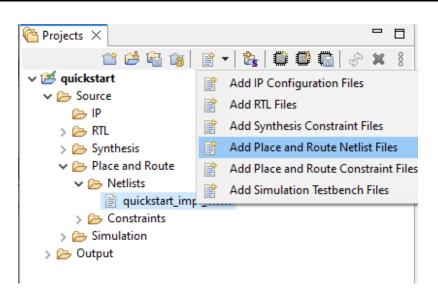


Figure 22 • Adding the Synthesized Netlist to the ACE Project

In this flow, RTL files or synthesis constraints files do not need to be added to the ACE project since the synthesis project outside of ACE. Also, the HDL Include Path, HDL Defines nor any of the synthesis implementation options in ACE need to be configured. only the synthesized gate- level netlist needs to be added to the ACE project.

When running the ACE flow steps, ensure that the option the **ACE-Driven Synthesis** is unchecked; otherwise, ACE will error as the project is not configured to run synthesis. If this happens, simply uncheck the **Run Synthesis flow** step and try running the ACE flow again.

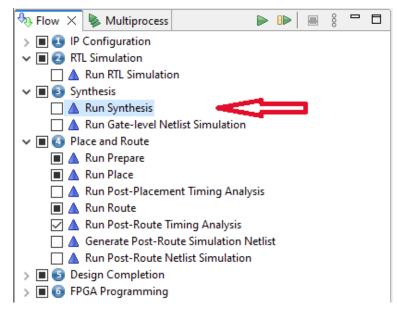


Figure 23 • Run Synthesis Flow Step Disabled

Chapter 5 : Synthesis Integration with Multiprocess Option Exploration

When using the ACE-Driven Integrated Synthesis (page 3) flow or the Synplify-Pro-Driven Integrated Synthesis (page 16) flow, users can take advantage of the automated design option exploration features built in to the ACE multiprocess tool. This tool can generate implementation option sets which sweep over both synthesis and place-and-route options to explore f_{MAX} performance variations.

The following items are required to enable synthesis implementation options exploration:

1. Enable the Run Synthesis flow step (checked in the Flow View)

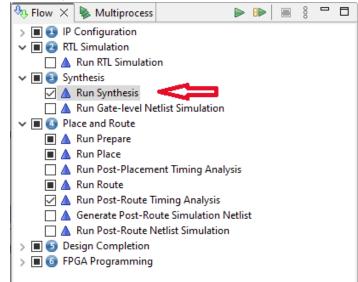


Figure 24 • Run Synthesis Flow Step Enabled

(i) Note

This option is only supported when using ACE-Driven Integrated Synthesis (page 3) or the Synplify-Pro-Driven Integrated Synthesis (page 16) flows.

2. Uncheck the **Exclude Synthesis Option** in the Multiprocess View must be unchecked

🗄 Options 🔖 Multiprocess 🗙		🔖 🖷 🗎 🗎 🗧 🗖
▼ Execution Queue Management		^
Configure the number of ACE project locally in the background, or submitte		executed simultaneously, and whether they are executed loud/grid/batch job system.
Enable Job Submission System (c	onfigured in Prefe	erences)
Parallel job count: 2 (valid rang	ge: 1-99)	
▼ Multiprocess Flow Management		
		eps that all multiprocess implementations should follow. Then, ow the implementations should run. (This provides a means of
Stop Flow After: Run Post-Route Tim	ing Analysis	×
✓ Select Implementations		
will be created if they don't already ex "Start Selected" button is pressed, the Incremental Compile is enabled, the u template (active) implementation to a	ist, or will overwri selected impleme ser can optionally Il the other imple	should be queued for execution. (Option Set Implementations ite existing implementations with the same name.) When the entations will run using the current Flow configuration. When y choose to copy the Incremental Compile DB file from the mentations before running the flow. This allows you to lock e unchanged partition place and route data to the other impls.
Existing Implementations		
Generate Implementation Option	Sets	Exclude Synthesis Options Refresh Option Sets
Seed Sweep of prime numbers; seed Sweep of prime numbers; seed Sweep of prime numbers; seed seed seed seed seed seed seed se	eedcount:	7
Copy Incremental Flow DB From	Template Impl	
Implementation	Execution State	Description
☑ 💭 impl_1	Selected	The template implementation itself.
🗹 💭 impl_1_autogen0_mpgf	Selected	This auto-generated option set changes the value for max_all
🗹 🗰 impl_1_autogen1_mrpg	Selected	This auto-generated option set changes the value for mlp_me
🗹 🋄 impl_1_autogen2_fprp	Selected	This auto-generated option set changes the value for fanout_
□ — — · · · ·		

Figure 25 • Multiprocess View

When these requirements are met, ACE will sweep over synthesis implementation options in addition to the placeand-route implementation options, which can create a wider range of performance variation and help hone in on the best options to achieve that last 5% to 10% of f_{MAX} performance boost. Refer to the "Running Multiple Flows in Parallel" section of the *ACE Users Guide* (UG070)¹³. for more details.

¹³ https://www.achronix.com/documentation/ace-user-guide-ug070

Chapter 6 : Managing Projects in Synplify Pro

This chapter is only applicable to the following synthesis flows:

- Synplify-Pro-Driven Integrated Synthesis (page 16)
- Stand-Alone Synthesis in Synplify Pro (page 27)

(i) Note

If using the **ACE-Driven Integrated Synthesis** (page 3) flow, Synplify Pro does not need to be launched outside of ACE. ACE will manage all aspects of synthesis automatically, including Synplify Pro project creation.

This guide assumes that Synplify Pro is installed with the **synplify_pro** command added to the \$PATH variable.

Creating and Setting up a Project

1. In a Linux command shell type synplify_pro to invoke Synplify Pro synthesis. When invoked, the following window will be displayed:

5 Synplify Pro (R) R-2021.03X - [<)	no projects loaded>]					- C	o x
🤔 🛅 File Edit View Projec	t Run Analysis HDL-Analyst Options Window Web Help						_6×
» 🐌 🖬 🙆 🗉 🔮 🔍) 15 49 64 🖓 🛞 62 🕞 16 🕸 V 😂 🚥 🕮 🖽 🖽						
◎Run	Synplify Pro®						
•Run	Ready						
Open Project	Project Files	Project Status Imple	mentation Directory Process View				
Diose Project				Project Settings			
Add File		Project Name	Imple	mentation Name			
Ba Change File				Run Status			
Add Implementation		Job Name	Status 🕕 🔥 CPU Time		Memory	Date/Time	
Implementation Options		Loop Hamo		nour rinic	moniory	Datorrinio	
R Add P&R Implementation							
🔌 View Log							
Frequency (MHz) 1 O Auto Const							
No projects>							
Version: R-2021.03X	20100102 2022	mormation					
Arguments: -product s ProductType: synplify_p	ymplify_pro ro						•
TCL Script Messages							
							.

Figure 26 • Synplify Pro Invoked from the Command Shell

2. Click the Open Project button on the left side to open the open project dialog-box:.

Synplify Pro (R) R-2021.03X - [<n< td=""><td></td><td>- D X</td></n<>		- D X
	Run Analysis HDL-Analyst Options Window Web Help	_6×
🚯 🔮 🔲 🙆 🗑 %) 10 49 49 49 49 49 19 19 19 19 19 19 19 19 19 19 19 19 19	
◎Run	Synplify Pro®	
	Ready	
Open Project	Project Files	Project Status Implementation Directory Process View
Close Project		Project Settings
Add File		Project Name Implementation Name
Ba Change File		⊖ Run Status
Add Implementation		Run Status Real Time Memory Date/Time
Implementation Options	S Open Project	× Kear fille menory bate/fille
R Add P&R Implementation	Recent Projects	s
🔌 View Log	Existing Project	
Frequency (MHz): 1	New Project	
		OK Cancel
Solution (1998)		
		Information (B(X)
Version: R-2021.03X		
Arguments: -product sy ProductType: synplify_pr		
4	* 1 ¹	
TCL Script Messages		
The compt		
		·



