Libero SoC v11.0

User's Guide

NOTE: PDF files are intended to be viewed on the printed page; links and cross-references in this PDF file may point to external files and generate an error when clicked. **View the online help included with software to enable all linked content.**

Table of Contents

Supported Families....10
Firmware Cores Frequently Asked Questions....15
General Questions....23
Instantiating into your SmartDesign....24
Working with Processor-Based Designs in SmartDesign....25
Making your Design Look Nice....26
Generating your Design....27
General Questions....28
Instantiating Into Your SmartDesign....29
Working with Processor-Based Designs in SmartDesign....31
Making your Design Look Nice....32

Getting Started with SmartDesign....33

Canvas View....37

Creating a SmartDesign....46

Connecting Instances....48

Bus Interfaces....52

Incremental Design....60

Reference....64

Designing with Designer Block Components....79

Creating a Designer Block Component in Libero SoC....82
Creating a Designer Block Component in Libero SoC....83

Place and Route....101

Device Programming....108

Generating Programming Files....133

TCL Command Reference....168
Introduction to Tcl Scripting....169

Project Manager Tcl Commands....189
set_root....230

Designer in Libero SoC....407

Preferences 421

Product Support 437

NOTE: Links and cross-references in this PDF file may point to external files and generate an error when clicked. **View the online help included with software to enable all linked content.**

Welcome to Microsemi's Libero® SoC v11.0

Libero SoC is the most comprehensive and powerful FPGA design and development software available, providing start-to-finish design flow guidance and support for novice and experienced users alike. Libero SoC combines Microsemi SoC Products Group (formerly Actel) tools with such EDA powerhouses as Synplify®, ModelSim®, and ViewDraw®.

What's New in Libero SoC v11.0

Support for SmartFusion2 devices - SmartFusion2 integrates an inherently reliable flash-based FPGA fabric, a 166 megahertz (MHz) ARM® Cortex™-M3 processor, advanced security processing accelerators, DSP blocks, SRAM, eNVM, and industry-required high-performance communication interfaces all on a single chip.

Libero SoC v11.0 includes new Pin Report options.

New for SmartFusion2-only updates to the:

- Design Flow, including I/O and Floorplan constraints
- SmartTime and batch mode Timing Verification
- I/O Constraints Editor
- Floorplanner
- System Builder
- Programming Connectivity and Interface
- Security Policy Manager (SPM)
- SmartDebug

Updated Family Support

Libero SoC v11.0 supports the following families and their derivatives:

- SmartFusion2
- SmartFusion
- Fusion
- IGLOO®
- ProASIC3

Design Flow - Libero SoC

See the Libero SoC SmartFusion2 Design Flow topic for more information on designing for that device.

The Libero SoC Build button ▶ enables you to proceed from synthesis to programming in one click.

Once you create your design (configure your MSS; create SmartDesign; Create HDL) and click the **Build** button the software automatically executes the following operations with default settings (if it encounters no errors):

- Synthesis
- Compile
- Place and Route
- Verify Timing
- Generate Programming Data

You can also import constraint files, organize and associate them for use during synthesis and compile.

In the event of an error the operation is halted and an explanatory error message appears in the Log window.

To change the default settings for any of the operations, right-click and choose **Open Interactively** to open the tool associated with the operation.

For example, to change the Compile settings, expand **Implement Design**, right-click **Compile** and choose **Open Interactively**. This displays the Compile options for your design.

Design Flow Window Updates for SmartFusion2 Only

The Design Flow window for the SmartFusion2 family has been changed in v11.0. Some functions, such as running SmartTime and the I/O Editor, no longer require that you open Designer.

When you move through the steps in the Design Flow window, only the steps in bold are required to complete and program your design. The bold steps are completed automatically if you use the Build button.

The table below summarizes the new or updated functions in the Design Flow window for the SmartFusion2 family.

Value	Function
Create Design > System Builder	System Builder creates your design based on high level design specifications by walking you through a set of high-level questions that will define your intended system.
Create Constraints	**I/O Constraints** - Import or edit I/O Constraint PDC files **Timing Constraints**- Import or edit Timing Constraint SDC files **Floorplan Constraints** - Import or edit Floorplan Constraint PDC files Note that I/O Constraint and Floorplan Constraint PDC files are handled separately in v11.0; if you have a PDC file that contains both I/O Constraints and FloorPlan Constraints then it will error out.
Configure Flash*Feeze	Enables you to configure your Flash*Freeze hardware settings
Edit Constraints	**I/O Constraints** - Opens the Post-Compile I/O Editor, enables you to edit the I/O Constraints PDC file you imported or created in Create Constraints (above). **Timing Constraints** - Opens SmartTime for SmartFusion2; enables you to create/edit your timing constraints (SDC files) **Floorplan Constraints** - Opens ChipPlanner for SmartFusion2; enables you to edit your Floorplan PDC files.
Generate Back Annotated Files	Similar to Export Back Annotated Files for other families, enables you to generate your Back Annotated files and/or set your options without opening Designer.
Generate Fabric Programming Data	Generates programming data for your design; this operation is completed automatically as the last step if you use the Build button.
Edit Design Hardware Configuration	Configures your Programing Connectivity, Programmer Settings and Device I/O States During Programming. These functions were previously available in FlashPro but are now managed from within Libero SoC.

Value	Function
Configure Security and Programming Options	Security Policy Manager - Sets the options for your Secured Programming Use Model, User Key Entry and Security Policies (Update Policy, Protocol Policy and Operational Integrity Policy) Programming Features - Enables you to select which features you wish to program. Update eNVM Memory Content - Enables you to change your eNVM content for programming without having to rerun Compile and Place and Route.

The figure below shows a SmartFusion Design Flow window on the left and a SmartFusion2 Design Flow window on the right.

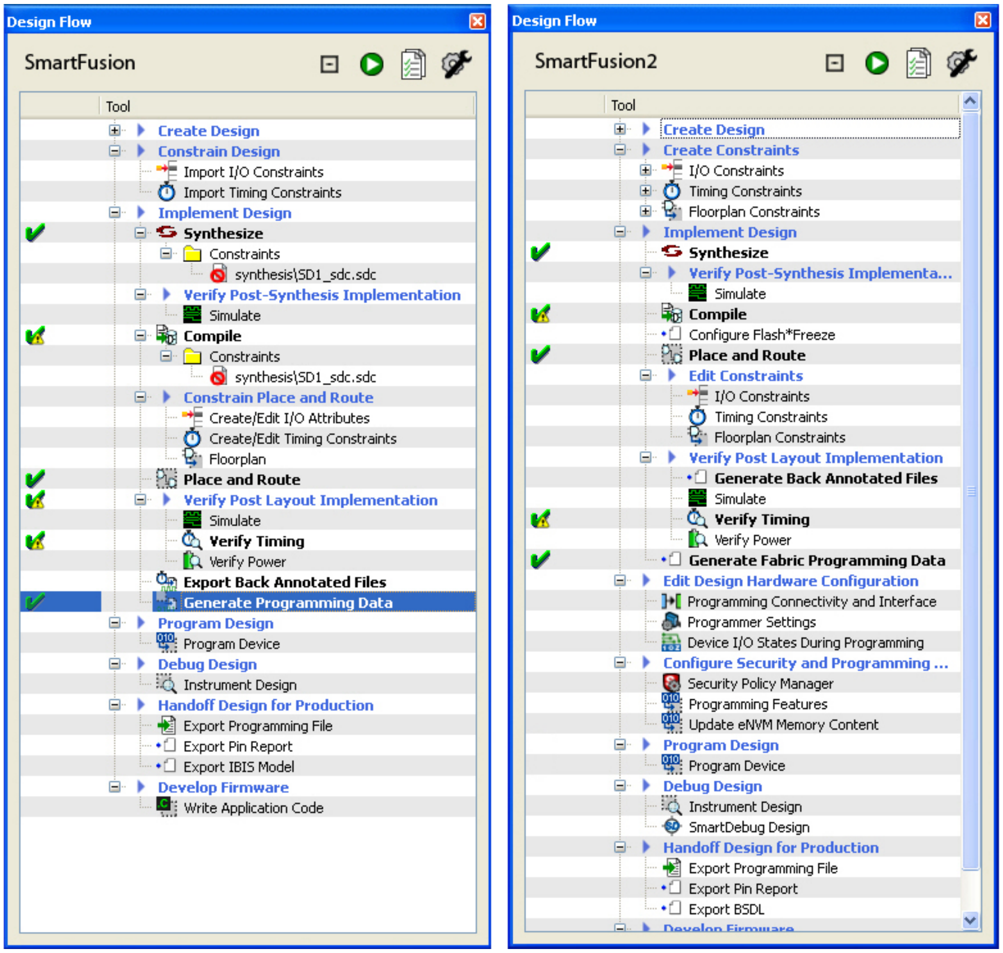


Figure 1 · SmartFusion (left) and SmartFusion2 (right) Design Flow Windows

Libero SoC Design Flow - SmartFusion2 ONLY

The Libero SoC v11.0 release incorporates several new features that are unique to SmartFusion2.

The Libero SoC Build button ▶ still enables you to proceed from synthesis to programming in one click (using default settings).

The basic design flow is shown in the figure below.

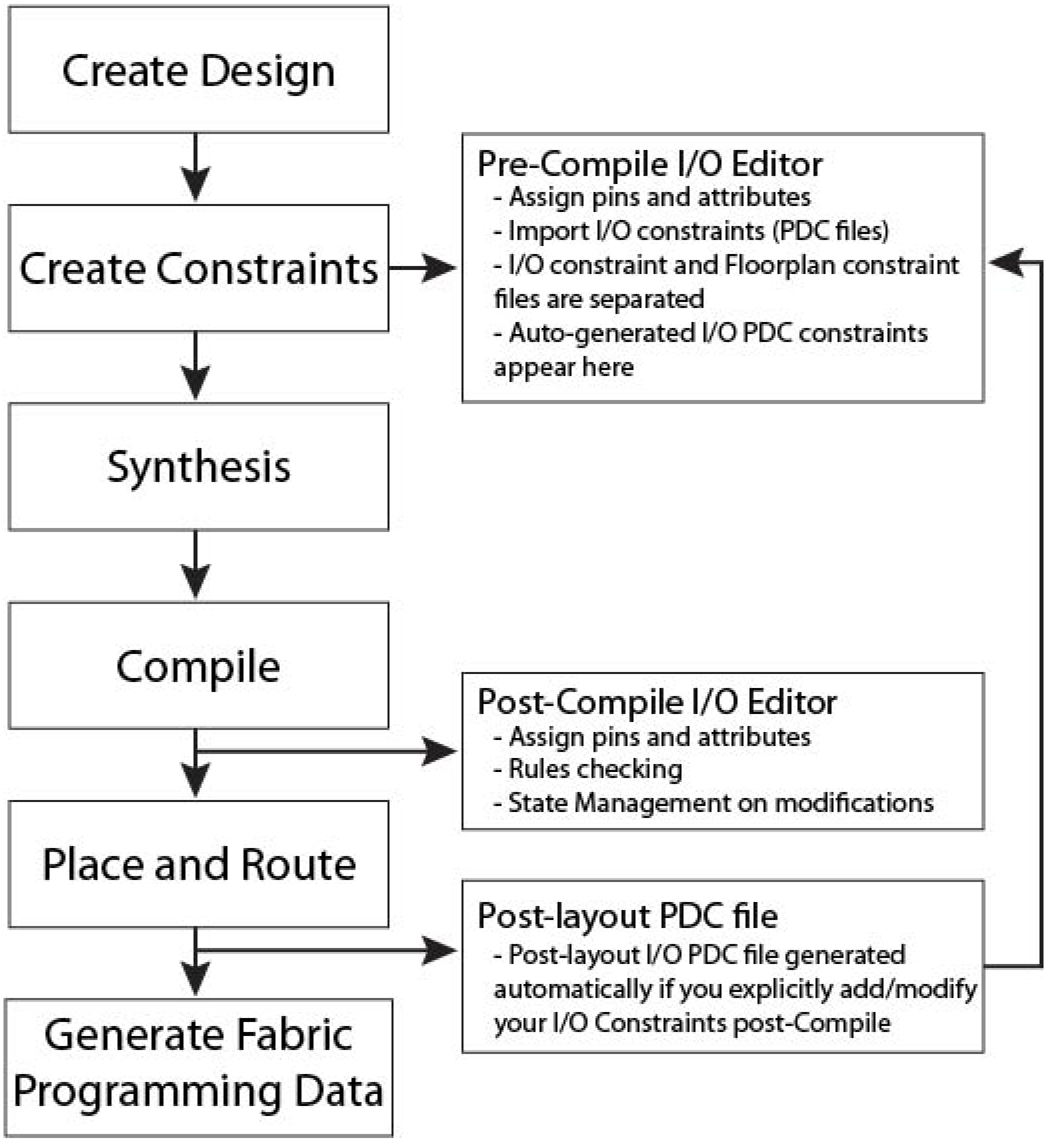

Create Design
Create Constraints
Synthesis
Compile
Place and Route
Generate Fabric Programming Data
Pre-Compile I/O Editor
- Assign pins and attributes
- Import I/O constraints (PDC files)
- I/O constraint and Floorplan constraint files are separated
- Auto-generated I/O PDC constraints appear here
Post-Compile I/O Editor
- Assign pins and attributes
- Rules checking
- State Management on modifications
Post-layout PDC file
- Post-layout I/O PDC file generated automatically if you explicitly add/modify your I/O Constraints post-Compile

Figure 2 · Design Flow for SmartFusion2

Create Design

Once you create your design (using System Builder; using the MSS builder - the flow for which is similar to the MSS flow for SmartFusion; create SmartDesign; Create HDL; SmartDesign Testbench) and click the **Build** button the software automatically executes the operations below with default settings (if it encounters no errors).

Create Constraints - Pre-Compile

SmartFusion2 I/O constraint PDC files are separate from Floorplan constraint PDC files; if you have a PDC file that contains both I/O and Floorplan constraints then Libero SoC errors out with an invalid constraint error.

- I/O Constraints - Created with the I/O Editor or text editor (pre-Compile). To add an I/O constraint, in the **Design Flow** window expand **Create Constraints**, right-click **I/O Constraints** and choose **Import Files**.
- Timing Constraints - Enables you to import SDC files pre-Compile.

- Floorplan Constraints - Created with the Floorplanner or a text editor; to add a Floorplan constraint, in the **Design Flow** window expand **Create Constraints**, right-click **Floorplan Constraints** and choose **Import Files**.

Synthesis

Double-click **Synthesize** to run synthesis on your design automatically; automatic synthesis uses the default settings in your synthesis tool.

Compile

To compile your design with custom settings, right-click **Compile** in the Design Flow window and choose **Configure Options**.

Place and Route - Post-Compile

Place and Route runs automatically with default settings as part of the push-button design flow in Libero SoC.

- I/O Editor - The Post-Compile I/O Editor displays all assigned and unassigned I/O macros and their attributes in a spreadsheet format; use this editor to view, sort, select, edit, lock and unlock assigned attributes.

  The post compile editor ensures that the Compile/Place and Route state is maintained (you do not have to rerun Compile or Place and Route), if you make changes to the attributes that do not require it.

  However, if you modify the I/O's in the Pre-Compile editor, it is equivalent to modifying the source file of the design, which means the tools starting from Compile will become out of date because one of the source files was modified.
- Timing Constraints - Run SmartTime to perform Min/Max analysis and manage timing constraints.
- Floorplan Constraints - Use to create and edit regions on your chip and assign logic to these regions.

Generate Fabric Programming Data

Generates programming data for your design. This operation is completed automatically as the last step if you use the Build button

Programming

You do not have to open FlashPro or FlashPoint to program your SmartFusion2 device. All programming functionality is available from within the Design Flow window, including:

Programming Connectivity and Interface - Organizes your programmer(s) and devices.

Programmer Settings - Opens your programmer settings; use if you wish to program using settings other than default.

Device I/O States During Programming - Sets your device I/O states during programming; use if your design requires that you change the default I/O states.

Security Policy Manager - Enables you to set your Secured Programming Use Model, User Key Entry and Security Policies for your design.

Supported Families

Microsemi's Libero SoC software supports the following families of devices:

- SmartFusion2
- SmartFusion
- Fusion
- IGLOO®
- ProASIC3

When we specify a family name, we refer to the device family and all its derivatives, unless otherwise specified. See the table below for a list of supported device families and their derivatives:

Table 1 · Product Families and Derivatives

Device Family	Family Derivatives	Description
SmartFusion2	SmartFusion2	Address fundamental requirements for advanced security, high reliability and low power in critical industrial, military, aviation, communications and medical applications.
SmartFusion	SmartFusion	SmartFusion intelligent mixed-signal FPGAs are the only devices that integrate an FPGA, ARM Cortex-M3, and programmable analog, offering full customization and IP protection.
Fusion	Fusion	Mixed-signal FPGA integrating ProASIC3 FPGA fabric, programmable analog block, support for ARM® Cortex^{TM}-M1 soft processors, and flash memory into a monolithic device.
IGLOO	IGLOO	The ultra-low-power, programmable solution
	IGLOOe	Higher density IGLOO FPGAs with six PLLs and additional I/O standards
	IGLOO nano	The industry’s lowest power, smallest size solution
	IGLOO PLUS	The low-power FPGA with enhanced I/O capabilities
ProASIC3	ProASIC3	The low-power, low-cost, FPGA solution
	ProASIC3E	Higher density ProASIC3 FPGAs with six PLLs and additional I/O standards
	ProASIC3 nano	Lowest cost solution with enhanced I/O capabilities
	ProASIC3L	The FPGA that balances low power, performance, and low cost
	Automotive ProASIC3	ProASIC3 FPGAs qualified for automotive applications

Device Family	Family Derivatives	Description
	Military ProASIC3/EL	Military temperature A3PE600L, A3P1000, and A3PE3000L
	RT ProASIC3	Radiation-tolerant RT3PE600L and RT3PE3000L

File Types in Libero SoC

When you create a new project in the Libero SoC it automatically creates new directories and project files. Your project directory contains all of your local project files. If you import files from outside your current project, the files must be copied into your local project folder. (The Project Manager enables you to manage your files as you import them.)

Depending on your project preferences and the version of Libero SoC you installed, the software creates directories for your project.

The top level directory (<project_name>) contains your PRJ file; only one PRJ file is enabled for each Libero SoC project.

component directory - Stores your SmartDesign components (SDB and CXF files) for your Libero SoC project.

constraint directory **-** All your constraint files (SDC, PDC, GCF, DCF, etc.)

designer directory **-** ADB files (Microsemi Designer project files), -_ba.SDF, _ba.v(hd), STP, PRB (for Silicon Explorer), TCL (used to run designer), impl.prj_des (local project file relative to revision), designer.log (logfile)

Note: Note: The Microsemi ADB file memory requirement is equivalent to 2x the size of the ADB file. If your computer does not have 2x the size of your ADB file's memory available, please make memory available on your hard drive.

hdl directory **-** all hdl sources. *.vhd if VHDL, *.v and *.h if Verilog

phy_synthesis directory **-** _palace.edn, _palace.gcf, palace_top.rpt (palace logfile) and other files generated by PALACE

simulation directory - meminit.dat, modelsim.ini files

smartgen directory - GEN files and LOG files from generated cores

stimulus directory - BTIM and VHD stimulus files

synthesis directory - *.edn, *_syn.prj (Synplify log file), *.psp (Precision project file), *.srr (Synplify logfile), precision.log (Precision logfile), *.tcl (used to run synthesis) and many other files generated by the tools (not managed by Libero SoC)

viewdraw directory - viewdraw.ini files

Software Tools - Libero SoC

The Libero SoC integrates design tools, streamlines your design flow, manages design and log files, and passes design data between tools.

For more information on Libero SoC tools, please visit: http://www.actel.com/products/software/libero/

Function	Tool	Company
Project Manager, HDL Editor, Core Generation	Libero SoC	Microsemi SoC
Schematic Capture	ViewDraw® ME	Mentor Graphics

Function	Tool	Company
Synthesis	Synplify® Pro ME	Synopsys
Simulation	Model*Sim*® ME	Mentor Graphics
Timing/Constraints, Power Analysis, NetlistViewer, Floorplanning, Package Editing, Place-and-Route, Debugging	Libero SoC	Microsemi SoC
Programming Software	FlashPro	Microsemi SoC

Project Manager, HDL Editor targets the creation of HDL code. HDL Editor supports VHDL and Verilog with color, highlighting keywords for both HDL languages.

Synplify Pro AE from Synopsys is integrated as part of the design package, enabling designers to target HDL code to specific devices.

Microsemi SoC software package includes:

- **ChipPlanner** displays I/O and logic macros in your design
- **NetlistViewer** design schematic viewer
- **SmartPower** power analysis tool
- **SmartTime** static timing analysis and constraints editor

Model*Sim* AE from Mentor Graphics enables source level verification so designers can verify HDL code line by line. Designers can perform simulation at all levels: behavioral (or pre-synthesis), structural (or post-synthesis), and back-annotated, dynamic simulation. (Model*Sim* is supported in Libero Gold, Platinum, and Platinum Eval only.)

Frequently Asked Questions - Libero SoC

The collection of Frequently Asked Questions are useful for anyone that is new to Libero SoC. All the information listed below is explained in detail in other sections of the help, but the information is summarized here for easy reference. Click any question to go to the corresponding explanation.

Libero SoC Frequently Asked Questions

1. How do I set my Multi-Pass place and route options?
2. How do I set FlashPro security options?
3. How do I instantiate my HDL in SmartDesign?
4. How do I add a bus interface to my HDL code and then add it to SmartDesign?
5. I don't see any DirectCore IP's in the Catalog but I have both Libero IDE 9.1 and Libero SoC 10.0 installed. Where are the DirectCore IP's?
6. How do I assign I/O/s in Libero SoC?
7. How do I make sure that my design is using the latest driver(s)?
8. How do I improve the timing of my design?
9. How do I manage clocks?
10. How do I write a testbench?

Firmware Cores Frequently Asked Questions

1. Where are the firmware files generated?
2. Why are some firmware in italics?
3. Why am I getting the following error on generation? "Error: 'Missing Core Definition': Core 'Actel:Firmware:MSS_SPI_Driver:2.0.101 ' is missing from the vault."?
4. Why is my firmware view empty?
5. Why are there multiple firmware instances of the same type?

Libero SoC Frequently Asked Questions

How do I set my Multi-Pass place and route options?

In the Design Flow window, expand **Implement Design**, right-click **Place and Route** and choose **Open Interactively**. Designer opens. Click **Layout** to open the Layout Options dialog box and choose your place and route options. Once Layout is complete, save your ADB to retain your custom place and route options.

How do I set FlashPro security options?

In the Libero SoC Design Flow window, expand **Program Design**, right-click **Program Device** and choose **Open Interactively**. FlashPro opens and enables you to set/change your security options. See the FlashPro help for more information.

How do I instantiate my HDL in SmartDesign?

Import your HDL file into the Libero SoC (File > Import Files). After you do this, your HDL module appears in the Project Manager Hierarchy. Then, drag-and-drop it from the Hierarchy onto your SmartDesign Canvas.

How do I add a bus interface to my HDL code and then add it to SmartDesign?

If you want to add a bus interface to your HDL code and then add it to SmartDesign, see the Adding or Modifying Bus Interfaces in SmartDesign topic.

I don't see any DirectCore IP's in the Catalog but I have both Libero IDE 9.1 and Libero SoC 10.0 installed. Where are the DirectCore IP's?

Make sure the vault location is correct. Click the Catalog Options button to open the Catalog Options dialog box. Then check and, if necessary, update your vault location.

How do I assign I/O's in Libero SoC?

In the Design Flow window, expand **Implement Design**, then expand **Constrain Place and Route**. Right-click **Edit I/O Attributes** and choose **Open Interactively** to open the I/O Attribute Editor.

How do I make sure that my design is using the latest driver(s)?

In the Design Flow tab, expand **Create Design** and double-click **View/Configure Firmware Cores** to view the DESIGN_FIRMWARE tab. The Firmware table lists the compatible firmware and drivers based on the hardware peripherals that you have used in your design. Use the Version drop down menus to check for the latest firmware and firmware drivers.

How do I improve the timing of my design?

The SmartTime tool enables you to set clock constraints, analyze timing, identify critical paths, and find the minimum cycle time that does not result in a timing violation.

To improve the timing of your design:

1. Run timing analysis to identify timing violations.
2. View the paths with timing violations.
3. Modify timing constraints on the critical path(s) in order to meet your timing requirements.
4. Run Timing-Driven Place and Route.

For more information on improving timing, see the Analysis and Optimization application notes. The Designing for Performance on Flash-Based FPGAs application note is a good starting point.

How do I manage clocks?

Specify clock constraints in your design. See the sections on explicit clocks, potential clocks and clock networks for more information on clocks in Libero SoC.

How do I write a testbench?

You can write or edit a testbench manually using the HDL editor, or you can create a new HDL testbench and automatically populate it with all your design information with Create New HDL Testbench in Libero SoC. Create New HDL Testbench is in the Design Flow window under **Create Design**.

Testbench file are generated automatically when you generate a SmartDesign. You can find them in your Files window in Libero SoC (**View > Window > Files**).

Firmware Cores Frequently Asked Questions

Where are the firmware files generated?

The firmware files are generated to the firmware working directory <project>\firmware. Your software IDE workspace is generated to <project>\<software IDE tool chain>.

Why are some firmware in italics?

This indicates the firmware is in the IP repository but not in your local IP vault. You must download it to your local IP vault so that the Libero SoC will generate the firmware files.

Why am I getting the following error on generation? "Error: 'Missing Core Definition': Core 'Actel:Firmware:MSS_SPI_Driver:2.0.101 ' is missing from the vault."?

This happens when a firmware that is in your design but the VLNV definition could not be found in your IP vault. This can happen if you:

- Changed your vault settings to point to another vault
- Opened a project that was created on another machine

Why is my firmware view empty?

Check that you are pointing to the proper firmware repository:

www.actel-ip.com/repositories/Firmware

Check with your network administrator to make sure you can communicate with Microsemi's IP repository URL.

Why are there multiple firmware instances of the same type?

Some firmware cores have configurable options, and in certain cases you will have two peripherals of the same firmware VLNV. In this situation, you may want to configure each peripheral driver separately.

Software IDE Integration

Libero SoC simplifies the task of transitioning between designing your FPGA to developing your embedded firmware.

Libero SoC manages the firmware for your FPGA hardware design, including:

- Firmware hardware abstraction layers required for your processor
- Firmware drivers for the processor peripherals that you use in your FPGA design.
- Sample application projects are available for drivers that illustrate the proper usage of the APIs

You can see which firmware drivers Libero SoC has found to be compatible with your design by opening the Firmware View. From this view, you can change the configuration of your firmware, change to a different version, read driver documentation, and generate any sample projects for each driver.

Libero SoC manages the integration of your firmware with your preferred Software Development Environment, including SoftConsole, Keil, and IAR Embedded Workbench. The projects and workspaces for your selected development environment are automatically generated with the proper settings and flags so that you can immediately begin writing your application.

See Also

Develop Firmware - Write Application Code

Libero SoC Frequently Asked Questions

Running Libero SoC from your Software Tool Chain

View/Configure Firmware Cores

System Builder

System Builder is a graphical design wizard designed specifically for SmartFusion2 based systems. System Builder walks you through the following steps:

- Asks basic questions about your system architecture
- Adds any additional needed peripherals in the fabric
- Sets required configuration options for each selected feature
- Builds a correct-by-design complete system

The SmartFusion2 System Builder wizard creates your design based on high level design specifications by walking you through a set of high-level questions that will define your intended system. System Builder enables you to focus on your design specializations instead of on the specific silicon requirements of a SmartFusion2 based design.

This simplifies the design creation process. The built-in design rule check feature prevents you from moving forward if there are mistakes or conflicts. The design that is produced by the System Builder follows all the SmartFusion2 silicon design rules.

You can also extend the System Builder generated design with your own custom peripherals and logic by specifying your options and then using SmartDesign to connect up your custom peripherals.

System Builder supports the SmartFusion2 family

See the System Builder documentation for a complete explanation of the tool.

SmartFusion Design Flow Overview

SmartFusion designs can be implemented from within the Libero SoC using the Microcontroller Subsystem (MSS) configurator or can be configured in your software IDE, such as Keil or IAR.

The Microcontroller Subsystem (MSS) Configurator (as shown in the figure below) is a specialized SmartDesign that enables you to configure and implement a SmartFusion design. The SmartDesign MSS Configurator enables you to configure your SmartFusion microcontroller hardware as well as produces the necessary firmware drivers for your design.

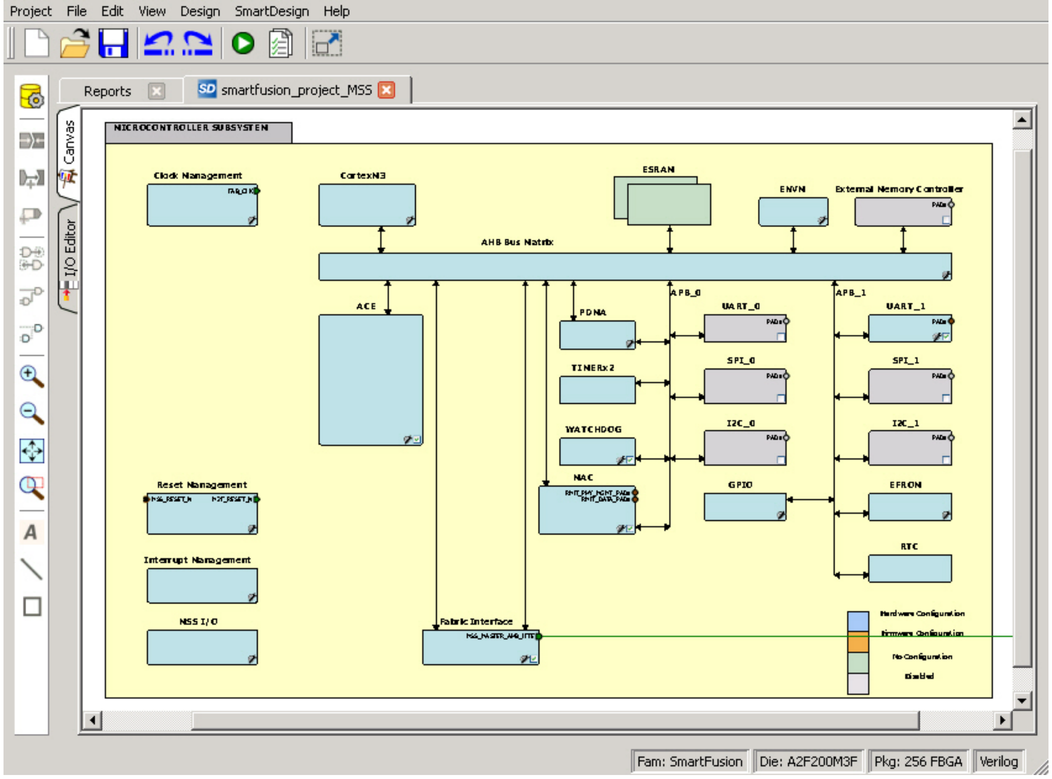


Figure 3 · Microcontroller Subsystem Configurator

The following is a high level overview of the SmartFusion design flow. Further details can be found directly inside the SmartDesign MSS Configurator tool. See the Additional User Support section (below) for information on where you can access additional resources.

SmartFusion in the Libero SoC: Design Flow

1. Create a SmartFusion project
2. Configure SmartFusion MSS peripherals
3. Generate SmartFusion files
4. Complete your design (simulation, synthesis, compile, place-and-route, and programming)

Additional User Support

Information regarding the MSS such as tutorials, simple how-to descriptions, design flows, Frequently Asked Questions, and videos can be accessed from the SmartDesign MSS Configurator Help menu under the sub-menu Microcontroller Subsystem (as shown in the figure below).

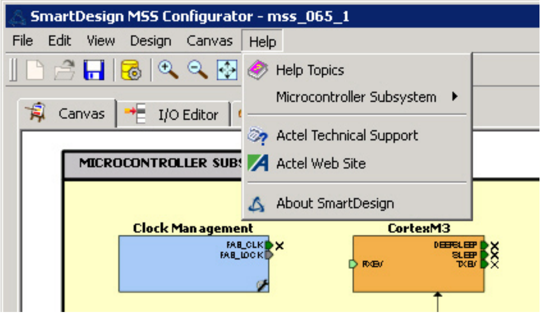
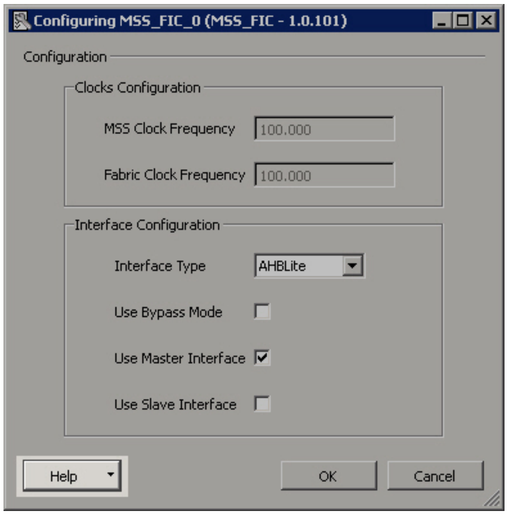


Figure 4 · Microcontroller Subsystem Help

Specific information regarding the MSS peripherals and their use is available inside the configurator of each peripheral. To access these resources, double-click the configurator to open it, then click the Help button (as shown in the figure below).