3. Click the **New Project** button to open the following window:

🗑 🗑 File Edit View Project	Run Analysis HDL-Analyst Options Window Web Help					-
🖉 🕼 🔲 🖉 📾 🎒 🔍 () ID 10 (M (M 🙆 🕢 🕞 🕢 🔛 📾 🖽 💷 💷					
PRun	Synplify Pro®					
- Run	Done: 0 errors, 6 warnings, 131 notes					
Dpen Project	Project Files Design Hierarchy		entation Direc	ctory Proces	ss View	
Close Project	quickstart : rev_1 - Achronik Speedster7t : AC7t1500ES0 : F53A0 : C1	C:\projects\quickstart\rev_1				
Add File	Verligg	Name /	Size	Туре	Modified	
Change File	with arm.v [work] ->WARNINGS: 5 ->NOTES: 110	B- Ø backup B- Ø coreip		Directory Directory	9:35:23 14-Feb-2022 9:35:23 14-Feb-2022	
Add Implementation	""""""""""""""""""""""""""""""""""""""	🖽 🭎 dm		Directory	9:41:44 14-Feb-2022	
		⊕		Directory	9:41:48 14-Feb-2022 9:41:48 14-Feb-2022	
Implementation Options				Directory Directory	9:41:48 14-Feb-2022 9:41:51 14-Feb-2022	
Add P&R Implementation		AutoConstraint_ram		sdc File	9:41:50 14-Feb-2022	
🔌 View Log		ram.fse	0 bytes 234 bytes	fse File htm File	9:41:46 14-Feb-2022 9:41:53 14-Feb-2022	
Frequency (MHz):		🛛 🗁 🛅 ram.map	28 bytes	map File	9:41:53 14-Feb-2022	
200 • Auto Const.		ram.sap	260 bytes	sap File	9:41:48 14-Feb-2022	
		ram.scf	924 bytes 12 kB	scf File Netlist	9:41:52 14-Feb-2022 9:41:50 14-Feb-2022	
utomatic Compile Point		ram.srm	9 kB		9:41:51 14-Feb-2022	
Continue on Error		ram.srr	33 kB 28 kB	srr File	9:41:53 14-Feb-2022 9:41:53 14-Feb-2022	
FSM Compiler Resource Sharing		······································	28 KB 2 kB		9:41:53 14-Feb-2022 9:41:46 14-Feb-2022	
Pipelining V		🗋 ram.vm	9 kB	Verilog Ne	9:41:52 14-Feb-2022	
Retiming V		ram_cck.rpt	2 kB	rpt File	9:41:48 14-Feb-2022 9:41:48 14-Feb-2022	
utomatic Compile Point		ram_cck.rpt.db	8 kB 2 kB	rpt.db File txt File	9:41:50 14-Feb-2022 9:41:50 14-Feb-2022	
		ram_scck.rpt	1 kB	rpt File	9:41:47 14-Feb-2022	
		ram_scck.rpt.db	8 kB		9:41:46 14-Feb-2022	
		rpt_ram1.areasrr	1 kB 883 bytes	areasrr File htm File	9:41:53 14-Feb-2022 9:41:53 14-Feb-2022	
		run_options.txt	1 kB	txt File	9:41:44 14-Feb-2022	
		scratchproject.prs	1 kB	prs File	9:41:44 14-Feb-2022	
		Version.log	30 bytes	log File	9:41:44 14-Feb-2022	
🕑 quickstart.prj						
		Information interaction and and and and and and and and and an				
eturn Code: 1						
lun Time:00h:00m:05s						
complete: Map on quicksta						
Complete: Logic Synthesis	on guickstart rev_1					
CL Script Messages						

Figure 28 • Starting a New Project

Notes

- 1. Synplify Pro can open multiple projects at once; however only one can be run at time.
- 2. A single project supports multiple implementations with each having different:
 - a. Device settings
 - b. Optimization settings
 - c. RTL define for different code builds

Adding the Synthesis Library Include File

After selecting and saving the project file inside the desired directory path, add the appropriate synthesis library include file and device specific synthesis constraints file:

- <ACE_INSTALL_DIR>/libraries/device_models/<DEVICE>_synplify.sv
- · <ACE_INSTALL_DIR>/libraries/device_models/<DEVICE>_synplify.fdc (page 31)

The first file in the project file list should be the relevant ACE library file.

For the path to ACE libraries, the **ACE_INSTALL_DIR** environment variable can be used. By manually editing the Synplify Pro .prj file, a TCL variable that stores the value of an environment variable can be defined. Then, each time the TCL variable is used, ensure the full string is enclosed in {} rather than "". For example:

```
#-- Synopsys, Inc.
#-- Version S-2021.09X-3
#-- Project file /views3/kevinhine/main/hls/PandA-Bambu/designs/pcie_mnist/syn/
pcie_mnist.prj
#-- Written on Thu Aug 31 10:01:41 2023
# Custom TCL source
syn_source {
    set ACE_INSTALL_DIR $::env(ACE_INSTALL_DIR)
}
...
add_file -verilog -vlog_std sysv {$ACE_INSTALL_DIR/libraries/device_models/
AC7t1500_synplify.sv}
set_option -include_path {../src/shell/include/;../hls/;$ACE_INSTALL_DIR/libraries/}
...
```

When the .prj is saved, the entire "syn_source" command written is preserved, as well as any places with the variable is enclosed with $\{\}$.

Warning!

If the variable is enclosed with "" instead of { }, the value of the variable will be written into the .prj on the next save.

Adding Source Files to the Project

There are two ways to add RTL source files. One is using the **Add File** button in the left menu bar, and the other one is to right-click on the project file and select **Add Source File**. Selecting either option directs the user to a dialog box listing available RTL files (see the figure below). The same procedure is followed for adding both source and constraint files.

In the examples that follow, the Speedster 7t technology has been selected, so the file AC7t1500ES0_synplify.sv is used. From this dialog box, select the desired RTL file(s) and then click **Add** followed by **OK**. The Verilog/VHDL file(s) will now be added to the project for synthesis.

Add Files to Proj	ect		X
Look in:	AC05SC0103R0 synplify.sv		
File name:	AC7t1500ES0_synplify.sv		
Files of type:	HDL Files (*.vhd *.vhdl *.v *.sv *.vma)	•	
VHDL/Verilog lib:			·J
Files to add to pro	ject: (1 file(s) selected)	Folder Options]
Z:\Achronix\Achr	onix-linux\libraries\device_models\AC7t1500ES0_synplify.sv		<- Add All
			<- Add
			Remove All ->
			Remove ->
			ОК
			Cancel

Figure 29 • Add Files to Project

Implementation Options

After adding the RTL files and constraint files, the next step is to set the implementation options. Click **Implementation Options** to open the window. shown below. This dialog box shows the default options. For example the "Fanout Guide" defaults to 10,000, but can be overwritten by the user for tuning QoR.

Implementation Options - quickstart : rev_1	×
Device Options Constraints Implementation Results Timing Report Verilog G0	CC Place an
Technology: Part Package: Speed:	rev_1
Achronix Speedster7t AC7t1500ES0 F53A0 C1	•
Device Mapping Options	
Option	lie
Fanout Guide 100	
Disable I/O Insertion	
Update Compile Point Timing Data	
Automatic Read/Write Check Insertion for RAM	
Retime Registers Forward	
Annotated Properties for Analyst	
mem_init_file 0	
Resolve Mixed Drivers	
Set the guideline for fanout-based optimizations such as replication	
	SYNOPSYS*
ОК Са	Ancel Help Predictable Success

Figure 30 • Implementation Options

(i) Note

For Achronix devices, ensure the **Disable I/O Insertion** option is checked as shown.

In the "Implementation Options" dialog box, the "Device" tab is selected by default. Each tab presentation additional options that can be set according to user's needs. Below are some guidelines for these options.

Verilog

Under this tab, the user may designate the top-level design module name. The user can also provide the names of any parameters existing in the design along with associated values. If parameters are defined in this manner, Synplify Pro propagates this value throughout the design. In this tab, the user must include the path to needed libraries under "Include Path Order." Click on the **+ file** icon to add the directory path and select from the ACE_installation path as shown below.

(i) Note

"Library Directories or Files" box can be left empty.

Device	Options	Constraints	Implementation Results	Timing Report	Verilog	GCC	Place and I	Implementations:
Top Leve	Module:		Compiler Directives and Pa	rameters				rev_1
			Parameter Name			Override \	/alue	
	Language —							
	og 2001 tem Verilog							
U 093	tern veniog							
V Push	Tristates							
Allow	Duplicate Mo	dules				Extract	Parameters	
 Multip 	le File Compi	ilation Unit						
🗌 Beta F	Features for \	/erilog	Compiler Directives: e.g. S	IZE=8				
Loop Lim	it 2000	-						
Include F	Path Order: (I	Relative to Pro	ject File)					
C:\projec	cts\quickstart	\include\						
Library D)irectories or	Files:					î ↓ <edit></edit>	
Library E	xtensions (sp	ace separated)					evinonev
								2 3110 52
					ок	Cancel	Help	Predictable Success

Figure 31 • Implementation Options: Include Path Order.

Place and Route

This tab is not presently utilized by ACE.

Timing Report

In the Timing report tab, the number of critical paths and number of start and end points can be specified to appear in the timing report. Default timing report is available in the synthesis report (.srr) file. The two available options are:

- Number of Critical paths sets the number of critical paths for the tool to report.
- Number of Start/End points specifies the number of start and end points to see reported in the critical path sections.

evice	Options	Constraints	Implementation Results	Timing Report	Verilog	GCC	Place and 4	Implementations:
umber o	of Critical Pa	aths: 100						rev_1
umbor c	f Stort/End	Points: 10						
unber c	i Stati/Ellu	Points. 10						
Descrip	tion							
		g report by spec Paths" report se	cifying the number of paths t actions.	to include in the "S	starting/End	ing Point	s with worst	

Figure 32 • Implementation Options: Timing Report

Implementation Results

Users may set their own implementation name in this tab; the default name is rev_1. The next box is the "Results Directory," specifying where users want to save the synthesized netlist file. The third box is "Results File Name," which sets the synthesized netlist file name.

Implementation Options - quickstart : rev_1				×
Device Options Constraints Implementa	tion Results Timing Re	oort Verilog G	CC Place an	Implementations:
Implementation Name:				rev_1
rev_1				
Results Directory:				
C:\projects\quickstart\rev_1			Browse	
Result Base Name:			Result Format:	
quickstart			vm 💌	
Optional Output File Options				
Write Mapped Verilog Netlist Write Mapped VHDL Netlist				
Write Vendor Constraint File				
				Synopsys
		ОК Са	ncel Help	Predictable Success

Figure 33 • Implementation Options: Implementation Results

Constraints

The Constraints tab is used to add synthesis constraint files if they were not added after adding source RTL files. This tab is also used to set the default clock speed of the design. Achronix highly recommends that a suitable constraint file be created for the synthesis project, specifying all of the clocks in the design. For details of how to add constraint files and their syntax see **Synthesis Constraints** (page 48).

In addition the default frequency should be set to the match the most common system clock frequency (by default it is set to 200 MHz).

Implementation Options -	quickstart : r	rev_1					
Device Options Con Frequency (MHz)	estraints	Implementation Results uto Constrain (Optimize to	Timing Report	Verilog	GCC	Place and	Implementations:
Use Clock Period for Un							
Check files to apply to the FPGA Constraints (FDG < <u><click add="" file="" to=""></click></u>			at (SSF)				
				ОК	Cancel	Help	SYNOPSYS Predictable Success

Figure 34 • Implementation Options: Constraints

Options

The Options tab sets the following optimization switches: **FSM Compiler, Resource Sharing, Pipelining** and **Retiming** — all are enabled by default. Users may change these optimization options according to design needs. For example, with resource sharing enabled, the software uses the same arithmetic operators for mutually exclusive statements as in branches of a case statement and hence area is optimized. Conversely, timing can be improved by disabling resource sharing, but at the expense of increased area.

mpleme	entation Opti	ons - quickstart	: rev_1						
evice	Options	Constraints	Implementation Results	Timing Report	Verilog	GCC	Place and	•	Implementations:
Optimiz	ation Switc	hes							rev_1
Aut	tomatic Con	npile Point						٦L	
Co	ntinue on Er M Compiler	ror							
Re:	source Shar								
✓ Pip ✓ Ref	elining								
Dis	tributed Cor								
🗌 Au	tomatic Con	npile Point with	soft						
									1
Option	Description								
Click (on an option	for a descriptio	n.						
									SYNOPSY
					ок	Cancel	Help	0	Predictable Success
						Gander		۲	J

Figure 35 • Implementation Options: Options

Chapter 7 : Synplify Pro Features

There are several features in Synplify Pro which can be very useful. This section covers recommendations for:

- Synplify Warnings
- Synthesis Hierarchical Report
- HDL Analyst Schematics
- Watch Window
- Validating Constraints
- Using Help

Synplify Warnings

Users can make use of strong linting and checking capabilities provided by Synplify Pro.

792 war	rnings,	1138 notes	Find: Set Filter	pply Filter	Status Page F	ilter 🗹 G	roup Common IDs
Туре	∇	ID	Message	Source Lc	Log Location	Time	Report 🔺
🖻 – 🥼	37	CL246	Input port bits 23 to 10 of ovc_avalable_all[23:0] are unused. Assign logic for all port bits or change the input port size.	-	large top.srr	01:14:52	Compiler Report
± 🔬	16	CG133	Object i is declared but not assigned. Either assign a value or remove the declaration.	-	large top.srr	01:14:52	Compiler Report
🕀 🕀	100	CG168	Type of parameter CVw on the instance class_table is not in accordance with the type of parameter on corresponding module. Pl	-	large top.srr	01:14:52	Compiler Report
🕀 – 🔔	20	CL247	Input port bit 0 of destport[4:0] is unused	-	large top.srr	01:14:52	Compiler Report
🗄 🔔	[64]	CG1239	Undriven input neighbors_r_addr on instance the router	mesh t	large top.srr	01:14:52	Compiler Report
		CG360	Removing wire refresh_w_counter, as there is no assignment to it.	router.v	large top.srr	01:14:52	Compiler Report
	2	CL156	*Input un1[1:0] to expression [ror] has undriven bits; assigning undriven bits to 0. Simulation mismatch possible. Assign all bits o	-	large top.srr	01:14:52	Compiler Report
🕀 🕀	8	CL169	Pruning unused register port_lp[5].ssa_flit_wr_all[5]. Make sure that there are no unused intermediate registers.	-	large top.srr	01:14:52	Compiler Report
😟 – 🧥	8	CL168	Removing instance extractor because it does not drive other instances. To preserve this instance, use the syn_noprune synthesis	ss allo	large top.srr	01:14:52	Compiler Report
🕀 🔼 I	[2]	CL271	Pruning unused bits 4095 to 3584 of filt_in_reg4[4095:0]. If this is not the intended behavior, drive the inputs with valid values, or i	large to	large top.srr	01:14:52	Compiler Report
		CL177	Sharing sequential element user_i_in_reg_ce. Add a syn_preserve attribute to the element to prevent sharing.	add c.s	large top.srr	01:14:52	Compiler Report
		FX474	User-specified initial value defined for some sequential elements which can prevent optimum synthesis results from being achieved.	-	larde top.srr	01:14:52	Pre-mapping Ret
L]		
TCL SC	crint	AnessaM					



Synthesis Hierarchical Report

Synplify Pro has a hierarchical report to show different design statistics. The right-hand pane also shows, Project build status, Predicted timing, Resource Utilization:

-					-		A Catting and	_				_	
\ominus					PI	rojec	t Settings						
Project Name		noc_ref_design	1_top	Device Name					rev_1: Achronix Speedster7t : AC7t1500ES0				
Implementatio	n Name	rev_1	rev_1 Top Mo						noc_ref_design	_top			
Pipelining		1		Retiming					1				
Resource Sha	ring	1		Fanout Guide					10000				
Disable I/O In	sertion	1		Disable Seque		Optim	izations		0				
Clock Conver	sion	1		FSM Compile					1				
~					_	Due	Status	_					
⊃ Job Name		0					U Time		Real Time		Date/Ti		
	(1)	Status	0	<u>^</u>		CP	UTIme		Real Time	Memory			
Compile Input Detailed report		Complete	<u>278</u>	<u>49</u>	0	-			00m:06s -		5/24/2 5:00 PM		
Premap (pren Detailed report	nap)	Complete	<u>17</u>	<u>3</u>	0	Om	n:04s		0m:05s	381MB	5/24/2 5:01 PM		
Map & Optim (fpga_mapper Detailed report		Complete	<u>1580</u>	<u>1335</u>	0	Orr	1:30s		0m:31s	414MB	5/24/2 5:01 PM		
Θ					A	rea (Summary						
DFF	11884 of 1	382400 (less than 19	%)			_	BRAM		2 of 2560 (less	than 1%)			
LRAM	1 of 2560 (less than 1%)					MLP	P 0 of 2560 (0.00%)					
LUT	48401 of 6	91200 (7.00%)			ALU8 73 of 172800 (less than 1%)								
Detailed repo	t				Hierarchical Area report								
\supset						ning	Summary						
	(clock_name))		eq Freq (req_f	req)				Freq (est_freq)		Slack (slack)	
clk_chk			-	00.0 MHz					7 MHz		-0.927		
clk_send			-	00.0 MHz 00.0 MHz				362. NA	2 MHz		-0.761 NA		
							INA						
Detailed repo	<u> </u>				_	_							
Э					Optim	izati	ons Summar	у					
Combined Clo	ck Conversio	n							2 / 0 <u>m</u>	nore			
6)					Optim	izati	ons Summar	у					
					-								

Figure 37 • Synthesis Hierarchical Report

Hierarchical Area Report

This report is useful to understand utilization of elements in the design, as well as, total sequential utilization for specific modules. The report is really helpful to understand the utilization hotspots in the design.