Configuring MSS_FIC_0 (MSS_FIC - 1.0.101)

Configuration

Clocks Configuration

MSS Clock Frequency 100.000

Fabric Clock Frequency 100.000

Interface Configuration

Interface Type AHBLite

Use Bypass Mode □

Use Master Interface ☑

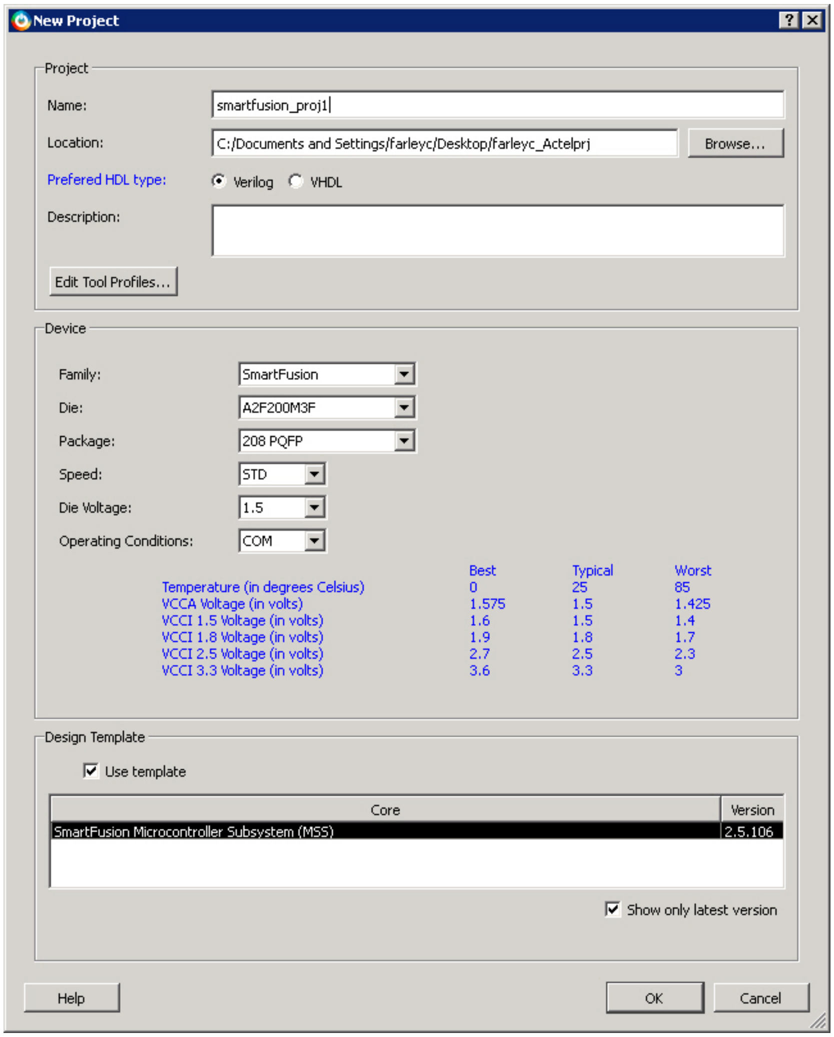
Use Slave Interface □

Figure 5 · Peripheral Help Button

Create a SmartFusion Project

Creating a SmartFusion design using the Microcontroller Subsystem (MSS) is similar to creating any other project in the Libero SoC:

1. In the Project Manager, create a new project.
2. Enter your Project Name and Location and select your Preferred HDL type.
3. Choose SmartFusion as the family and select your Device settings: Family, Die, Package, Speed, Die Voltage, and Operating Conditions (as shown in the figure below).



Project

Name: smartfusion_proj1

Location: C:/Documents and Settings/farleyc/Desktop/farleyc_Actelprj

Prefered HDL type: ● Verilog ○ VHDL

Description:

Device

Family: SmartFusion

Die: A2F200M3F

Package: 208 PQFP

Speed: STD

Die Voltage: 1.5

Operating Conditions: COM

	Best	Typical	Worst
Temperature (in degrees Celsius)	0	25	85
VCCA Voltage (in volts)	1.575	1.5	1.425
VCCI 1.5 Voltage (in volts)	1.6	1.5	1.4
VCCI 1.8 Voltage (in volts)	1.9	1.8	1.7
VCCI 2.5 Voltage (in volts)	2.7	2.5	2.3
VCCI 3.3 Voltage (in volts)	3.6	3.3	3

Design Template

☑ Use template

Core	Version
SmartFusion Microcontroller Subsystem (MSS)	2.5.106

☑ Show only latest version

Figure 6 · New Project: SmartFusion

4. Configure the MSS, firmware and I/Os according to your design specification.

Note: Note: If you opt not to use a Design Template and create the MSS, expand Create Design in the Design Flow tab and double-click Configure MSS to create an MSS at any time. If you already have an MSS in your project, then this button opens your existing MSS component.

Configure SmartFusion Microcontroller Subsystem (MSS) Peripherals

The SmartDesign MSS Configurator enables you to configure your microcontroller peripherals, such as the ACE, GPIO, and External Memory Controller.

Peripherals that have configurable options are shown with a wrench icon on the instance (as shown in the figure below). Clicking the wrench icon opens the configuration dialog box for the peripheral.

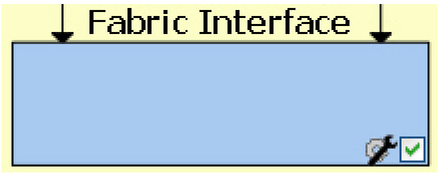
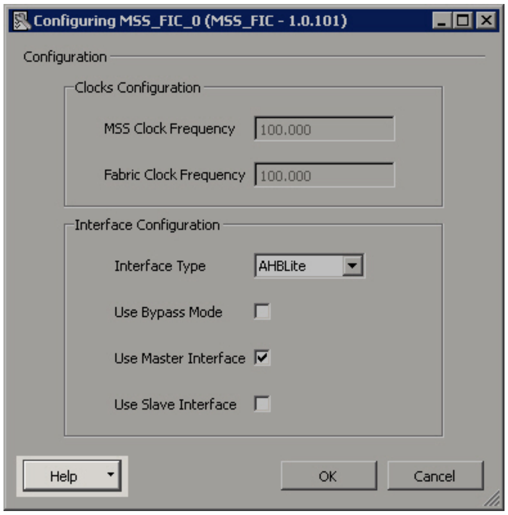


Figure 7 · Fabric Interface Peripheral in the MSS

For more details on the configuration options of each peripheral, click the Help button in the peripheral configuration dialog box (as shown in the figure below).



Configuring MSS_FIC_0 (MSS_FIC - 1.0.101)

Configuration

Clocks Configuration

MSS Clock Frequency: 100.000

Fabric Clock Frequency: 100.000

Interface Configuration

Interface Type: AHBLite

Use Bypass Mode □

Use Master Interface ☑

Use Slave Interface □

Figure 8 · Help Button in Fabric Interface Peripheral

Generate SmartFusion Files

See the MSS Configurator help for more information on generating SmartFusion files.

Click the **Generate Component** button to create your SmartFusion files.

The MSS Configurator generates the following files:

- HDL files for the MSS design and its sub-components: MSS CCC, etc. HDL files are automatically managed by the Libero SoC and passed to the Synthesis and Simulation point tools.
- EFC File. MSS hardware configuration that is loaded into eNVM. FlashPro automatically detects this file and includes it in your final programming file.
- UFC file. This file contains the Embedded FlashROM configuration and data: FlashPro automatically detects this file and includes it in your final programming file.
- Firmware drivers and memory maps are exported into the <project>\firmware\ directory. These files can be imported into your software IDE to begin the software part of your design.
- Testbench HDL and BFM script for the MSS design: These files are also managed by Libero SoC and automatically passed to the Simulation point tool.
- PDC files for the MSS and the top-level design: These files are managed by Libero SoC and automatically integrated during Compile and Layout.

Completing a Design Using the Libero SoC Tool Suite

After the MSS design has been successfully generated, you can complete your design using the Libero SoC design tools.

Synthesis, Compile, Place and Route and Generate Programming Files all run automatically with default settings as part of the Libero SoC push-button design flow. If you wish to run any of these operations with different settings, view them in the Design Hierarchy window, right-click and choose **Open Interactively**. The tool opens and enables you to change the settings before you simulate / compile / place-and-route.

For example, you can set timing constraints in **Implement Design > Constraint Place and Route > Edit Timing Constraints** (double-click to open SmartTime).

Programming

FlashPro automatically detects the presence of the FlashPro data (FDB) file and the MSS Hardware Files (EFC and UFC) produced by the SmartDesign MSS Configurator.

Once you open FlashPro, you can view the input files that will be used to create the programming data file (PDB), by clicking the **Modify** button in the Programming file panel. The Modify button starts the FlashPoint dialog box.

The FlashPoint dialog box enables you to modify the content of the FlashROM (UFC), FPGA Array (FDB) and Embedded Flash Memory (EFC) directly from FlashPro. For example, you may wish to update an NVM data storage client that has been set up to load the MSS application data. See the FlashPro help for a tutorial on programming SmartFusion.

Create ViewDraw Schematic

You must enable ViewDraw in your Project Settings to create a schematic source file in Libero SoC.

To create a schematic source file:

1. In the Design Flow window, double-click **Create ViewDraw Schematic**.
2. Type a name for your schematic file in the Name field. Click **OK**. ViewDraw AE starts.
3. **Using ViewDraw AE, create your schematic**.
4. When you are done, click **Save+Check** in ViewDraw. The Save+Check command creates your WIR file. When Save and Check is complete, the message Check complete, 0 errors and 0 warnings in project <name> appears in the status bar.

   You must select Save & Check. Selecting Save will not generate the needed WIR file for that block.

5. (Optional) **Right-click the schematic file** in the Files tab and choose **Check Schematic**. The connectivity checker checks the connectivity of the WIR file. Errors and warnings appear in the log window.
6. From the **File** menu, choose **Exit**. The schematic is saved to your project in Libero SoC and appears in both the File Manager and the Design Hierarchy tabs.

About SmartDesign

SmartDesign enables you to take configured cores, IP cores, macros from a Catalog, and user-created HDL source files and instantiate them into your design.

You can drag configured cores onto a Canvas where they are viewed as blocks in a functional block diagram. From the Canvas you can:

- Make connections between your blocks
- View individual connection details
- Show or hide individual nets
- Set or clear attributes (such as Invert, Tie Low, Tie High, or Tie Open)
- Add slices
- Move, duplicate, or delete blocks
- Add notations such as labels, shapes, lines, or arrows to document your design
- Auto-stitch interfaces and other connections (such as AMBA)
- View a Memory Map / Datasheet - The datasheet reports the memory map of the different subsystems of your design, where a subsystem is any independent bus structure with a Master and Slave peripheral attached.

SmartDesign supports all Microsemi SoC product families.

SmartDesign Design Flow

SmartDesign enables you to stitch together design blocks of different types (HDL, IP, etc) and generate a top-level design. The Files tab lists your SmartDesign files in alphabetical order.

You can build your design using SmartDesign with the following steps:

Step One – Instantiating components: In this step you add one or more building blocks, HDL modules, components, and schematic modules from the project manager to your design. The components can be blocks, cores generated from the core Catalog, and IP cores.

Step Two – Connecting bus interfaces: In this step, you can add connectivity via standard bus interfaces to your design. This step is optional and can be skipped if you prefer manual connections. Components generated from the Catalog may include pre-defined interfaces that allow for automatic connectivity and design rule checking when used in a design.

Step Three – Connecting instances: The Canvas enables you to create manual connections between ports of the instances in your design. Unused ports can be tied off to GND or VCC (disabled); input buses can be tied to a constant, and you can leave an output open by marking it as unused.

Step Four – Generating the SmartDesign component: In this step, you generate a top-level (Top) component and its corresponding HDL file. This component can be used by downstream processes, such as synthesis and simulation, or you can add your SmartDesign HDL into another SmartDesign.

When you generate your SmartDesign the Design Rules Check verifies the connectivity of your design; this feature adds information to your report; design errors and warnings are organized by type and message and displayed in your Datasheet / Report.

You can save your SmartDesign at any time.

Using Existing Projects with SmartDesign

You can use existing Libero SoC projects with available building blocks in the project to assemble a new SmartDesign design component. You do not have to migrate existing top-level designs to SmartDesign and there is no automatic conversion of the existing design blocks to the SmartDesign format.

SmartDesign Frequently Asked Questions

The collection of SmartDesign Frequently Asked Questions are useful for anyone that is new to SmartDesign. All the information listed below is explained in detail in other sections of the help, but the information is summarized here for easy reference. Click any question to go to the corresponding explanation.

General Questions

1. What is SmartDesign?
2. How do I create my first SmartDesign?

Instantiating into your SmartDesign

1. Where is the list of cores that I can instantiate into my SmartDesign?
2. How do I instantiate cores into my SmartDesign?
3. I have a block that I wrote in VHDL (or Verilog), can I use that in my SmartDesign?
4. My HDL module has Verilog parameters or VHDL generics declared; how can I configure those in SmartDesign?

Working in SmartDesign

1. How do I make connections?
2. Auto Connect didn’t connect everything for me; how do I make manual connections?
3. How do I connect a pin to the top level?
4. Oops, I just made a connection mistake. How do I disconnect two pins?
5. I need to apply some simple ‘glue’ logic between my cores. How do I do that?
6. My logic is a bit more complex than inversion and tie offs - what else can I do?
7. How do I create a new top level port for my design?
8. How do I rename one of my instances?
9. How do I rename my top level port?
10. How do I rename my group pins?
11. I need to reconfigure one of my Cores, can I just double click the instance?
12. I want more Canvas space to work with!

Working with Processor-Based Designs in SmartDesign

1. How do I connect my peripherals to the bus?
2. How do I view the Memory Map of my design?
3. How do I simulate my processor design?
4. I have my own HDL block that I want to connect as a peripheral on the AMBA bus. How can I do that?
5. How do I generate the firmware drivers for my design?
6. How do I start writing my application code for my design?

Making your Design Look Nice

1. Can the tool automatically place my instances on the Canvas to make it look nice?
2. My design has a lot of connections, and the nets are making my design hard to read. What do I do?
3. My instance has too many pins on it, how can I minimize that?
4. Oops, I missed one pin that needs to be part of that group? How do I add a pin after I already have the group?
5. I have a pin that I don’t want inside the group, how do I remove it?
6. How can I better see my design on the Canvas?

Generating your Design

1. Ok, I'm done connecting my design, how do I 'finish' it so that I can proceed to synthesis?
2. I get a message saying it's unable to generate my SmartDesign due to errors, what do I do? What is the Design Rules Check?
3. How do I generate my firmware? Software IDE?

General Questions

What is SmartDesign?

SmartDesign is a design entry tool. It’s the first tool in the industry that can be used for designing System on a Chip designs, custom FPGA designs or a mixture of both types in the same design. A SmartDesign can be the entire FPGA design, part of a larger SmartDesign, or a user created IP that can be stored and reused multiple times. It’s a simple, intuitive tool with powerful features that enables you to work at the abstraction level at which you are most comfortable.

It can connect blocks together from a variety of sources, verify your design for errors, manage your memory map, and generate all the necessary files to allow you to simulate, synthesize, and compile your design.

How do I create my first SmartDesign?

In the Libero SoC Project Manager Design Flow window, under Create Design, double-click **Create SmartDesign**.

Instantiating Into Your SmartDesign

Where is the list of Cores that I can instantiate into my SmartDesign?

The list of available cores is displayed in the Project Manager Catalog. This catalog contains all DirectCore IP, Design Block cores, and macros.

How do I instantiate cores into my SmartDesign?

Drag and drop the core from the Catalog onto your SmartDesign Canvas. An instance of your Core appears on the Canvas; double-click to configure it.

I have a block that I wrote in VHDL (or Verilog), can I use that in my SmartDesign?

Yes! Import your HDL file into the Project Manager (File > Import Files). After you do this, your HDL module will appear in the Project Manager Hierarchy. Then, drag-and-drop it from the Hierarchy onto your SmartDesign Canvas.

My HDL module has Verilog parameters or VHDL generics declared, how can I configure those in SmartDesign?

If your HDL module contains configurable parameters, you must create a ‘core’ from your HDL before using it in SmartDesign. Once your HDL module is in the Project Manager Design Hierarchy, right-click it and choose **Create Core from HDL**. You will then be allowed to add bus interfaces to your module if necessary. Once this is complete, you can drag your new HDL+ into the SmartDesign Canvas and configure your parameters by double-clicking it.

Working in SmartDesign

How do I make connections?

Let SmartDesign do it for you. Right-click the Canvas and choose **Auto Connect**.

Auto Connect didn’t connect everything for me, how do I make manual connections?

Enter **Connection Mode** and click and drag from one pin to another. Click the Connection Mode button in the Canvas to enter Connection Mode.

Alternatively:

1. Select the pins you want connected by using the mouse and the CTRL key.
2. Right-click one of the selected pins and choose **Connect**.

How do I connect a pin to the top level?

Right-click the pin and choose **Promote to Top Level**. You can even do this for multiple pins at a time, just select all the pins you want to promote, right-click one of the pins and choose **Promote to Top Level**. All your selected pins will be promoted to the top level.

Oops, I just made a connection mistake. How do I disconnect two pins?

Use CTRL+Z to undo your last action. If you want to undo your ‘undo’, hit redo (CTRL+Y).

To disconnect pins you can:

- Right-click the pin you want to disconnect and choose **Disconnect**
- Select the net and hit the delete key

I need to apply some simple ‘glue’ logic between my cores. How do I do that?

For basic inversion of pins, you can right-click a pin and choose **Invert**. An inverter will be placed at this pin when the design is generated. You can also right-click a pin and choose Tie Low or Tie High if you want to connect the pin to either GND or VCC.

To tie an input bus to a constant, right-click the bus and choose **Tie to Constant**. To mark an output pin as unused, right-click the pin and choose **Mark as Unused**.

To clear these, just right-click on the pin again and choose **Clear Attribute**.

My logic is a bit more complex than inversion and tie offs - what else can I do?

You have full access to the library macros, including AND, OR, and XOR logic functions. These are located in the Project Manager Catalog, listed under Actel Macros. Drag the logic function you want onto your SmartDesign Canvas.

How do I create a new top level port for my design?

Click the **Add Port** button in the Canvas toolbar

How do I rename one of my instances?

Double-click the instance name on the Canvas and it will become editable. The instance name is located directly above the instance on the Canvas.

How do I rename my top level port?

Right-click the port you want to rename and choose **Modify Port**.

How do I rename my group pins?

Right-click the group pin you want to rename and choose **Rename Group**.

I need to reconfigure one of my Cores, can I just double-click the instance?

Yes.

I want more Canvas space to work with!

Maximize your workspace (CTRL-W), and your Canvas will maximize within the Project Manager. Hit CTRL-W again if you need to see your Hierarchy or Catalog.

Working with Processor-Based Designs in SmartDesign

How do I connect my peripherals to the bus?

Click **Auto Connect** and it will help you build your bus structure based on the processor and peripherals that you have instantiated.

But I need my peripheral at a specific address or slot.

Right-click the Canvas and choose **Modify Memory Map** to invoke the Modify Memory Map dialog that enables you to set a peripheral to a specific address on the bus.

The bus core will show the slot numbers on the bus interface pins. These slot numbers correspond to a memory address on the bus.

Verify that your peripheral is mapped to the right bus address by viewing your design's Memory Map.

How do I view the Memory Map of my design?

Generate your project and open datasheet in the **Report View**.

The memory map section will also show the memory details of each peripheral, including any memory mapped registers.

How do I simulate my processor design?

SmartDesign automatically generates the necessary Bus Functional Model (BFM) scripts required to simulate your processor based design. A top level testbench for your SmartDesign is generated automatically as well.

Create your processor design, generate it, and you will be able to simulate it in ModelSim.

I have my own HDL block that I want to connect as a peripheral on the AMBA bus. How can I do that?

SmartDesign supports automatic creation of data driven configurators based on HDL generics/parameters.

If your block has all the necessary signals to interface with the AMBA bus protocol (ex: address, data, control signals):

1. Right-click your custom HDL block and choose **Create Core from HDL**. The Libero SoC creates your core and asks if you want to add bus interfaces.
2. Click **Yes** to open the Edit Core Definition dialog box and add bus interfaces. Add the bus interfaces as necessary.
3. Click **OK** to continue.

Now your instance has a proper AMBA bus interface on it. You can manually connect it to the bus or let Auto Connect find a compatible connection. See the DirectCore Advanced Microcontroller Bus Architecture - Bus Functional Model User's Guide for more information on CoreAMBA BFM commands.

How do I generate the firmware drivers for my design?

SmartDesign automatically finds all the compatible firmware drivers based on your peripherals and processor. You can view the list of firmware drivers that the design found by going to the design flow and choosing **View/Configure Firmware Cores**.

How do I start writing my application code for my design?

Libero SoC simplifies the embedded development process by automatically creating the workspace and project files for the Software IDE that you specify in the Tools profile.

Once you have generated your design, the firmware and workspace files will automatically be created. Click **Write Application Code** in the Design Flow tab and the Software IDE tool will open your design's workspace files.

Making your Design Look Nice

Can the tool automatically place my instances on the Canvas to make it look nice?

Yes. Right-click the Canvas white space and choose **Auto Arrange Instances**.

My design has a lot of connections, and the nets are making my design hard to read. What do I do?

You can disable the display of the nets in the menu bar (Canvas > Nets). This automatically hides all the nets in your design.

You can still see how pins are connected by selecting a connected pin, the net will automatically be visible again.

You can also selectively show certain nets, so that they are always displayed, just right click on a connected pin and choose **Show Net**.

My instance has too many pins on it; how can I minimize that?

Try grouping functional or unused pins together. For example, on the CoreInterrupt there are 8 FIQSource* and 32 IRQSource* pins, group these together since they are similar in functionality.

To group pins: Select all the pins you want to group, then right-click one of the pins and choose **Add pins to group**.

If a pin is in a group, you are still able to use it and form connections with it. Expand the group to gain access to the pin.

Oops, I missed one pin that needs to be part of that group? How do I add a pin after I already have the group?

Select the pin you want to add and the group pin, right-click and choose **Add pins to <name> group**.

I have a pin that I don't want inside the group, how do I remove it?

Right-click the pin and choose **Ungroup selected pins**.

How can I better see my design on the Canvas?

There are zoom icons in the Canvas toolbar. Use them to Zoom in, Zoom out, Zoom to fit, and Zoom selection. You can also maximize your workspace with CTRL-W.

Generating your Design

Ok, I'm done connecting my design, how do I 'finish' it so that I can proceed to synthesis?

In the Canvas toolbar, click the Generate Project icon .

I get a message saying it's unable to generate my SmartDesign due to errors, what do I do? What is the Design Rules Check?

The Design Rules Check is included in your Report View. It lists all the errors and warnings in your design, including unconnected input pins, required pin connections, configuration incompatibilities between cores, etc.

Errors are shown with a small red stop sign and must be corrected before you can generate; warnings may be ignored.

What does this error mean? How do I fix it?

Review the Design Rules Check topic for an explanation of errors in the Design Rules Check and steps to resolve them.

How do I generate my firmware? Software IDE?

In the Design Flow window, expand **Create Design** and double-click **View/Configure Firmware Cores**.

The Software IDE workspace is produced if have selected a Software IDE in your Tools Profile. Once this has been set you will be able to click Develop Firmware – Write Application Code.

Getting Started with SmartDesign

Creating a New SmartDesign Component

1. From the **File** menu, choose **New > SmartDesign** or in the Design Flow window double-click **Create SmartDesign**. The **Create New SmartDesign** dialog box opens (see figure below).

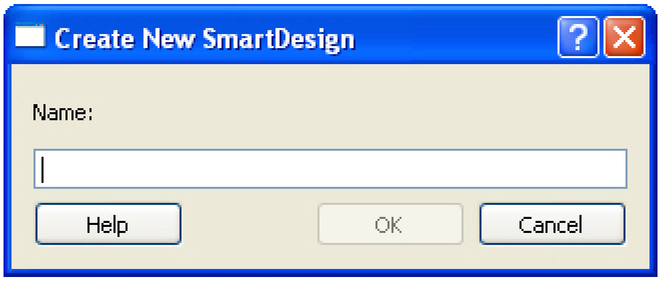


Figure 9 · Create New SmartDesign Dialog Box

2. Enter a component name and click **OK**. The component appears in the Hierarchy tab of the Design Explorer. Also, the main window displays the design Canvas.

Note: Note: The component name must be unique in your project.

Opening an Existing SmartDesign Component

To open an existing component do one of the following:

Click the **Design Hierarchy** tab and double-click the component you want to open.

The main window displays the SmartDesign Canvas for the SmartDesign component.

Saving/Closing a SmartDesign Component

To save the current SmartDesign design component, from the **File** menu, choose **Save** <component_name>. Saving a SmartDesign component only saves the current state of the design; to generate the HDL for the design refer to Generating a SmartDesign component.

To close the current SmartDesign component without saving, from the **File** menu, choose **Close**. Select **NO** when prompted to save.

To save the active SmartDesign component with a different name use Save As. From the **File** menu choose **Save SD_<filename> As**. Enter a new name for your component and click **OK**.

Generating a SmartDesign Component

Before your SmartDesign component can be used by downstream processes, such as synthesis and simulation, you must generate it.

Click the Generate button to generate a SmartDesign component.

This will generate a HDL file in the directory <libero_project>/components/<library>/<yourdesign>.

Note: Note: The generated HDL file will be deleted when your SmartDesign design is modified and saved to ensure synchronization between your SmartDesign component and its generated HDL file.

Generating a SmartDesign component may fail if there are any DRC errors. DRC errors must be corrected before you generate your SmartDesign design.

Generating a Datasheet

If your SmartDesign is the root design in your project, then a Memory Map / Datasheet is produced that contains the information for your design.

Generating Firmware and Software IDE Workspace

If your SmartDesign is the root design in your project, then any compatible firmware drivers for your peripherals are generated to <project>/firmware. Furthermore, if you have specified a Software IDE tool in your profile, then the workspace and projects for that Software IDE are generated into <project>/<SoftwareIDE>.

The datasheet provides all the specifics of the generated firmware drivers and Software IDE workspaces.

Importing a SmartDesign Component

From the **File** menu, choose **Impor**t and select the CXF file type.

Importing an existing SmartDesign component into a SmartDesign project will not automatically import the sub-components of that imported SmartDesign component.

You must import each sub-component separately.

After importing the sub-components, you must open the SmartDesign component and replace each sub-component so that it references the correct component in your project. .

Deleting a SmartDesign Component from the Libero SoC Project

To delete a SmartDesign component from the project:

1. In the **Design Hierarchy** tab, select the SmartDesign component that you want to delete.
2. Right-click the component name and select **Delete from Project** or **Delete from Disk and Project**, or click the **Delete** key to delete from project.

Memory Maps / Data Sheet

If your design contains standard Bus Instances such as the DirectCore AMBA bus cores, CoreAPB or CoreAHB, then you can view the Memory Map Configuration of your design in the Report View. To do so, generate your top level design and click the Reports button in the toolbar.

The design’s memory map is determined by the connections made to the bus component. A bus component is divided into multiple slots for slave peripherals or instances to plug into. Each slot represents a different address location and range to the Master of the bus component.

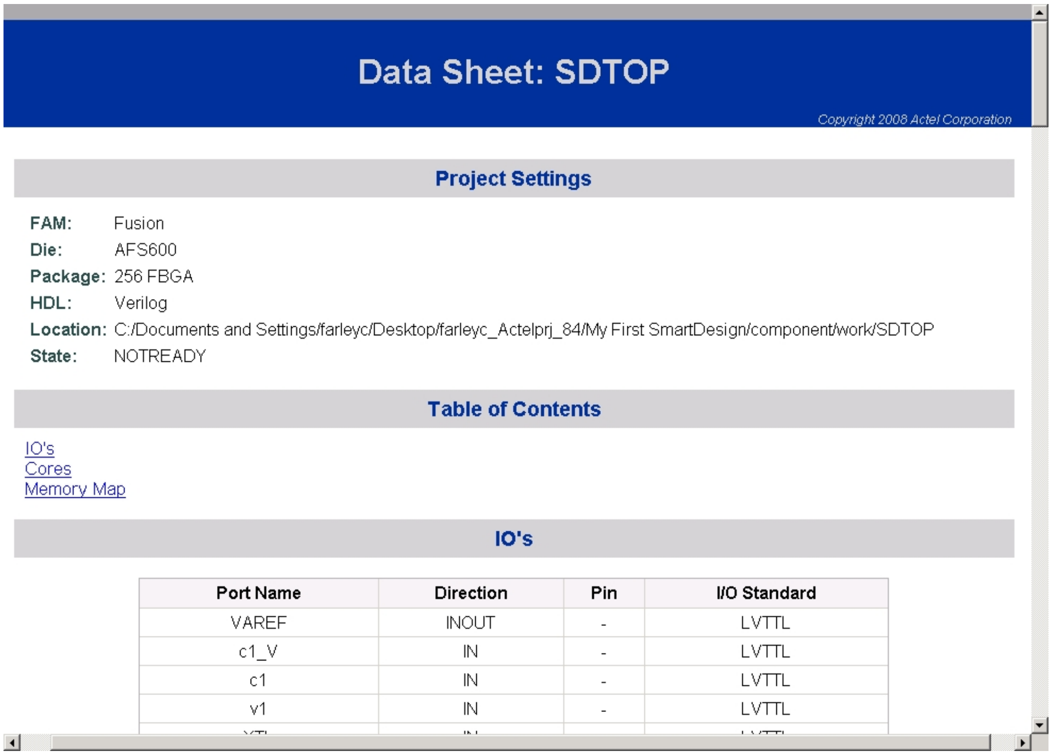
The datasheet reports the memory map of the different subsystems of your design, where a subsystem is any independent bus structure with a Master and Slave peripheral attached.

Connecting peripherals to busses can be accomplished using the normal SmartDesign connectivity options:

- **Auto-Connect -** the system creates a bus structure based on the peripherals that you have instantiated and finds compatible bus interfaces and connects them together
- The Modify Memory Map dialog box
- Canvas - Make connections between your blocks.

Your application and design requirements dictate which address location (or slots) is most suitable for your bus peripherals. For example, the memory controller should be connected to Slot0 of the CoreAHB bus because on Reset, the processor will begin code execution from the bottom of the memory map.

An example of the datasheet is shown in the figure below.



Data Sheet: SDTOP

Copyright 2008 Actel Corporation

Project Settings

FAM: Fusion
Die: AFS600
Package: 256 FBGA
HDL: Verilog
Location: C:/Documents and Settings/farleyc/Desktop/farleyc_Actelprj_84/My First SmartDesign/component/work/SDTOP
State: NOTREADY

Table of Contents

IO's
Cores
Memory Map

IO's

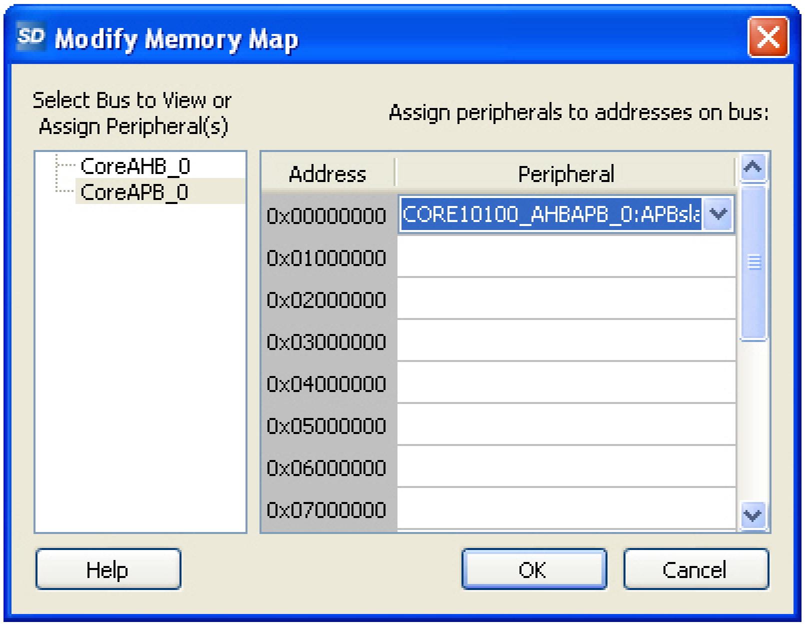
Port Name	Direction	Pin	I/O Standard
VAREF	INOUT	-	LVTTL
c1_V	IN	-	LVTTL
c1	IN	-	LVTTL
v1	IN	-	LVTTL

Figure 10 · Example Memory Map

Modify Memory Map Dialog Box

The Modify Memory Map dialog box (shown in the figure below) enables you to connect peripherals to buses via a drop-down menu. To open the dialog box, right-click the bus instance and choose **Modify Memory Map**.