Module name	name LUT4 LUT6				BRAM	LRAM	MLP	PADS
⊡- Inoc_ref_design_top	47327	47752	11884	73	2	1	0	0
🐵 🛅 axi_bram_responder_Z1509640	27	271	1141	5	2	0	0	0
🐵 🛅 axi_pkt_chk_Z3124600	71	95	417	19	0	0	0	0
🖶 🗾 axi_pkt_gen_Z1803940	35	27	143	5	0	0	0	0
🗄 🛅 data_stream_pkt_chk_Z1617590	163	223	619	16	0	0	0	0
🖶 🛅 data_stream_pkt_chk_Z1626830	176	227	638	16	0	0	0	0
🗄 🛅 data_stream_pkt_gen_Z1916190	82	114	385	0	0	0	0	0
🐵 🛅 data_stream_pkt_gen_Z1925430	85	114	388	0	0	0	0	0
million map_horizontal_wrapper_Z1461360	0	0	0	0	0	0	0	0
million map_horizontal_wrapper_Z959520	0	0	0	0	0	0	0	0
million map_slave_wrapper_Z4909260	4	0	0	0	0	0	0	0
million map_vertical_wrapper_Z1492480	0	0	0	0	0	0	0	0
nap_vertical_wrapper_Z1494460	0	0	0	0	0	0	0	0

Figure 38 • Hierarchical Area Report

HDL Analyst Schematics

The Synplify Pro HDL Analyst features enable the user to visualize the end user design in several useful schematic views, including the hierarchical RTL view and flattened gate level netlist view. There are a variety of features to help filter and explore the design which can be accessed by the HDL Analyst top level menus or by right click menus within the schematic.

Browsing back and forth between the RTL view and the Technology (gate-level netlist) view enables users to visualize how the design RTL was mapped to FPGA primitives such as LUTs and registers.

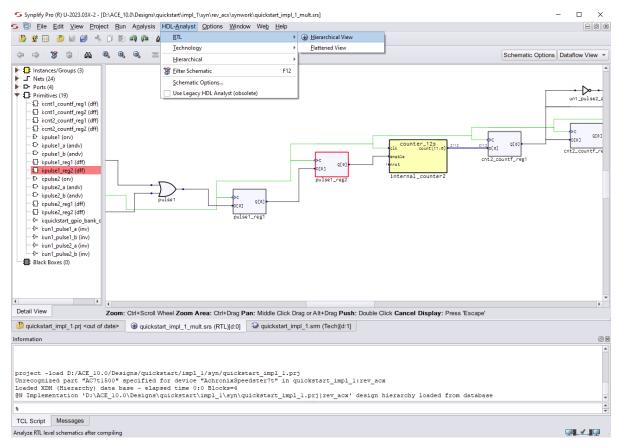


Figure 39 • HDL Analyst Hierarchical RTL View

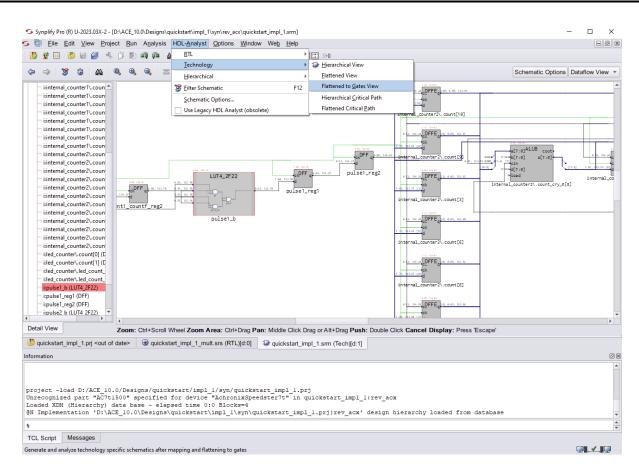


Figure 40 • HDL Analyst Flattened Gate-Level Netlist View

Watch Window

Watch window is useful to view and compare results of multiple implementations. Watch window can be enabled by the **View** \rightarrow **Watch Window** command. Click in the **Log Parameter** section of the window and then click the pull-down arrows to display the parameter choices.

(i) Note

Only a limited set of design parameters are supported for display.

To choose the implementations to watch, use the "Configure Watch" dialog box (right-click on **"Log Parameter**" section of the window) and select the implementations to watch.

Log Parameter	rev_default	rev_logic	rev_BROM	
clk - Estimated Frequency	190.4 MHz	192.4 MHz	210.7 MHz	
clk - Requested Frequency	250.0 MHz	250.0 MHz	250.0 MHz	
clk - Estimated Period	5.253	5.196	4.747	
clk - Requested Period	4.000	4.000	4.000	
clk - Slack	-1.253	-1.196	-0.747	
top Part	ac7t1500es0f53a0c2	ac7t1500es0f53a0c2	ac7t1500es0f53a0c2	
CPU Time	0h:02m:28s	0h:00m:34s	0h:02m:22s	

Figure 41 • Watch Window

Log Watch Configuration	on 🛞
Watch Selection]
O Watch Active Implementation	
Watch <u>Selected Implementations</u>	
O Watch All Implementations	
Selected Implementations to watch:	
 ✓ rev_default ✓ rev_logic 	Select All
✓ rev_BROM	
	Clear All
	ОК
	Cancel
L	J

Figure 42 • Log Watch Configuration

Validating Constraints

Synplify Pro provides a constraint checker, which runs the preliminary stages of synthesis, and then checks the project constraint files against the objects in the design. It will report if any constraints cannot be successfully applied. It is highly recommended that constraint check is run to ensure that all constraints the user requires to be applied to the design are in fact being applied.

Select Run \rightarrow Constraint Check to validate a project's constraints.

Project	<u>R</u> un	A <u>n</u> alysis HDL	<u>A</u> nalyst	<u>O</u> ptions	Window	Web H
🚯 🟦 🗐 🙋 🗑 🦧 (R	un			F8	VCS
	R	es <u>y</u> nthesize All				_
Run	c	ompile Only			F7	
	<u> </u>	rite Output Netlis	st Only			
Open Project	E	SM Explorer				
	Ī	ranslate Vendor I	O			-
没 Close Project	P	ost Place & Rout	te Resynt	hesis		na
Add File	S	ynta <u>x</u> Check			Shift+F	
월 Change File	S	ynthesis Chec <u>k</u>			Shift+F	8
Add Implementation	M c	onstraint C <u>h</u> eck			Shift+F	10
Market Ma	A	rrange VHDL File	es			
	🕑 La	aunch SYNCore.				
R Add P&R Implementation	VCS C	onfigure and Lau	nch <u>V</u> CS	Simulator		
🙇 View Log	R	un TCL Script				
Frequency (MHz):	-	un Implementatio	ons Setup			
500 Auto Const.	J	ob Status			Ctrl+J	
Automatic Compile Point	N	ext Error/Warning	9		F5	
Continue on Error	P	revious Error/Wa	rning		Shift+F	5
FSM Compiler	L	og File Message	Filter			
Resource Sharing						
Disalining						

Figure 43 • Validating Constraints

Using Help

For getting help quickly, Synplify Pro provides very useful context sensitive help. For example, to access more information about the "**Attributes**" tab of the Scope editor, click **F1** key.

) 🛣 (🗏 🚺 🖬 🚺 🔦	D 🐌 🔊	🖗 🍪 🚯 🚯	ک 🗉 😂	V) 🕑 🕬	® I 0					
urrent De	sign: <pre><top level=""></top></pre>							•	Check Co	nstraints	
Enat	ole Object Type		Object			Attribute	Value	Value Type	Des	cription	
2											
3											
4											
5											
6											
7											
8											
9											
10											
11											
12											
13											
14											
15											
16											
17											
18											
19											
20											
21											
22											
23											
24											
25											
26											
27											
28											
29											
30											
31											
Clocks	Generated Clocks	Collections	Inputs/Outputs	Registers	Delay Paths	Attributes	I/O Standards	Compile Points	TCL View		

Figure 44 • Attributes Tab Within the Scope Editor

On clicking the **F1** key, help will automatically direct to relevant section of the help.