This dialog simplifies connecting peripherals to specific base addresses on the bus. The dialog shows all the busses in the design; select a bus in the left pane to assign or view the peripherals on a bus. Busses that are bridged to other busses are shown beneath the bus in the hierarchy.


Modify Memory Map
Select Bus to View or Assign Peripheral(s)
CoreAHB_0
CoreAPB_0
Assign peripherals to addresses on bus:
Help
OK
Cancel

Address	Peripheral
0x00000000	CORE10100_AHBAPB_0:APBsla
0x01000000	
0x02000000	
0x03000000	
0x04000000	
0x05000000	
0x06000000	
0x07000000	

Figure 11 · Modify Memory Map Dialog Box

Click the Peripheral drop-down menu to select the peripheral you wish to assign to each address. To remove (unassign) a peripheral from an address, click the drop-down and select the empty element.

Click OK to create the connections between the busses and peripherals in the design.

Canvas View

Canvas Overview

The SmartDesign Canvas is like a whiteboard where functional blocks from various sources can be assembled and connected; interconnections between the blocks represent nets and busses in your design.

You can use the Canvas to manage connections, set attributes, add or remove components, etc. The Canvas displays all the pins for each instance (as shown in the figure below).

The Canvas enables you to drag a component from the Design Hierarchy or a core from the Catalog and add an instance of that component or core in the design. Some blocks (such as Basic Blocks) must be configured and generated before they are added to your Canvas. When you add/generate a new component it is automatically added to your Design Hierarchy.

To connect two pins on the Canvas, click the **Connection Mode button** to enable it and click and drag between the two pins you want to connect. The Connection Mode button is disabled if you attempt to illegally connect two pins.

Click the **Maximize Work Area button** to hide the other windows and show more of the Canvas. Click the button again to return the work area to the original size.

The Canvas displays bus pins with a + sign (click to expand the list) or - (click to hide list). If you add a slice on a bus the Canvas adds a + to the bus pin.

Components can be reconfigured any time by double-clicking the instance on the Canvas. You can also add bus interfaces to instances using this view. In the Canvas view, you can add graphic objects and text to your design.

Inputs and bi-directional pins are shown on the left of components, and output pins are shown on the right.

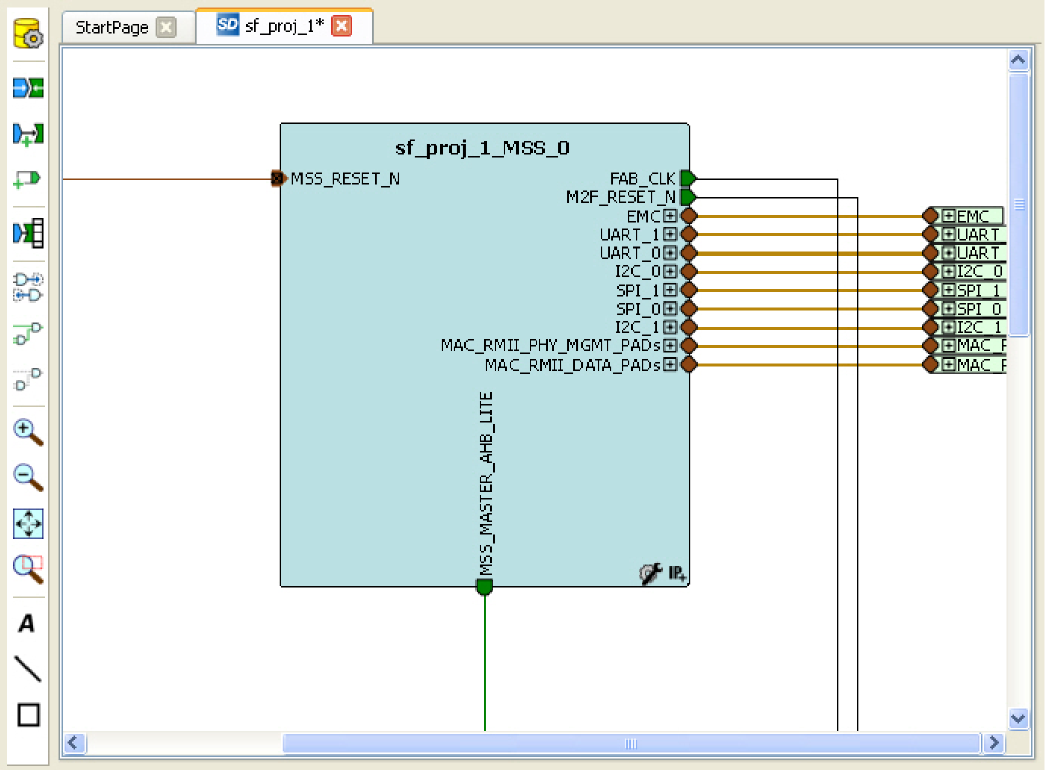


Figure 12 · SmartDesign Canvas

See Also

Canvas Icons

Displaying Connections on the Canvas

The Canvas shows the instances and pins in your design (as shown in the figure below). Right-click the Canvas and choose Show Nets to display nets.

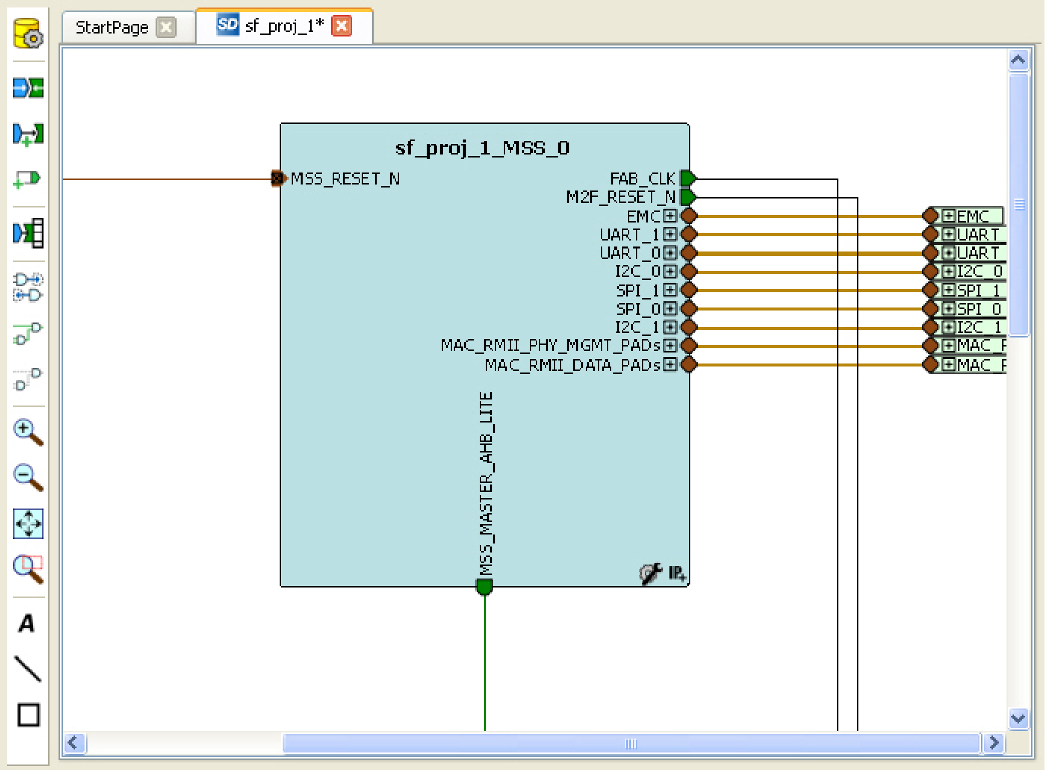


Figure 13 · Components in SmartDesign

Pin and Attribute Icons

Unconnected pins that do not require a connection are gray.

Unconnected pins that require a connection are red.

Unconnected pins that have a default tie-off are pale green.

Connected pins are green.

Right-click a pin to assign an attribute.

Pins assigned attributes are shown with an icon, as shown in the table below.

Table 2 · Pin Attribute Icons

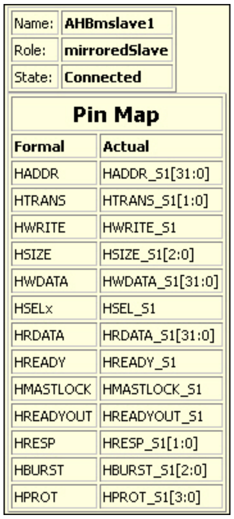
Attribute	Icon
Tie Low	⏚
Tie High	↑

Attribute	Icon
Invert	▷○
Mark as Unused	✕
Tie to Constant	k ▶

See the Canvas Icons reference page for definitions for each element on the Canvas.

Each connection made using a bus interface is shown in a separate connection known as a bus-interface net.

Move the mouse over a bus interface to display its details (as shown below).

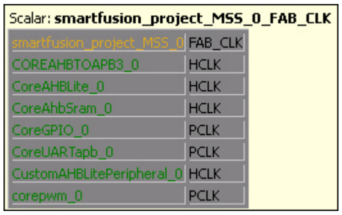


Name:	**AHBmslave1**
Role:	**mirroredSlave**
State:	**Connected**

Pin Map

Formal	**Actual**
HADDR	HADDR_S1[31:0]
HTRANS	HTRANS_S1[1:0]
HWRITE	HWRITE_S1
HSIZE	HSIZE_S1[2:0]
HWDATA	HWDATA_S1[31:0]
HSELx	HSEL_S1
HRDATA	HRDATA_S1[31:0]
HREADY	HREADY_S1
HMASTLOCK	HMASTLOCK_S1
HREADYOUT	HREADYOUT_S1
HRESP	HRESP_S1[1:0]
HBURST	HBURST_S1[2:0]
HPROT	HPROT_S1[3:0]

Hover over a bus interface net to see details (as shown below).



Scalar: **smartfusion_project_MSS_0_FAB_CLK**

smartfusion_project_MSS_0	FAB_CLK
COREAHBTOAPB3_0	HCLK
CoreAHBLite_0	HCLK
CoreAhbSram_0	HCLK
CoreGPIO_0	PCLK
CoreUARTapb_0	PCLK
CustomAHBLitePeripheral_0	HCLK
corepwm_0	PCLK

Making Connections Using the Canvas

Use the Canvas or Connectivity dialog box to make connections between instances.

You can use Connection Mode on the Canvas to quickly connect pins. Click the **Connection Mode** button to start, then click and drag between any two pins to connect them. Illegal connections are disabled. Click the Connection Mode button again to exit Connection Mode.

To connect two pins on the Canvas, select any two (Ctrl + click to select a pin), right-click one of the pins you selected and choose **Connect**. Illegal connections are disabled; the Connect menu option is unavailable.

Promoting Ports to Top Level

To automatically promote a port to top level, select the port, right-click, and choose **Promote To Top Level**. This automatically creates top-level ports of that name and connects the selected ports to them. If a port name already exists, a choice is given to either connect to the existing ports or to create a new port with a name <port name>_<i> where i = 1…n.

Double-click a top-level port to rename it.

Bus slices cannot be automatically promoted to top level. You must create a top level port of the bus slice width and then manually connect the bus slice to the newly created top level port.

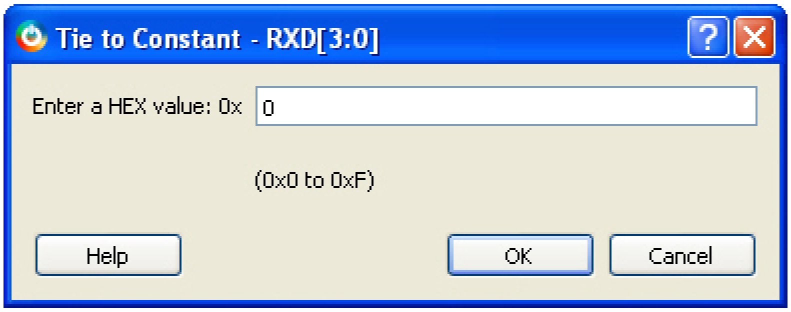
Tying Off Input Pins

To tie off ports, select the port, right-click and choose **Tie High** or **Tie Low**.

Tying to Constant

To tie off bus ports to a constant value, select the port, right-click and choose **Tie to a Constant**. A dialog appears (as shown in the figure below) and enables you to specify a hex value for the bus.

To remove the constant, right-click the pin and choose **Clear Attribute** or **Disconnect**.


Tie to Constant - RXD[3:0]

Enter a HEX value: 0x 0

(0x0 to 0xF)

Figure 14 · Tie to Constant Dialog Box

Making Driver and Bus Interface Pins Unused

Driver or bus interface pins can be marked unused (floating/dangling) if you do not intend to use them as a driver in the design. If you mark a pin as unused the Design Rules Check does not return Floating Driver or Unconnected Bus Interface messages on the pin.

Once a pin is explicitly marked as unused it cannot be used to drive any inputs. The unused attribute must be explicitly removed from the pin in order to connect it later. To mark a driver or bus interface pin as unused, right-click the driver or bus interface pin and choose **Mark as Unused**.

See Also

Show/Hide Bus Interface Pins

Simplifying the Display of Pins on an Instance using Pin Groups

The Canvas enables you to group and ungroup pins on a single instance to simplify the display. This feature is useful when you have many pins in an instance, or if you want to group pins at the top level. Pin groups are cosmetic and affect only the Canvas view; other SmartDesign views and the underlying design are not affected by the pin groups.

Grouping pins enables you to:

- Hide pins that you have already connected
- Hide pins that you intend to work on later
- Group pins with similar functionality
- Group unused pins
- Promote several pins to Top Level at once

To group pins:

1. Ctrl + click to select the pins you wish to group. If you try to click-and-drag inside the instance you will move the instance on the Canvas instead of selecting pins.
2. Right-click and choose **Add pins to group** to create a group. Click + to expand a group. The icon associated with the group indicates if the pins are connected, partially connected, or unconnected (as shown in the figure below).

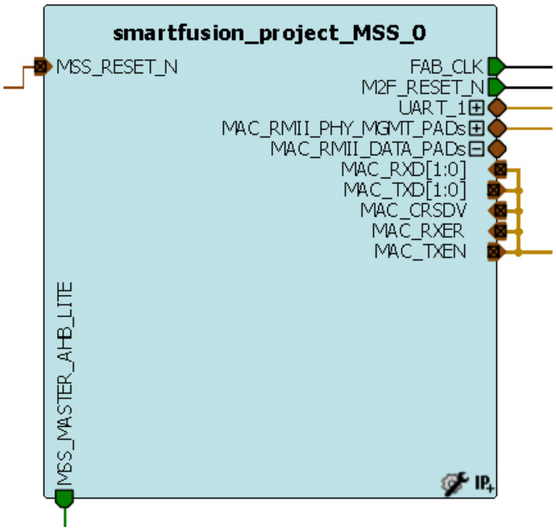


Figure 15 · Groups in an Instance on the Canvas

To add a pin to a group, Ctrl + click to select both the pin and the group, right-click and choose **Add pin to group**.

To name a group:

To name a group, right-click the port name and choose **Rename Group**.

To ungroup pins:

1. Click + to expand the group.
2. Right-click the pin you wish to remove from the group and choose **Ungroup selected pins**. Ctrl + click to select and remove more than one pin in a group.

A group remains in your instance after you remove all the pins. It has no effect on the instance; you can leave it if you wish to add pins to the group later, or you can right-click the group and choose **Delete Group** to remove it from your instance.

If you delete a group from your instance any pins still in the group are unaffected.

To promote a group to Top level:

1. Create a group of pins.
2. Right-click the group and choose **Promote to Top Level**.

Bus Instances

Bus Components in the Core Catalog, such as CoreAHB or CoreAPB, implement an on-chip bus fabric. When these components are instantiated into your canvas they are displayed as horizontal or vertical lines. Double-click the bus interfaces of your component to edit the connections.