Elle Edit View Go Bookmarks Help Feedback			
◆ • ◇ - ᠿ 🖸 🗇 😓 🙇 🔍 🔍 🤤			
Contents Index Bookmarks Search # x	+ Provinces		
Search for:	Samplify Pho for Achronia : Command Figh	renze Manual : Constraint Commands : (COOPE Tabs : Athrbuins	
+ Advanced search			
NI 4 0 - 0 of 0 Hits (b) bi	Attributes		
	You can assign attrib	utes directly in the editor.	
	Enabled (Dated Type	Object Atribute halos /w lipp Designer	nn •
	i B explore S i	bele aynyododd • synyberynel e	
	1 3	en off St funct	
	• 38	en Javas ordeats	
	5 2	an receive the second	
		anjanite a	
	Here are description	s for the Attributes columns:	
	Column	Description	
	Enabled	(Required) Tum this on to enable the constraint.	
Open Pages # ×	Object Type	Specifies the type of object to which the attribute is assigned. Choose from the pull-down list, to filter the available choices in	
Attributes		the Object field.	
	Object	(Required) Specifies the object to which the attribute is attached. This field is synchronized with the Attribute field, so selecting an object here filters the available choices in the Attribute field.	
	Attribute	Integrand population the introduct network. You can drough them a selection in the totake and available extendes for them specifical becomingly. This field is synchronized within the Object of the synchronized from the synchronized within the Object . If you select an attracter first, the synchronized within the Object matching of the object that the synchronized selection and the object of the object that the synchronized and the object that the object that the object of the object that the poper in the object of the object that the pad-obsen list, strengy page in the object of the object that the object operation values.	
	Value	(Required) Specifies the attribute value. You must specify the actitute first. Clicking in the column displays the default value, a drop-down arrow lists available values where appropriate.	
	Val Type	Specifies the kind of value for the attribute. For example, string or boolean .	
	Description	Contains a one-line description of the attribute.	
	Comment	Lets you enter comments about the attributes.	
	Enter the appropriate	attributes and their values, by clicking in a cell and choosing t	irom the pull-down menu.
	To specify an object	to which you want to assign an attribute, you may also drag-ar	nd-drop it from the RTL or Technology view into a cell in the Object column. After you have entered the attributes, save the constraint file and add it to your project.
	See Also		
	 For more information 	tion on specifying attributes, see <u>How Attributes and Directive</u>	s are Specified, on page 4.
	 For information a 	bout all SCOPE panels, see SCOPE Tabs	
	+ Provinces		© 2022 Sympetry, Inc.

Figure 45 • Sample Help Screen

Chapter 8 : Synthesis Constraints

Synplify Pro constraints can be specified in two file types:

- Synopsys design constraints (SDC) normally used for timing (clock) constraints. A second SDC file would be required for any non-timing constraints.
- FPGA design constraints (FDC) usually used for non-timing constraints; however, can contain timing constraints as well.

SDC files are usually edited using a text editor, either as part of Synplify Pro or an external editor. FDC files can be edited in either a text editor or using the Scope editor within Synplify Pro. When using Synplify Pro to edit FDC files, an assistant tab is available which provides details of available FDC commands and their format.

Timing Constraints

It is highly recommended that the user defines all clocks in the design using an SDC file. If the design has multiple clocks, clock constraints should be set accordingly, defining either appropriate clock groups or false paths between asynchronous clocks. In addition, if required, the user can specify specific duty cycles for any particular clock.

Use the create_clock timing constraint to define each input clock signal and the create_generated_clock t iming constraint to define a clock signal output from clock divider logic. The clock name (set with the -name option) will be applied to the output signal name of the source register instance. When constraining a differential clock, the user only needs to constrain the positive input.

For any clock signal that is not defined, Synplify Pro uses a default global frequency, which can be set with the set_option -frequency Tcl command in the Synplify Pro project file. However, Achronix recommends defining each clock in the design rather than relying on using this default frequency for undefined clocks.

A list of SDC commands are given below with examples. Refer to

```
fpga_reference.pdf
```

available in Synplify Pro Tool \rightarrow Help \rightarrow PDF documents for the description of the various options of the remaining SDC commands listed here.

create_clock

This command creates a clock object and defines its waveform in the current design. The options for create_clock are described in the table following.

Syntax

```
create_clock -name clockName [-add] {objectList} | -period {Value} [-waveform {riseValue
fallValue}] [-disable] [-comment commentString]
```

Command Examples

```
create_clock -name inclk -period 10 [get_ports {inclk1}]
create_clock -name divclk -period 20 [get_nets {divclk}]
create_clock -name inclkfast -period 5 -add [get_ports {inclk1}]
create_clock -name inclk -period 20 [get_ports {inclk1 inclk2 inclk3}] -waveform {
10 15 }
```

Table 1 • Option Description for create_clock

Option	Descriptions
-name clockName	Specifies the name for the clock being created, enclosed in quotation marks or curly braces. If this option is not used, the clock is given the name of the first clock source specified in the objectList option. If the objectList option is not specified, the -name option must also be used, which creates a virtual clock not associated with a port, pin, or net. Both the -name and objectList options can be used to give the clock a more descriptive name than the first source pin, port, or net. If specifying the -add option, the -name option must be used, and clocks with the same source must have different names.
-add	Specifies whether to add this clock to the existing clock or to overwrite it. Use this option when multiple clocks must be specified on the same source for simultaneous analysis with different waveforms. When this option is specified, the -name option must also be used.
-period Value	Specifies the clock period in nanoseconds (ns). The value type must be greater than zero.
-waveform riseValue fallValue	Specifies the rise and fall times for the clock in nanoseconds with respect to the clock period. The first value is a rising transition, typically the first rising transition after time zero. There must be two edges, and they are assumed to be a rise followed by a fall. The edges must be monotonically increasing. If this option is not specified, a default timing is assumed which has a rising edge of 0.0 and a falling edge of periodValue/2.
objectList	Clocks can be defined on the following objects: pins, ports, and nets.
-disable	Disables the constraint.
-comment textString	Allows the command to accept a comment string.

create_generated_clock

This command creates a generated clock object.

Syntax

```
create_generated_clock -name {clockName} [-add] -source {masterPin} -divide_by integer
```

Command Examples

The period (.) is used as a separator between levels of hierarchy and instances. The backslash (\) is only used when referencing what is inside a generate block name. For example, the RTL appears as follows:

```
generate
begin: ddr3_inst
ddr3_macro i_ddr3_macro (...)
```

set_clock_groups

Specifies clock groups that are mutually exclusive or asynchronous with each other in a design.

Syntax

```
set_clock_groups -asynchronous -name clockGroupname -group{clockList}
```

Command Example

```
set_clock_groups -asynchronous -group {clk1 clk2} -group {clk3 clk4} -name clkgroup
```

set_false_path

This command removes timing constraints from particular paths.

Syntax

```
set_false_path [-setup] [-from | -rise_from | -fall_from] [-through] [-to | -rise_to
| -fall_to] value {objectList}
```

Command Examples

set_input_delay

Sets input delay on pins or input ports relative to a clock signal.

Syntax

2.3

```
set_input_delay [-clock {clockName}] [-clock_fall] [-rise] [-fall] [-min] [-max] [-
add_delay] {delayValue} {portPinList}
```

Command Examples

```
set_input_delay 1.00 -clock clk {at} -max
set_input_delay {1.00} -clock [get_clocks {clk}] -max [get_ports {at}]
set_input_delay 2.00 -clock clk {bt} -min
set_input_delay 1.00 -clock clk -min -add_delay {bt}
set_input_delay 3.00 -clock clk {st}
set_input_delay 4.00 -clock clk -add_delay {st}
set_input_delay 1.00 -clock clk {din2} -clock_fall
set_input_delay 1.50 -clock clk {din1 din2}
set_input_delay 2.00 -clock clk [all_inputs]
```

set_output_delay

Sets output delay on pins or output ports relative to a clock signal.

Syntax

```
set_output_delay [-clock clockName [-clock_fall]] [-rise|[-fall] [-min|-max] [-
add_delay] delayValue {portPinList} [-disable] [-comment commentString]
```

Command Examples

```
set_output_delay 1.00 -clock clk {o1} -max
set_output_delay 3.00 -clock clk -max -add_delay {o1}
set_output_delay 2.00 -clock clk {o2} -min
```

set_max_delay

Specifies a maximum delay target for paths in the current design.

Syntax

```
set_max_delay [-from |-rise_from | -fall_from] [-through] [-to | -rise_to | -fall_to]
{delay_value}
```

Command Examples

```
set_max_delay 2 -from {a b } -to {o1}
set_max_delay -rise_from {clk} {1}
set_max_delay -through {{n:dout1}} {1}
set_max_delay 1 -fall_to {clk1}
```

set_multicycle_path

Modifies single-cycle timing relationship of a constrained path.

Syntax

```
set_multicycle_path [-start |-end] [-from {objectList}] [-through {objectList} [-through
{objectList} ...] ] [-to {objectList}] pathMultiplier [-disable] [-comment
commentString]
```

Command Examples

```
set_multicycle_path 2 -from [get_clocks inclk1] -to [get_clocks inclk2]
set_multicycle_path 4 -from temp2 -to out
```

set_clock_latency

Specifies clock network latency.

Syntax

```
set_clock_latency -source [-clock {clockList}] delayValue {objectList}
```

Command Example

```
set_clock_latency 0.2 -source [get_ports clk] -clock [get_clocks {clk}]
```

set_clock_uncertainty

Specifies the uncertainty or skew of the specified clock networks.

Syntax

```
set_clock_uncertainty {objectList} -from fromClock |-rise_from riseFromClock |
-fall_from fallFromClock -to toClock |-rise_to riseToClock | -fall_to fallToClock value
```

Command Example

```
set_clock_uncertainty 0.4 [get_clocks clk]
```

Below is an example of clock constraint commands for a multiple clock domain design.

(i) Note

Most timing engines only use up to three decimal places of accuracy; therefore, it is normal to truncate non-rational values to this level.

```
# Clock definitions
create_clock -period 10
                           [ get_ports
{pll_refclk_p}
                                                                                ] -name
pll_refclk_p
create_clock -period 100
                           [ get_ports
{tck}
                                                                                ] -name
tck
create_clock -period 1.527 [ get_pins
{i_clock_generator.i_PLL_EN.SW_APLL_0_pll_en_clk_APLL.iACX_PLL/ogg_gm_clk[0]}
                                                                                 ] -name
en_mac_ref_clk
create_clock -period 3.175 [ get_pins
{i_clock_generator.i_PLL_FF.SW_APLL_1_pll_ff_clk_APLL.iACX_PLL/ogg_gm_clk[0]}
                                                                                 ] -name
ff_clk
create_clock -period 3.448 [ get_pins
{i_clock_generator.i_PLL_SYS.SW_APLL_2_pll_sys_clk_APLL.iACX_PLL/ogg_gm_clk[0]} ] -name
sys_clk
create_clock -period 62.5 [ get_pins
{i_clock_generator.i_PLL_DCC.SW_APLL_3_pll_dcc_clk_APLL.iACX_PLL/ogg_gm_clk[0]} ] -name
sbus_clk
# By specifying clock group, each of the above clocks will be determined to be
asynchronous to all other clocks
set_clock_groups -asynchronous -name clk_grp1 -group {sbus_clk}
                                              -group {en_mac_ref_clk} \
                                              -group {pll_refclk_p}
                                                                       ١
                                              -group {sys_clk}
                                                                       \
                                              -group {ff_clk}
                                                                       \
                                              -group {tck}
```

Non-timing Constraints

An FDC file is used to specify non-timing constraints, which can be either attributes on an object (global or local), using the define_attribute statement, or compile points.

Compile Points

To implement compile points, they are specified in the FDC file as follows.

(i) Note

For a detailed explanation of compile points how and when to use them, see Compile Points (page 64).

To set a single compile point, enter:

define_compile_point {v:work.pac_ddr3_ip} -type {locked}

To find every instance of a module and set as a compile point, enter:

Compile Point syntax

```
foreach inst [c_list [find -hier -view pac_ddr3_ip*]] {
    define_compile_point $inst -type {locked}
}
```

Attributes

Attributes provides a mechanism to control how a design is mapped by Synplify Pro. Attributes can be defined both globally and also applied to individual instances. Attributes can be entered both in HDLs or in the SCOPE attributes tab, FDC files for project-wide entities. Attributes with **syn_*** do not affect synthesis and passed to the netlist.

Here is summary and examples of some of these attributes:

Attribute	Description
syn_allow_retiming	Controls retiming of registers across combinatorial logic on a global level or to specific register.
syn_dspstyle	Controls the mapping of objects to technology-specific DSP components. Options are "dsp" and "logic" for DSP64 or LUT/FF, respectively.
syn_ramstyle	Controls the implementation of an inferred RAM. Options are "block_ram", "logic_ram", and "registers" for BRAM, LRAM, and registers, respectively.
syn_romstyle	Controls the implementation of an inferred ROM. Options are "block_rom", "logic_rom" for BRAM and LRAM, respectively.
syn_keep	To preserve net in synthesis during optimization.
syn_preserve	To prevent sequential optimizations.
syn_noprune	To prevent optimization on instances and black boxes when output is not used.
syn_maxfan	To override global fanout guide for an individual port, net, register.

To override the number of available resources in a device, enter the following command. This command can be used to limit the mapping to certain resources.

define_global_attribute syn_allowed_resources {blockrams=1000}

To synthesize all ROMs using logic, enter:

define_global_attribute {syn_romstyle} {logic}

To ensure that RAMs are only inferred for sufficiently large register sets, enter:

```
define_global_attribute {syn_max_memsize_reg} {2048}
```

For more detailed information on all the supported attributes, refer to Synplify Pro online help "Attribute Reference Manual"

Chapter 9 : Synthesis Optimizations

There are several optimizations that can be performed by the user during Synplify Pro synthesis. This sections covers recommendations for:

- Preventing Objects from Being Optimized Away (page 57)
- Pipelining (page 58)
- Retiming (page 58)
- Forward Annotation of RTL Attributes to the Netlist (page 59)
- Compile Points (page 64)
- Finite State Machines (page 66)

Preventing Objects from Being Optimized Away

Dangling Nets

Synplify Pro always performs optimization on redundant or feed-through nets. At times, the user may want to preserve these nets. In order for these nets not to be optimized away (removed), add the following directive to the RTL, In this example, synthesis will not optimize away (remove) the logic. Instead, it infers a buffer between the two wire statements. If it is not specified, the user may not see the buffer insertion by the tool.

```
wire net1 /* synthesis syn_keep = 1 */ ;
wire net2 ;
assign net2 = net1 ;
```

Dangling Sequential Logic

For sequential logic the syn_preserve attribute is used.

```
reg net_reg1 /* synthesis syn_preserve = 1 */ ;
always @ (posedge clk)
    net_reg1 <= some_net;</pre>
```

Unconnected Instances

For input instances when their output pins are unconnected, the syn_noprune attribute is used. The following examples show how to apply this attribute to both Speedster I/O pads and Speedcore boundary pins.

Speedster Output Pad

PADIN ipad (.padin(in[0])) /* synthesis syn_noprune = 1 */;

Speedcore Output Pin

IPIN ipin (.din(in[0])) /* synthesis syn_noprune = 1 */;

Prevent ACE Optimizing Objects Away

In the above examples, Synplify Pro does not remove the unconnected entity, ensuring that the Synplify Pro netlist retains these entities. However, when the netlist is read into ACE, ACE performs netlist optimization and resynthesis. If the objects retained by synthesis are still unconnected, then ACE will remove these entities from the final placeand-route netlist. To prevent ACE from optimizing these entities, use the ACE must_keep directive in conjunction with the above attributes. Using the preceding sequential logic example, the must_keep attribute is passed through Synplify and included in the synthesized netlist. ACE will then recognize this attribute and keep the instance.

(i) Note

The attribute <code>must_keep</code> can be applied to both sequential elements and wires.

```
(* must_keep=1 *) reg net_reg1 /* synthesis syn_preserve = 1 */ ;
always @ (posedge clk)
    net_reg1 <= some_net;</pre>
```

Pipelining

Pipelining is the process of splitting logic into stages so that the first stage can begin processing new inputs while the last stage is finishing the previous inputs. Pipelining ensures better throughput and faster circuit performance. If using selected technologies which use pipelining, also use the related technique of retiming to improve performance.

When this switch is enabled in a project file, synthesis uses register balancing and pipeline registers on multipliers and ROMs. This option is equivalent to enabling the Pipelining option on the Options panel of the Implementation Options dialog box.

Retiming

The retiming process moves storage devices (flip-flops) across computational elements with no memory (only gates/LUTs) to improve the performance of the circuit.

When this switch is enabled, synthesis tries to improve the timing performance of sequential circuits. This option is equivalent to enabling the Retiming option on the Options panel of the Implementation Options dialog box. Use the syn_allow_retiming attribute to enable or disable retiming for individual flip-flops. This option also adds a retiming report to the log file.

(i) Note

Pipelining is automatically enabled when retiming is enabled.

Forward Annotation of RTL Attributes to the Netlist

Synplify Pro supports forward annotation of RTL attributes to the netlist. These user-defined attributes propagate to the netlist to be used by ACE place and route for optimization. This feature requires the usage of various directives available in Synplify Pro such as syn_noprune, syn_keep, syn_hier, syn_preserve, etc., to propagate user-define attributes to the netlist. The table below lists the directives to be set on the mentioned objects in order to forward annotate the RTL attribute.