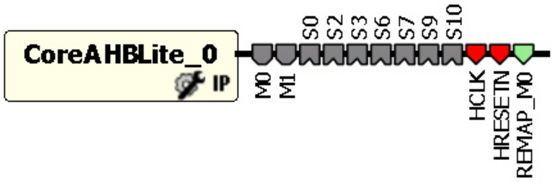


Figure 16 · Bus Instance in SmartDesign

Adding Graphic Objects

You can document your design by adding comments and notations directly on the Canvas.

The Canvas toolbar enables you to add and modify decorative graphic objects, such as shapes, labels and lines on the Canvas.

Adding and Deleting Lines and Shapes

To add a line or a shape:

1. Select the line or shape button.
2. Click, drag and release on the Canvas. The table below provides a description of each button.

Button	Description
╲	Line
□	Rectangle

Note: Hold the Shift key to constrain line and arrow to 45 degree increments or constrain the proportions of the rectangle (square).

To change the line and fill properties:

1. Select the element(s), right-click it, and choose **Properties**.
   - Select **Line** to modify the color, style and width of the line.
   - Select **Fill** to modify the crosshatch and the foreground and background colors.
2. Click **OK**.

To delete a line or shape, select the object and press Delete.

Adding Text

To add text, select the text tool and click the Canvas to create a text box. To modify the text, double-click the text box and then type.

To modify the text box properties:

1. Select the text box, right-click it, and choose **Properties**.
   - Select **Text** to modify the text alignment.
   - Select **Line** to modify the color, style and width of the line.
   - Select **Fill** to modify the crosshatch and the foreground and background colors.
   - Select **Font** to modify the font properties.
2. Click **OK**.

Editing Properties for Graphic Objects on the Canvas

Right-click any graphic object to update properties, such as Fill, and Line properties for shapes and lines, or Font options for text properties.

Auto-Arranging Instances

Right-click the Canvas and choose **Auto Arrange Instances** from the right-click menu to auto-arrange the instances on the Canvas.

Locking Instance and Top Level Port Positions

You can lock the placement of instances on the Canvas. Right-click the instance or Top-level port and choose **Lock** to lock the placement. When you lock placement you can click and drag to move the instance manually but the Auto Arrange Instances menu option has no effect on the instance.

To unlock an instance, right-click the instance and choose **Unlock**.

Right-click a top level port and choose **Unlock Position** to return it to its default position.

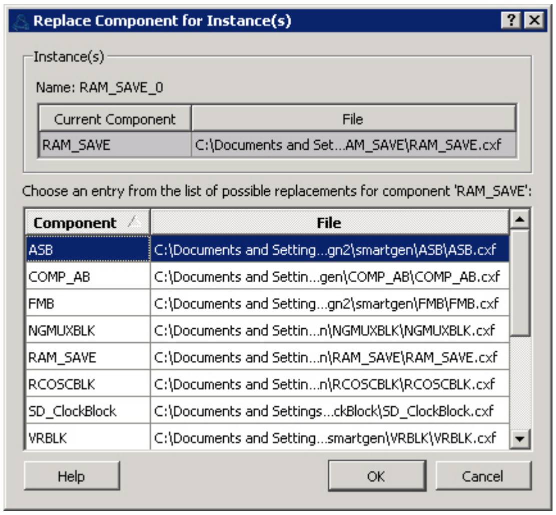
See Also

[Bus Instances]()

[Simplifying the Display of Pins on an Instance using Pin Groups]()

Replace Component for Instance

You can use the Replace Component for Instance dialog box (shown in the figure below) to restore or update version instances on your Canvas without creating a new instance and losing your connections.



Name: RAM_SAVE_0

Current Component	File
RAM_SAVE	C:\Documents and Set...AM_SAVE\RAM_SAVE.cxf

Choose an entry from the list of possible replacements for component 'RAM_SAVE':

Component	File
ASB	C:\Documents and Setting...gn2\smartgen\ASB\ASB.cxf
COMP_AB	C:\Documents and Settin...gen\COMP_AB\COMP_AB.cxf
FMB	C:\Documents and Setting...gn2\smartgen\FMB\FMB.cxf
NGMUXBLK	C:\Documents and Settin...n\NGMUXBLK\NGMUXBLK.cxf
RAM_SAVE	C:\Documents and Settin...n\RAM_SAVE\RAM_SAVE.cxf
RCOSCBLK	C:\Documents and Settin...n\RCOSCBLK\RCOSCBLK.cxf
SD_ClockBlock	C:\Documents and Settings...ckBlock\SD_ClockBlock.cxf
VRBLK	C:\Documents and Setting...smartgen\VRBLK\VRBLK.cxf

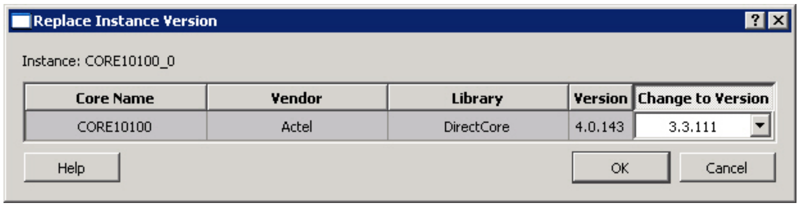
Figure 17 · Replace Component for Instance Dialog Box

To change the version of an instance:

1. From the right-click menu choose **Replace Component for Instance**. The Replace Component for Instance dialog box appears.
2. Select a component and choose a new version from the list. Click **OK**.

Replace Instance Version

The Replace Instance Version dialog box enables you to replace an IP instance with another version. You can restore or replace your IP instance without creating a new instance or losing your connections.



Instance: CORE10100_0

Core Name	Vendor	Library	Version	Change to Version
CORE10100	Actel	DirectCore	4.0.143	3.3.111

Figure 18 · Replace Instance Version Dialog Box

To replace an instance version:

1. Right click any IP instance and choose **Replace Instance Version**. The dialog box appears.
2. Choose the version you wish to use from the **Change to Version** dropdown menu (as shown in the figure above) and click OK to continue.

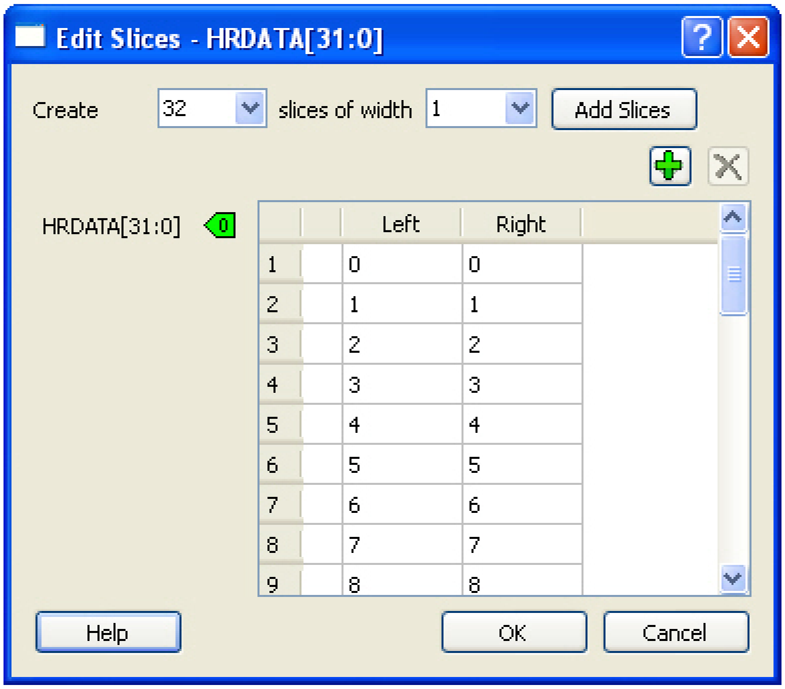
Slicing

Bus ports can be sliced or split using Slicing. Once a slice is created, other bus ports or slices of compatible size can be connected to it.

The Edit Slices dialog box enables you to automatically create bus slices of a specified width.

To create a slice:

1. Select a bus port, right-click, and choose **Edit Slice.** This brings up the **Edit Slices** dialog box (see figure below).



Create 32 slices of width 1

	Left	Right
1	0	0
2	1	1
3	2	2
4	3	3
5	4	4
6	5	5
7	6	6
8	7	7
9	8	8

Figure 19 · Edit Slices Dialog Box

2. Enter the parameters for the slice and click **Add Slices**. You can also create individual slices and specify their bus dimensions manually.
3. Click **OK** to continue.

Note: Note: Overlapping slices cannot be created for IN and INOUT ports on instances or top-level OUT ports.

To remove a slice, select the slice, right-click, and choose **Delete Slice.**

Rename Net

To rename a net:

1. Right-click the net on the Canvas and choose **Rename Net**. This opens the Rename Net dialog box.
2. Type in a new name for the net.

Note: Note: The system automatically assigns net names to nets if they are not explicitly specified. Once you have specified a name for a net, that name will not be over-written by the system.

Automatic Names of Nets

Nets are automatically assigned names by the tool according to the following rules:

In order of priority

1. If user named then name = user name
2. If net is connected to top-level port then name = port name; if connected to multiple ports then pick first port
3. If the net has no driver, then name = net_[i]
4. If the net has a driver, name = instanceName_driverpinName

Slices

For slices, name = instanceName_driverpinName_sliceRange; for example u0_out1_4to6.

GND and VCC Nets

The default name for GND/VCC nets is net_GND and net_VCC.

Expanded Nets for Bus Interface Connections

Expanded nets for bus interface connections are named busInterfaceNetName_<i>_driverPinName.

Organizing Your Design on the Canvas

You may find it easier to create and navigate your SmartDesign if you organize and label the instances and busses on the Canvas.

You can show and hide nets, lock instances, rotate busses, group and ungroup pins, rename instances / groups / pins, and auto-arrange instances.

To organize your design:

1. Right-click the Canvas and choose **Auto Arrange Instances** from to automatically arrange instances. SmartDesign's auto-arrange feature optimizes instance location according to connections and instance size.
2. Right-click any instance and choose Lock Location to fix the placement. Auto-Arrange will not move any instances that are locked.
3. Click Auto-Arrange again to further organize any unlocked instances. Continue arranging and locking your instances until you are satisfied with the layout on the Canvas.

If your design becomes cluttered, group your pins. It may help to group pins that are functionally similar, or to group pins that are already connected or will be unused in your design.

To further customize your design's appearance:

Double-click the names of instances to add custom names. For example, it may be useful to rename an instance based on a value you have set in the instance: the purpose of an instance named 'array_adder_bus_width_5' is easier to remember than 'array_adder_0'.

Creating a SmartDesign

Adding Components and Modules (Instantiating)

SmartDesign components, Design Block cores, IP cores, and HDL modules are displayed in the Design Hierarchy and Files tabs.

To add a component, do either of the following:

- Select the component in the Design Hierarchy tab or Catalog and drag it to the Canvas.
- Right-click a component in the Design Hierarchy tab or Catalog and choose **Instantiate in <SmartDesign name>.**

The component is instantiated in the design.

SmartDesign creates a default instance name. To rename the instance, double-click the instance name in the Canvas.

Adding a SmartDesign Component

SmartDesign components can be instantiated into another SmartDesign component.

Once a SmartDesign is generated, the exported netlist can be instantiated into HDL like any other HDL module.

Note: Note: HDL modules with syntax errors cannot be instantiated in SmartDesign. However, since SmartDesign requires only the port definitions, the logic causing syntax errors can be temporarily commented out to allow instantiation of the component.

Adding or Modifying Top Level Ports

You can add ports to, and/or rename ports in your SmartDesign.

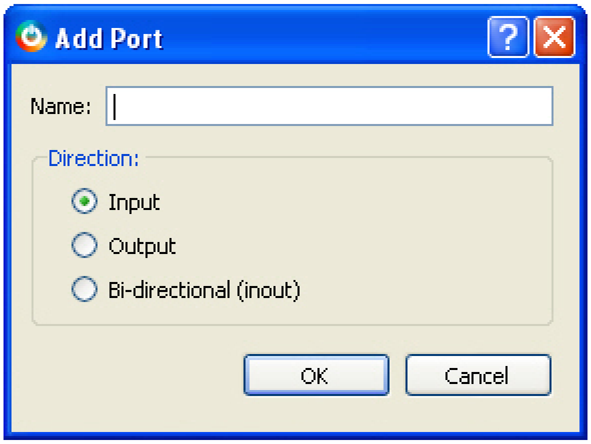
Add Prefixes to Bus Interface / Group Names on Top-level Ports:

Bus Interfaces and Groups are composed of other ports. On the top level, you can add prefixes to the group or bus interface port name to the sub-port names. To do so, right-click the group or bus interface port and choose **Prefix <name> to Port Names**.

Adding/Renaming Ports

To add ports:

1. From the **SmartDesign** menu, choose **Add port**. The Add Port dialog box appears (as shown below).



Direction:

● Input

○ Output

○ Bi-directional (inout)

Figure 20 · Add New Port Dialog Box

2. Specify the name of the port you wish to add. You can specify a bus port by indicating the bus width directly into the name using brackets [ ], such as mybus[3:0].
3. Select the direction of the port.

To remove a port from the top level, right-click the port and choose **Delete Top Level Port**.

Modify Port

To rename a top-level port, right-click the top-level port and choose **Modify Top Level Port**. You can rename the port, change the bus width (if the port is a bus), and change the port direction.

Right-click a top-level port and choose **Modify Port** to change the name and/or direction (if available).

See Also

Top Level Connections

Connecting Instances

Automatic Connections

Using automatic connections (as shown in the figure below) enables the software to connect your design efficiently, reducing time required for manual connections and the possibility of introducing errors.

Auto Connect also constructs your bus structure if you have a processor with peripherals instantiated. Based on the type of processor and peripherals, the proper busses and bridges are added to your design.

To auto connect the bus interfaces in your design, right-click the design Canvas and select **Auto Connect**, or from the **SmartDesign** menu, choose **Auto Connect**.

SmartDesign searches your design and connects all compatible bus interfaces.

SmartDesign will also form known connections for any SoC systems such as the processor CLK and RESET signals.

If there are multiple potential interfaces for a particular bus interface, Auto Connect will not attempt to make a connection; you must connect manually. You can use the Canvas to make the manual connections.

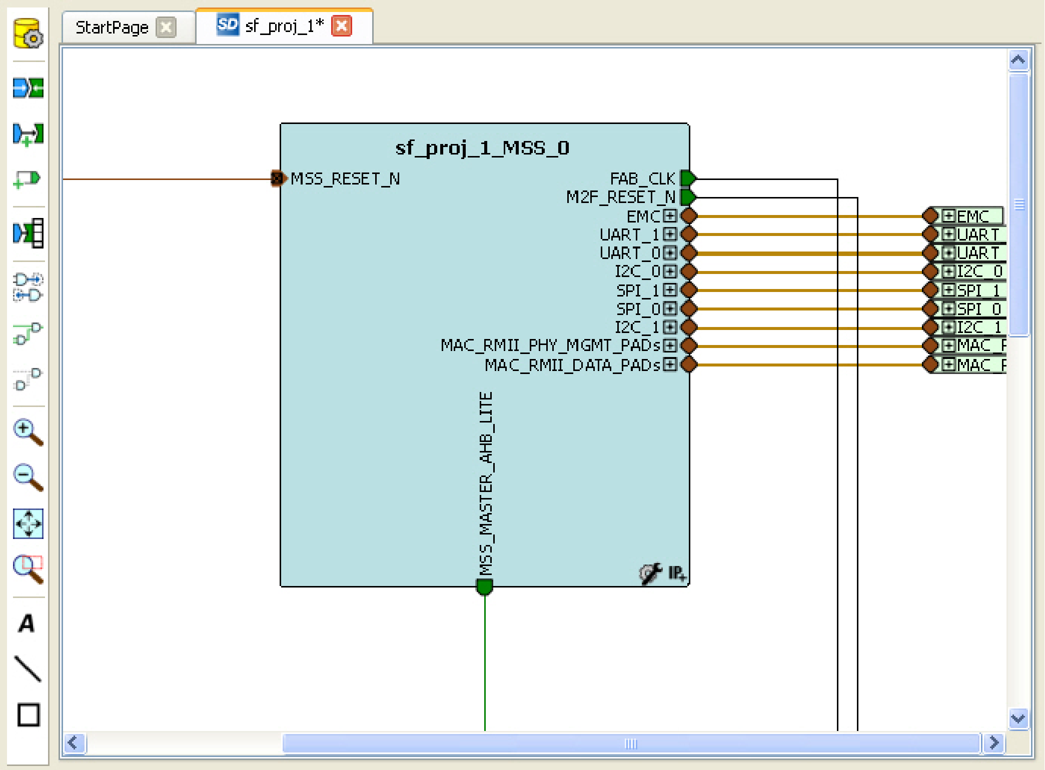


Figure 21 · Auto-Connected Cores

QuickConnect

The QuickConnect dialog box enables you to make connections in your design without using the Canvas. It is useful if you have a large design and know the names of the pins you wish to connect. Connections are reflected in the Canvas as you make them in the dialog box; error messages appear in the Log window immediately. It may be useful to resize the QuickConnect dialog box so that you can view the Log window or Canvas while you make connections.

To connect pins using QuickConnect:

1. Find the **Instance Pin** you want to connect and click to select it.
2. In **Pins to Connect**, find the pin you wish to connect, right-click and choose **Connect**. If necessary, use the **Search** field to narrow down the list of pins displayed in Pins to Connect.

   Note that if the connection is invalid then Connect is grayed out.

If you wish to invert or tie a pin high, low or Mark Unused:

1. Find the **Instance Pin** you want to invert or tie high/low
2. Right-click **Connection** and choose **Invert**, **Tie High**, **Tie Low** or **Mark Unused**.

If you wish to promote a pin to the top level of your design:

1. Find the Instance Pin you want to promote.
2. Right-click the pin and choose **Promote to Top**.

You can perform all connectivity actions that are available in the Canvas, including: slicing bus pins, tying bus pins to a constant value, exposing pins from a bus interface pins and disconnecting pins. All actions are accessible from the right-click context menu on the pin.

Instance Pins lists all the available instance pins in your design and their connection (if any). Use the drop-down list to view only unconnected pins, or to view the pins and connections for specific elements in your design.

Pins to Connect lists the instances and pins in your design. Use the Search field to find a specific instance or pin. The default wildcard search is '*.*'. Wildcard searches for CLK pins (*.*C*L*K) and RESET pins (*.*R*S*T) are also included.

Here are some of the sample searches that you may find useful:

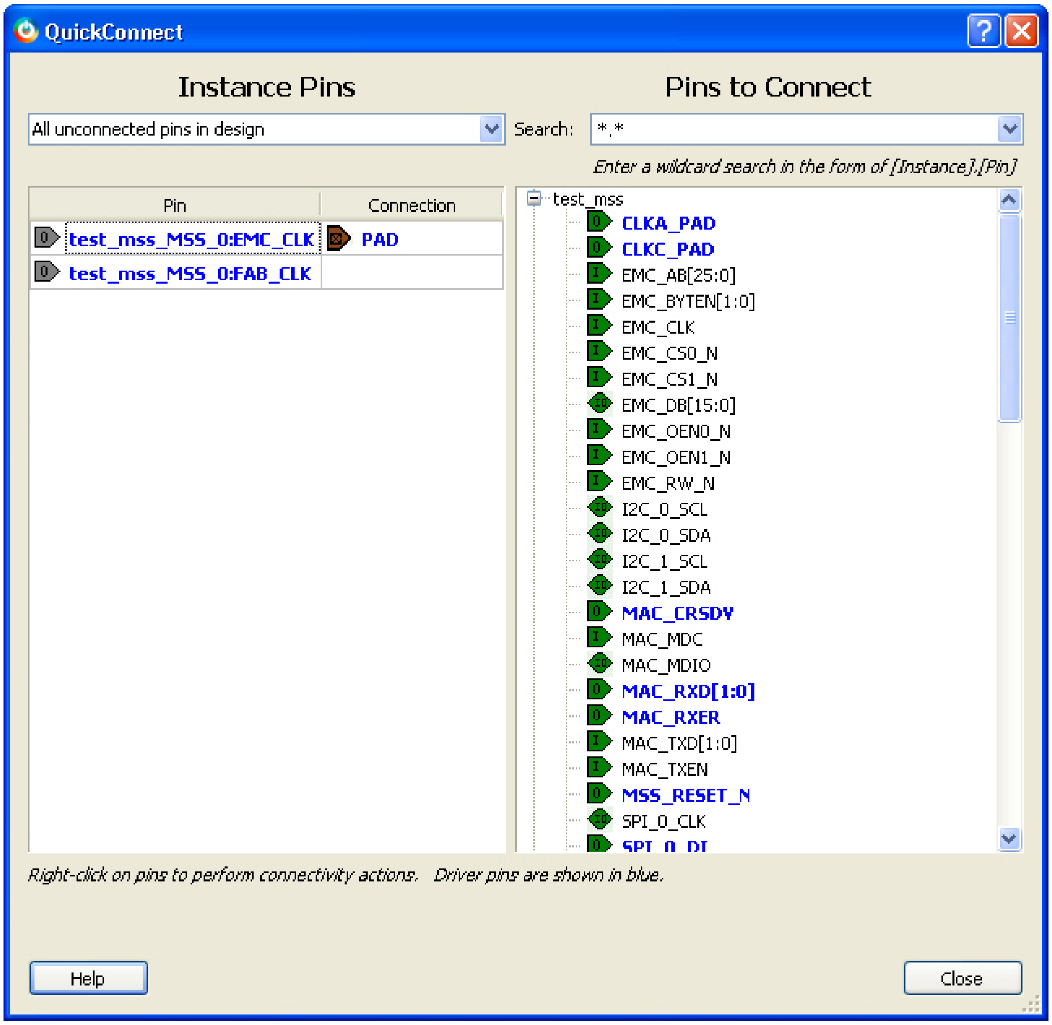
- *UART*.*: show all pins of any instances that contain UART in the name
- MyUART_0.*: only show the pins of the “MyUART_0” instance
- *.p: show all pins in the design that contain the letter ‘p’

Double-click an instance in Pins to Connect to expand or collapse it.

The pin letters and icons in the QuickConnect dialog box are the same as the Canvas icons and communicate information about the pin. Inputs, Outputs and I/Os are indicted by I, O, and I/O, respectively.

Additional information is communicated by the color:

- Red - Mandatory connection, unconnected
- Green - Connected
- Grey - Optional, unconnected pin
- Brown - Pad
- Light Green - Connected to a default connection on generation
- Blue - Driver pin



All unconnected pins in design

Search: *.*

Enter a wildcard search in the form of [Instance].[Pin]

Pin	Connection
O test_mss_MSS_0:EMC_CLK	PAD
O test_mss_MSS_0:FAB_CLK	

test_mss
- O CLKA_PAD
- O CLKC_PAD
- I EMC_AB[25:0]
- I EMC_BYTEN[1:0]
- I EMC_CLK
- I EMC_CS0_N
- I EMC_CS1_N
- IO EMC_DB[15:0]
- I EMC_OEN0_N
- I EMC_OEN1_N
- I EMC_RW_N
- IO I2C_0_SCL
- IO I2C_0_SDA
- IO I2C_1_SCL
- IO I2C_1_SDA
- O MAC_CRSDV
- I MAC_MDC
- IO MAC_MDIO
- O MAC_RXD[1:0]
- O MAC_RXER
- I MAC_TXD[1:0]
- I MAC_TXEN
- O MSS_RESET_N
- IO SPI_0_CLK
- O SPI_0_DI

Right-click on pins to perform connectivity actions. Driver pins are shown in blue.

Figure 22 ·
QuickConnect Dialog Box

Manual Connections

You can use Connection Mode to click and drag and connect pins. Click the Connection Mode button to toggle it, and click and drag between any two pins to connect them. Illegal connections will not be allowed.

To make manual connections between to pins on the Canvas, select both pins (use CTRL + click), right-click either pin and choose **Connect**. If the pins cannot be legally connected the connection will fail.

Deleting Connections

To delete a net connection on the Canvas, click to select the net and press the Delete key, or right-click and choose **Delete**.

To remove all connections from one or more instances on the Canvas, select the instances on the Canvas, right-click and choose **Clear all Connections**. This disconnects all connections that can be disconnected legally.

Certain connections to ports with PAD properties cannot be disconnected. PAD ports must be connected to a design’s top level port. PAD ports will eventually be assigned to a package pin. In SmartDesign, these ports are automatically promoted to the top level and cannot be modified or disconnected.

Top-Level Connections

Connections between instances of your design normally require an OUTPUT (Driver Pin) on one instance to one or more INPUT(s) on other instances. This is the basic connection rule that is applied when connecting.

However, directions of ports at the top level are specified from an external viewpoint of that module. For example, an INPUT on the top level is actually sending (‘driving’) signals to instances of components in your design. An OUTPUT on the top level is receiving (‘sinking’) data from a Driver Pin on an internal component instance in your SmartDesign design.

The implied direction is essentially reversed at the top-level. Making connections from an OUTPUT of a component instance to an OUTPUT of top-level is legal.

This same concept applies for bus interfaces; with normal instance to instance connections, a MASTER drives a SLAVE interface. However, they go through a similar reversal on the top-level.

Bus Interfaces

About Bus Interfaces

A bus interface is a standard mechanism for specifying the interconnect rules between components or instances in a design. A bus definition consists of the roles, signals, and rules that define that bus type. A bus interface is the instantiation of that bus definition onto a component or instance.

The available roles of a bus definition are master, slave, and system.

A master is the bus interface that initiates a transaction (such as read or write) on a bus.

A slave is the bus interface that terminates/consumes a transaction initiated by a master interface.

A system is the bus interface that does not have a simple input/output relationship on both master/slave. This could include signals that only drive the master interface, or only drive the slave interface, or drive both the master and slave interfaces. A bus definition can have zero or more system roles. Each system role is further defined by a group name. For example, you may have a system role for your arbitration logic, and another for your clock and reset signals.

Mirror roles are for bus interfaces that are on a bus core, such as CoreAHB or CoreAPB. They are equivalent in signal definition to their respective non-mirror version except that the signal directions are reversed.

The diagram below is a conceptual view of a bus definition.

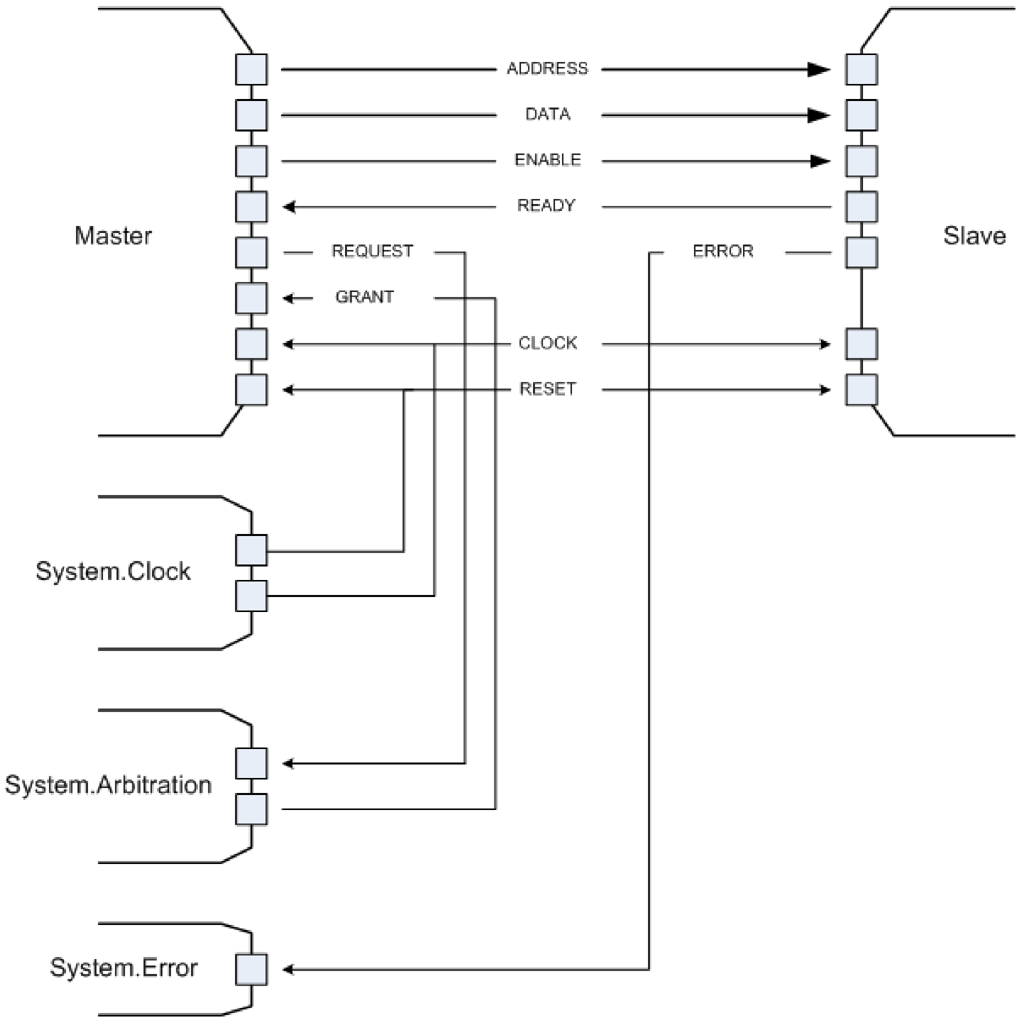


Figure 23 · Bus Definition

See Also:

Using bus interfaces in SmartDesign

Using Bus Interfaces in SmartDesign

Adding bus interfaces to your design enables SmartDesign to do the following:

- Auto connect compatible interfaces
- Enforce DRC rules between instances in your design
- Search for compatible components in the project

The Catalog in the Project Manager contains a list of Microsemi SoC-specific and industry standard bus definitions, such as AMBA.

You can add bus interfaces to your design by dragging the bus definitions from the Bus Interfaces tab in the Catalog onto your instances inside SmartDesign.

SmartDesign supports automatic creation of data driven configurators based on HDL generics/parameters.

If your block has all the necessary signals to interface with the AMBA bus protocol (ex: address, data, control signals):

1. Right-click your custom HDL block and choose **Create Core from HDL**. The Libero SoC creates your core and asks if you want to add bus interfaces.
2. Click **Yes** to open the Edit Core Definition dialog box and add bus interfaces. Add the bus interfaces as necessary.
3. Click **OK** to continue.

Now your instance has a proper AMBA bus interface on it. You can manually connect it to the bus or let Auto Connect find a compatible connection.

Some cores have bus interfaces that are instantiated during generation.

Certain bus definitions cannot be instantiated by a user. Typically these are the bus definitions that define a hardwired connection and are specifically tied to a core/macro. They are still available in the catalog for you to view their properties, but you will not be able to add them onto your own instances or components. These bus definitions are grayed out in the Catalog.

A hardwired connection is a required silicon interconnect that must be present and specifically tied to a core/macro. For example, when using the Real Time Counter in a Fusion design you must also connect it to a Crystal Oscillator core.

Maximum masters allowed - Indicates how many masters are allowed on the bus.

Maximum slaves allowed - Indicates how many slaves are allowed on the bus.

Default value - indicates the value that the input signal will be tied to if unused. See Default tie-offs with bus interfaces.

Required connection - Indicates if this bus interface must be connected for a legal design.

Adding or Modifying Bus Interfaces in SmartDesign

SmartDesign supports automatic creation of data driven configurators based on HDL generics/parameters. You can add a bus interface from your HDL module or you can add it from the Catalog.