Object	Directive	Result
Module	syn_hier="hard"	Attribute applied on the module will propagate to the netlist
Instantiated Components	syn_noprune	Attribute applied on the instantiated component will propagate to the netlist
Input/Output ports	syn_hier="hard" on the module containing the ports	Attribute applied on ports will propagate to the input/output port in the netlist
Registers	syn_preserve	Attribute applied on the registers will propagate to the netlist
Wire	syn_keep	Attribute applied on nets/wires will propagate to the netlist

Below are some examples:

Example 1

The attribute weight="3.0" propagates to my_reg in the netlist. The syntax used is Verilog 2001 style parenthetical comments.

```
(* syn_preserve=1, weight="3.0" *) reg my_reg;
```

Example 2

The syntax used is C-style comment.

reg my_reg /* synthesis syn_preserve=1 weight=4 */;

(i) Note

When using C-style comment, a comma is *not* required after syn_preserve=1. When using Verilog 2001 style, a comma *is required* after syn_preserve=1.

Example 3

This example illustrate attribute propagation on nets.

```
(* syn_keep = 1, weight = 3 *) wire n2;
```

Example 4

This feature of attribute propagation is utilized in flop pushing to boundary pins or I/O pads via the ACE attribute syn_useioff. The syn_useioff is applied to the input and output ports in the below example.

```
module flop_push_test1 (
    ina, inb, sel, clk, z0
);
input wire [3:0] ina /* synthesis syn_useioff=1 */;
input wire [3:0] inb /* synthesis syn_useioff=0 */;
input wire
                   sel /* synthesis syn_useioff=1 */;
input wire
                   clk;
output reg
                        /* synthesis syn_useioff=1 */;
                   z0
               sel_r0=1'b0, sel_r1=1'b0;
    reg
    reg [3:0] ina_r0=4'h0, ina_r1=4'h0, inb_r0=4'h0, inb_r1=4'h0;
    always @(posedge clk)
    begin
      sel_r0 <= sel;</pre>
      sel_r1 <= sel_r0;</pre>
      ina_r0 <= ina;</pre>
      ina_r1 <= ina_r0;</pre>
      inb_r0 <= inb;</pre>
      inb_r1 <= inb_r0;</pre>
      z0 <= sel_r1 ? & inb_r1 : |ina_r1;</pre>
    end
endmodule
```

(i) Note

In example 4, the module flop_push_test1 is a top module; therefore, syn_hier="hard" is not specified on the module. If it were a sub module, syn_hier="hard" is required for the attribute on ports to propagate to the netlist; for example:

module flop_push_test1 (ina, inb, sel, clk, z0) /* synthesis syn_hier="hard" */;

(i) Note

In example 4, the syn_useioff attribute could also be specified in the Verilog 2001 comment style. For example:

(* syn_useioff=1 *) input [3:0] ina;

However, that style only works correctly when the attribute has a non-zero value. Synplify Pro cannot distinguish between the value zero and and an attribute that is not present. In that case it will not forward annotate the attribute to the netlist used by ACE. Therefore, it is recommended to always use the C-style comment used in example 4.

Example 5

This example illustrates attribute propagation on instantiated components:

```
module att_propagate_instcomp (
    d1, d2, d3, clk, out1
);
input wire d1,d2, d3, clk;
output reg out1;
reg q1,q2;
//Instantiate 2 instances U1 and U2 of module test2
    (* must_keep = 1, syn_noprune = 1 *) test2 U1 (d1,d2,d3, clk,out2);
    (* syn_noprune = 1, must_keep = 1 *) test2 U2 (d1,d2,d3, clk,out2);
    always @(posedge clk)
    q1 <= d1;
    assign combol = q1 & d2 & d3;
    always @(posedge clk)
    q2 <= combol;</pre>
```

```
assign combo2 = q2 | combo1;
   always @(posedge clk)
       out1 <= combo2;</pre>
endmodule
// -----
                -----
module test2 (
   d1, d2, d3, clk, out1
) /*synthesis syn_hier = hard */;
input wire d1, d2, d3, clk;
output reg out1;
reg q1,q2;
   always @(posedge clk)
       q1 <= d1;
   assign combol = q1 | d2 | d3;
   always @(posedge clk)
       q2 <= combo1;
   assign combo2 = q2 & combo1;
   always @(posedge clk)
       out1 <= combo2;</pre>
endmodule
```

Example 6

This example shows attribute propagation on modules:

```
(* att0=1 *) module top (
    d1, d2, d3, clk, out1, out2
);
input wire d1, d2, d3, clk;
output wire out2;
output wire out1,
    // Instantiate test1
```

```
test1 U1 (d1, d2, d3, clk, out1);
endmodule
// ------
(* must_keep=1 *) module test1 (
   d1, d2, d3, clk, out1
) /* synthesis syn_hier="hard" */;
input wire d1, d2, d3, clk;
output reg out1;
reg q1,q2;
   always @(posedge clk)
       q1 <= d1;
   assign combo1 = q1 & d2 & d3;
   always @(posedge clk)
       q2 <= combo1;
   assign combo2 = q2 | combo1;
   always @(posedge clk)
       out1 <= combo2;</pre>
endmodule
```

Example 7

As shown above, flop pushing can take advantage of attribute propagation to control specific I/O pads or boundary pins. The examples below shows how to control flop pushing from within the RTL, applying the attribute to both Speedster I/O pads and Speedcore device boundary pins.

This example illustrates the application of the syn_useioff attribute with a value of 0 on, respectively:

- A wire
- A black-box PAD instance, an IPIN instance, the IPIN input net, the IPIN output net (Speedcore only)
- An PADIN instance (Speedster only)
- · A pair of DFF instances

All of the above are valid instances to which to apply this property:

```
wire dff1_q, dff2_q;
```

```
BB_PADIN i_bb_padin
                    ( .padin(sc_in) , .dout(bb_pad_dout)
                                                            ) /* synthesis
syn_useioff = 0 */;
PADIN
        i_padin
                    ( .padin(sp_in) , .dout(padin_dout)
                                                               /* synthesis
                                                            )
syn_useioff = 0 */;
                    ( .din(ipad_dout), .dout(ipin_dout)
                                                            ) /* synthesis
IPIN
        i_ipin
syn_useioff = 0 */;
ACX_DFF i_dff1
                    ( .d(ipin_dout) , .ck(clk) . .q(dff1_q) ) /* synthesis
syn_useioff = 0 */;
                    ( .d(ipin_dout) , .ck(clk) . .q(dff2_q) ) /* synthesis
ACX_DFF i_dff2
syn_useioff = 0 */;
```

For full details on all the options for flop pushing, see the section "Automatic Flop Pushing into I/O Pins" in the *ACE* Users Guide (UG070)¹⁴.

(i) Note

As in Example 7, the syn_useioff attribute must be specified with a synthesis directive in a C-style comment because it has a value of zero. However, the syn_keep=1 attribute on the wire can be specified in either style.

Compile Points

Compile points are RTL partitions of the design which are defined before synthesizing a design. The advantages of using compile points is design preservation, runtime savings and improves efficiency of top-down and traditional bottom-up design flows.

Synplify Pro supports both automatic and manual compile points. The automatic compile-point feature can be selected from "Implementation Options" dialog box as shown below. When automatic compile points are enabled, the tool automatically identifies compile points based on various parameters such as size of the design, hierarchical modules, boundary logic, etc. Refer the

fpga_user_guide.pdf

available with Synplify Pro for details on compile points.

¹⁴ https://www.achronix.com/documentation/ace-user-guide-ug070

Run		Synplif	y Pro	®									
* Kull		Ready											
没 Open Project	1	Project Files	Design	Hierarchy		(Project Status Implementat			ation	tion Directory		
Close Project		gddr_ref_desig	n_top : rev	_1 (gddr_ref_d	esign_top) - Achronix Speedste	er7t : AC7t1500 : F5							
Add File		1 🕑 Implement	ntation Opti	ions - gddr_ref_c	lesign_top : rev_1								
Change File		Device	Options	Constraints	Implementation Results	Timing Report	Verilog	GCC	Place and	4.14	Implementa		
Add Implementation					Implementation Resolution	Thing Report	veniog	000	T Idoc din		rev_1		
Implementation Options	s	Optimizati	ion Switche	es									
R Add P&R Implementati			omatic Com tinue on Err										
A View Log		✓ Valic	date MIF File										
Frequency (MHz):			l Compiler ource Sharin	g									
• 200 O Auto	Const.	Reti	lining ming ributed Con	pilation									
Automatic Compile Point				pile Point with so	oft								
Continue on Error	~												
FSM Compiler	V												
Resource Sharing	v												
Pipelining	~												
Retiming													
Automatic Compile Point wit													
		Option [Description										
		Click o	n an option	for a description	n.								
											SVDD		
							ж	Cancel	Help		Predictable		

Figure 46 • Setting Compile Points

Although compile points can deliver significant runtime savings, users should be aware that they can have a detrimental effect on quality of results (QoR) if not used with care. Compile points identify blocks of code that are repeated, guiding Synplify Pro to only synthesize that block once. The level of optimization between a compile point and it's enclosing module is defined by the compile point type:

- Locked No optimizations across compile point boundary. Locked compile points are used for the Achronix incremental compile flow
- Hard Signals can be optimized across the compile point boundary (i.e., back-to-back inverters removed). However, the actual interface is not optimized — all signals remain. All automatic compile points are set to hard.
- **Soft** Signals can be optimized across the compile point boundary, and the signals themselves may be removed, or renamed. Therefore, almost full optimization can occur as though the design did not have compile points.

The three modes above result in increasing runtimes; however, they also generally result in increased QoR as greater optimizations can be performed. Users should determine which configuration of compile points, if any, best meet their needs with regards to performance versus runtime.

Caution!

If automatic compile points are enabled, users must be aware that all automatic compile points are set to **hard**. Therefore, it may not be possible to achieve the highest QoR.

(i) Note

Compile points will only have a significant effect on runtime either when used as locked to enable incremental synthesis (and place and route), or else in designs with a large number of repeating structures.

Finite State Machines

The FSM compiler is an automatic tool for encoding state machines. FSM coding style in the RTL design will directly impact performance. By default Synplify Pro implements the following FSM encoding:

- 0-4 states is binary encoded
- 5-40 states is one-hot encoded
- >40 states is Gray encoded

FSM compiler is used to generate better results and to debug state machines.

Generating Better Results

The software uses optimization techniques that are specifically tuned for FSMs such as reachability analysis. The FSM compiler examines the design for state machines, converting them to a symbolic form that provides a better starting point for logic optimization. The FSM compiler may convert an encoded state machine into a different encoding style (to improve speed and area utilization) without changing the source. This optimization can be overridden by choosing a particular encoding style through appropriate synthesis attributes in the RTL design.

Debugging the State Machines

State machine description errors can result in unreachable states. The user can also use the FSM viewer to see a high-level bubble diagrams and cross-probe from the diagram with respect to RTL. The user can then check whether the source code describes the state(s) correctly.

FSM Encoding

There are two choices to define the encoding via attributes in the RTL code:

- Use syn_encoding attribute and enable the FSM compiler.
- Use syn_enum_encoding to define the states (sequential, one-hot, gray, and safe) and disable the FSM compiler. If the user does not disable the FSM compiler, the syn_enum_encoding values are not implemented. This behavior is because the FSM compiler, which is a mapper operation, overrides any user attributes for the FSM encoding. The FSM compiler can be disabled via the GUI or the from the Synplify Pro project file with the following syntax:

```
set_option -symbolic_fsm_compiler 0
```

The user may also direct the synthesis process to deploy a user-defined FSM encoding, for example:

attribute syn_enum_encoding of state_type: type is "001 010 101";

There is a synthesis attribute to turn on/off FSM extraction. By using this attribute the user can see how state machines are extracted. The attributes is set in the source code as follows:

- Specify a state machine for extraction and optimization syn_state_machine=1
- Prevent state machines from being extracted and optimized syn_state_machine=0

In VHDL

```
----- Attribute ----
attribute syn_state_machine : boolean;
attribute syn_state_machine of tx_training_cstate : signal is true;
```

In Verliog

If user does not want to optimize the state machine, add the syn_state_machine directive to the registers in the Verilog code. Set the value to 0. When synthesized, these registers are not extracted as state machines.

```
reg [39:0] curstate /* synthesis syn_state_machine=0 */ ;
```

For greater than 40 states, Synplify Pro performs Gray encoding. For one-hot encoding, specify the syn_encoding = "onehot" as shown below.

reg [39:0] state /* synthesis syn_encoding = "onehot" */ ;

Replication of States with High Fan-ins

Large and complex state machines present another unique challenge in state machine design. Complex state machines can be made to run faster by actually making them larger by adding more states. This technique can be counter intuitive as the number of levels of logic between the states and not the number of states typically limits state machine performance. The performance of a state machine is limited by both the number of fan-ins into a given state and the decisions made in that state. For example, idle-type states can have a large number of inputs plus increased computational load. With the 6-input LUT architecture of Achronix devices, once the number of fan-ins exceeds six, another level of logic is needed. An easy method to reduce the number of fan-ins is to replicate these states. The duplicated high fan-in states reduce the number of inputs, thus reducing the number of levels of logic.

Both state machines in the figure below are equivalent in function, but State A is duplicated in Version II so that A and A1 have two or less return inputs. As a result, if each state has to deal with four additional inputs, they can now be contained in one 6-input LUT. Although this example is simplistic, the methodology can be applied to larger and more complex state machines.

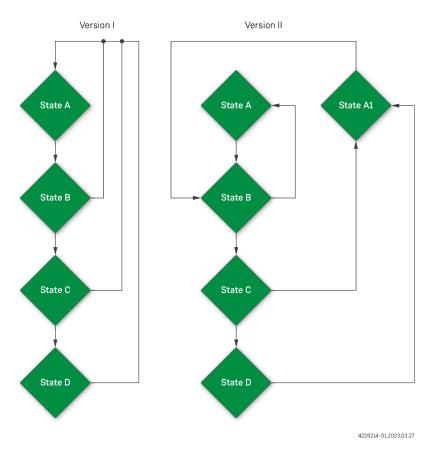


Figure 47 • Replicated High Fan-in State Example

Fanout Limit

This fanout limit can also be controller through RTL design. In this case if the user knows about a net with high fanout and wants to replicate the cell after a certain fanout is reached, the following coding style is needed:

Here Synplify Pro will infer a buffer/logic if the fanout limit on net1 exceeds 8.

Chapter 10 : Synthesis User Guide Revision History

Version	Date	Description
1.0	17 Jul 2016	 Initial revision. Ported document to Confluence and made it Speedcore specific.
1.1	31 Oct 2016	 Fix for minor type and additional clock constraint example. Updated document template to include confidentiality note.
1.2	31 Mar 2017	• Corrected one of the create_generated_clock examples in the code block.
1.3	01 Oct 2018	 Synthesis Optimizations (page 57): Corrected the syn_keep attribute in Example 7 (page 63). Removed the instantiation templates, referred the user to the Speedcore IP Component Library User Guide (UG065). Added details on Compile Points. (page 64) Updated DSP64 (page 0) . Updated Block RAM (page 0) . Managing Projects in Synplify Pro (page 31): Removed references to version L-2016 limitations. Example Synplify-Pro Project File: Removed internal paths from file names.
1.4	10 Jun 2019	 Synthesis Optimizations (page 57): Removed technology specific entries to make the guide suitable for all technologies. Technology specific parts moved to their appropriate IP Component Library User Guide Specifically removed inference templates for Speedster16t parts, (DSP64, BRAMTDP & BRAMSDP). Managing Projects in Synplify Pro (page 31): Combined Speedster and Speedcore differing library files into single Synthesis library include files table. Example Synplify-Pro Project File: Added ACE_INSTALL_DIR environment variable to example project file

Version	Date	Description
2.0	20 Jun 2024	 Overview (page 1): Minor correction. Added major new content for integrated synthesis flows with ACE 10.0 and beyond: ACE-Driven Integrated Synthesis (page 3) Synplify-Pro-Driven Integrated Synthesis (page 16) Stand-Alone Synthesis in Synplify Pro (page 27) Managing Projects in Synplify Pro (page 31) Added chapter Synthesis Integration with Multiprocess Option
2.1	20 Aug 2024	 Exploration (page 29) Updated screenshots for ACE 10.1. Added information on new ACE feature to open Synplify Pro projects in the Synplify Pro GUI from within ACE.
2.2	05 Dec 2024	 Updated screenshots for ACE 10.2 Updated information on Synplify Pro installation and environment path setup Updated content for the name change of synthesis project option "Generate Project File" to "ACE-Driven Synthesis"
2.3	16 Apr 2025	 Updated screenshots for ACE 10.3. Added a new section "Managing Memory Initialization Files in ACE" in the following pages: ACE-Driven Integrated Synthesis (page 3) Synplify-Pro-Driven Integrated Synthesis (page 16)