To add a bus interface using your custom HDL block:

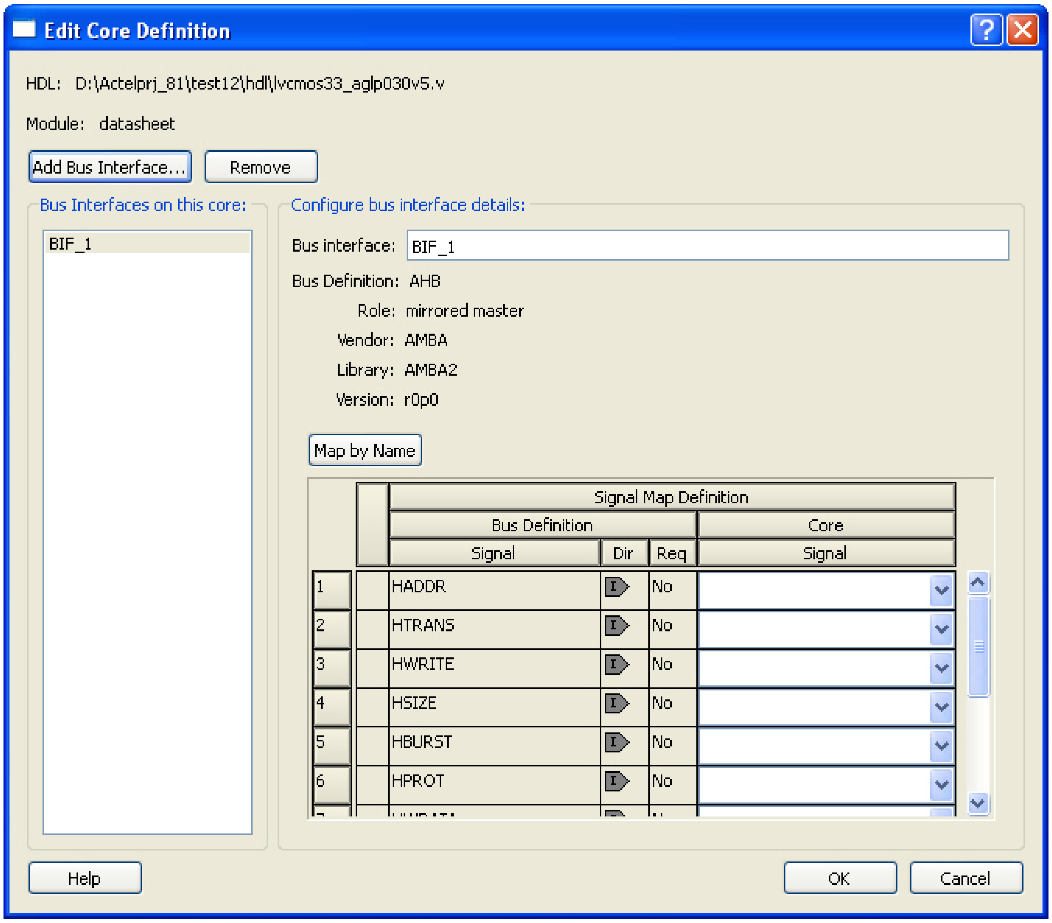
If your block has all the necessary signals to interface with the AMBA bus protocol (such as address, data, and control signals):

1. Right-click your custom HDL block and choose **Create Core from HDL**. The Libero SoC creates your core and asks if you want to add bus interfaces.
2. Click **Yes** to open the Edit Core Definition dialog box and add bus interfaces. Add the bus interfaces as necessary.
3. Click **OK** to continue.

Now your instance has a proper AMBA bus interface on it. You can manually connect it to the bus or let Auto Connect find a compatible connection.

To add (or modify) a bus interface to your Component:

1. Right-click your Component and choose **Edit Core Definition**. The Edit Core Definition dialog box opens, as shown in the figure below.



HDL: D:\Actelprj_81\test12\hdl\lvcmos33_aglp030v5.v

Module: datasheet

BIF_1

Bus interface: BIF_1

Bus Definition: AHB

Role: mirrored master

Vendor: AMBA

Library: AMBA2

Version: r0p0

		Signal Map Definition			
		Bus Definition			Core
		Signal	Dir	Req	Signal
1		HADDR	I	No	
2		HTRANS	I	No	
3		HWRITE	I	No	
4		HSIZE	I	No	
5		HBURST	I	No	
6		HPROT	I	No	

Figure 24 · Edit Core Definition Dialog Box

2. Click **Add Bus Interface**. Select the bus interface you wish to add and click **OK**.
3. If necessary, edit the bus interface details.
4. Click **Map by Name** to map the signals automatically. Map By Name attempts to map any similar signal names between the bus definition and pin names on the instance.
5. Click **OK** to continue.

Bus Interface Details

Bus Interface: Name of bus interface. Edit as necessary.

Bus Definition: Specifies the name of the bus interface.

Role: Identifies the bus role (master or slave).

Vendor: Identifies the vendor for the bus interface.

Version: Identifies the version for the bus interface.

Configuration Parameters

Certain bus definitions contain user configurable parameters.

Parameter: Specifies the parameter name.

Value: Specifies the value you define for the parameter.

Consistent: Specifies whether a compatible bus interface must have the same value for this bus parameter. If the bus interface has a different value for any parameters that are marked with consistent set to **yes**, this bus interface will not be connectable.

Signal Map Definition

The signal map of the bus interface specifies the pins on the instance that correspond to the bus definition signals. The bus definition signals are shown on the left, under the **Bus Interface Definition**. This information includes the name, direction and required properties of the signal.

The pins for your instance are shown in the columns under the Component Instance. The signal element is a drop-down list of the pins that can be mapped for that definition signal. .

If the Req field of the signal definition is Yes, you must map it to a pin on your instance for this bus interface to be considered legal. If it is No, you can leave it unmapped.

Bus Interfaces

When you add a bus interface the Edit Core Definition dialog box provides the following Microsemi SoC-specific bus interfaces:

ExtSeqCtrl

This bus interface defines the set of signals required to interface to the Analog System External Sequence Control. If the Analog System is configured with more than a single procedure, it will export this bus interface. Your own logic would need to connect to this bus interface to properly communicate and control the sequencer.

RTCXTL

This bus interface represents the hardwired connection needed between the Real Time Counter and the Crystal Oscillator.

RTCVR

This bus interface represents the hardwired connection needed between the Real Time Counter and the Voltage Regulator Power Supply Monitor.

InitCfg

This is the initialization and configuration interface that is generated as part of the Flash Memory Builder. Any clients can be initialized from the Flash Memory as long as it can connect to this bus interface. This is for pure initialization clients that do not require save-back to the Flash Memory.

InitCfgSave

This is the initialization and configuration interface that is generated as part of the Flash Memory Builder. Any client can be initialized or saved-back to the Flash Memory as long as it can connect to this bus interface. This is for clients that require initialization and save-back capabilities to the Flash Memory.

InitCfgCtrl

This interface is used to initiate the save-back procedure of the Flash Memory.

InitCfgAnalog

This interface is required between the Flash Memory System and the Analog System core.

FlashDirect

This bus interface defines the set of signals that are required to interface directly to the Flash Memory. From the Flash Memory, if you add a data storage client, this interface will be exported. Interfacing to this interface enables direct access to the Flash Memory.

XTLOscClk

This interface represents the Crystal Oscillator clock.

RCOscClk

This interface represents the RC Oscillator clock.

DirectCore Bus Interfaces

When you add a bus interface the Edit Core Definition dialog box provides the following DirectCore bus interfaces.

AHB

The AMBA AHB defines the set of signals for a component to connect to an AMBA AHB or AHBLite bus. The bus interface that is defined in the system is a superset of the signals in the AHB and AHBLite protocol. You can use the AHB bus interface in the bus definition catalog to connect your module to an AHB or AHBLite bus.

APB

The AMBA APB defines the set of signals for a component to connect to an AMBA APB or APB3 bus. The bus interface that is defined in the system is a superset of the signals in the APB and APB3 protocol. You can use the APB bus interface in the bus definition catalog to connect your module to an APB or APB3 bus.

SysInterface

The SysInterface is the interface used between the CoreMP7 and CoreMP7Bridge cores.

DBGInterface

This is the set of debug ports on the CoreMP7 core.

CPInterface

This is the co-processor interface on the CoreMP7 core.

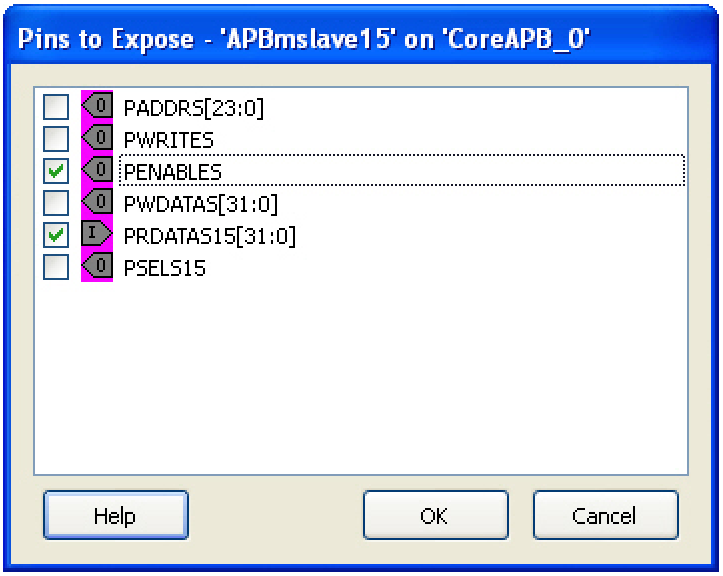
Show/Hide Bus Interface Pins

Pins that are contained as part of a bus interface will automatically be filtered out of the display. These ports are considered to be connected and used as part of a bus interface.

However, there are situations where you may wish to use the ports that are part of the bus interface as an individual port, in this situation you can choose to expose the pin from the bus interface.

To Show/Hide pins in a Bus Interface:

1. Select a bus interface port, right-click, and choose **Show/Hide BIF Pins**. The Show/Hide Pins to Expose dialog box appears (as shown below).



Pins to Expose - 'APBmslave15' on 'CoreAPB_0'

☐ O PADDRS[23:0]
☐ O PWRITES
☑ O PENABLES
☐ O PWDATAS[31:0]
☑ I PRDATAS15[31:0]
☐ O PSELS15

Figure 25 · Expose Driver Pin Dialog Box

2. Click the checkbox associated with the driver pin you want to show. Once the port is shown it appears on the Canvas and is available for individual connection.

Note: Note: If you have already connected the bus interface pin, then you will not be able to expose the non-driver pins. They will be shown grayed out in the dialog. This is to prevent the pin from being driven by two different sources.

To un-expose a driver pin, right-click the exposed port and choose **Show/Hide BIF Pins** and de-select the pin.

Default Tie-offs with Bus Interfaces

Bus definitions can contain default values for each of the defined signals. These default values specify what the signal should be tied to if it is mapped to an unconnected input pin on the instance.

Bus definitions are specified as required connection vs optional connection that defines the behavior of tie-offs during SmartDesign generation.

Required bus interfaces - The signals that are not required to be mapped will be tied off if they are mapped to an unconnected input pin.

Optional bus interfaces - All signals will be tied off if they are mapped to an unconnected input pin.

Tying Off (Disabling) Unused Bus Interfaces

Tying off (disabling) a bus interface sets all the input signals of the bus interface to the default value.

To tie off a bus interface, right-click the bus interface and select Tie Off.

This is useful if your core includes a bus interface you plan to use at a later time. You can tie off the bus interface and it will be disabled in your design until you manually set one of the inputs.

Some bus interfaces are required; you cannot tie off a bus interface that is required. For example, the Crystal Oscillator to RTC (RTCXTL) bus interface is a silicon interface and must be connected.

To enable your pin, right-click the pin and choose **Clear Attribute**.

Required vs. Optional Bus Interfaces

A required bus interface means that it must be connected for the design to be considered legal. These are typically used to designate the silicon interconnects that must be present between certain cores. For example, when using the Real Time Counter in a Fusion design you must also connect it up to a Crystal Oscillator core.

An optional bus interface means that your design is still considered legal if it is left unconnected. However, it may not functionally behave correctly.

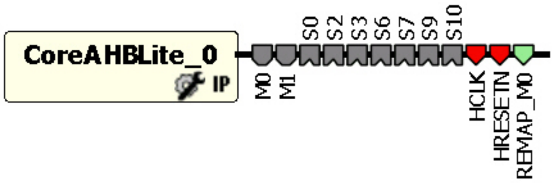


Figure 26 · Required Unconnected, Optional Unconnected, and Connected Bus Interfaces

See Also

Canvas icons

Promoting Bus Interfaces to Top-level

To automatically connect a bus interface to a top-level port, select the bus interface, right-click, and choose **Promote To Top Level**.

This automatically creates a top-level bus interface port of that name and connects the selected port to it. If a bus interface port name already exists, a choice is given to either connect to the existing bus interface port or to create a new bus interface port with a name <port name>_<i> where i = 1...n.

The signals that comprise the bus interface are also promoted.

Promoting a bus interface is a shortcut for creating a top-level port and connecting it to an instance pin.

Incremental Design

Reconfiguring a Component

To reconfigure a component used in a SmartDesign:

- In the Canvas, select the instance and double-click the instance to bring up the appropriate configurator, or the HDL editor; or select the instance, right-click it, and choose **Configure Component**.
- Select the component in the Design Hierarchy tab and from the right-click menu select **Open Component**.

When the configurator is launched from the canvas, you cannot change the name of the component.

See Also

Design state management

Replacing components

Fixing an Out-of-Date Instance

Any changes made to the component will be reflected in the instance with an exclamation mark when you update the definition for the instance. An instance may be out-of-date with respect to its component for the following reasons:

- If the component interface (ports) is different – after reconfiguration - from that of the instance
- If the component has been removed from the project
- If the component has been moved to a different VHDL library
- If the SmartDesign has just been imported

You can fix an out-of-date instance by:

- Replacing the component with a new component (as shown in the figure below)
- Updating with the latest component

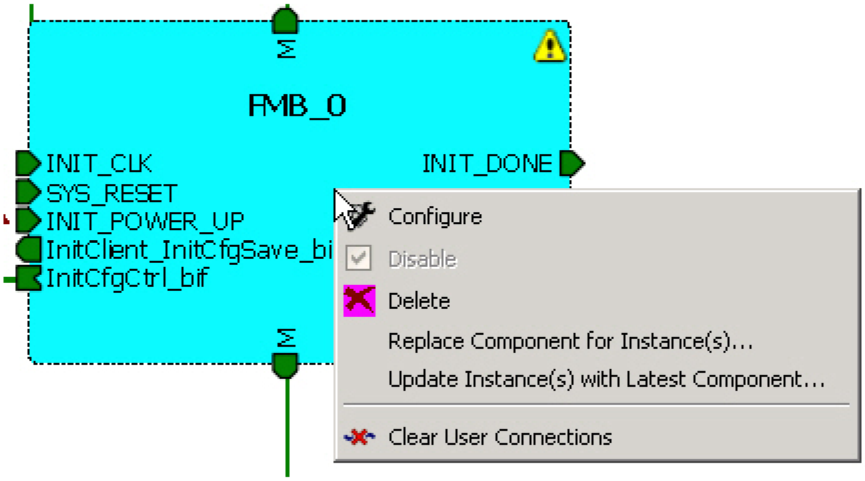

FMB_0
INIT_CLK
SYS_RESET
INIT_POWER_UP
InitClient_InitCfgSave_bi
InitCfgCtrl_bif
INIT_DONE
Configure
Disable
Delete
Replace Component for Instance(s)...
Update Instance(s) with Latest Component...
Clear User Connections

Figure 27 · Right-Click Menu - Replace Component for this Instance

See Also

Design state management

Reconfiguring components

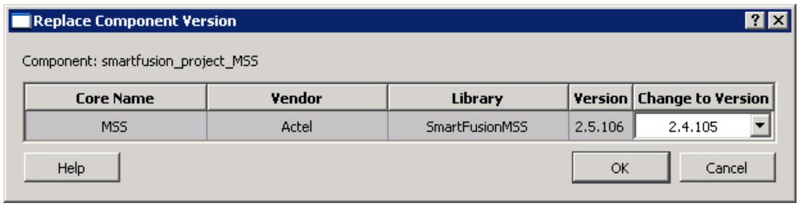
Replacing components

Replacing Component Version

Components of an instance on the Canvas can be replaced with another component and maintain connections to all ports with the same name.

To replace a component in your design:

1. Select the component in the Design Hierarchy, right-click, and choose **Replace Component Version**. The Replace Component for Instance dialog box appears (see figure below).



Replace Component Version

Component: smartfusion_project_MSS

Core Name	Vendor	Library	Version	Change to Version
MSS	Actel	SmartFusionMSS	2.5.106	2.4.105

Figure 28 · Replace Component Version Dialog Box

2. Select the version you want to replace it with and click **OK**.

Design State Management

When any component with instances in a SmartDesign design is changed, all instances of that component detect the change.

If the change only affects the memory content, then your changes do not affect the component's behavior or port interface and your SmartDesign design does not need to be updated.

If the change affects the behavior of the instantiated component, but the change does not affect the component's port interface, then your design must be resynthesized, but the SmartDesign design does not need to be updated.

If the port interface of the instantiated component is changed, then you must reconcile the new definition for all instances of the component and resolve any mismatches. If a port is deleted, SmartDesign will remove that port and clear all the connections to that port when you reconcile all instances. If a new port is added to the component, instances of that component will contain the new port when you reconcile all instances.

The affected instances are identified in your SmartDesign design in the Grid and the Canvas with an exclamation point. Right-click the instance and choose **Update With Latest Component.**

Note: Note: For HDL modules that are instantiated into a SmartDesign design, if the modification causes syntax errors, SmartDesign does not detect the port changes. The changes will be recognized when the syntax errors are resolved.

Changing memory content

For certain cores such as Analog System Builder, Flash Memory, or FlexRAM it is possible to change the configuration such that only the memory content used for programming is altered. In this case Project Manager (SoC) will only invalidate your programming file, but your synthesis, compile, and place-and-route results will remain valid.

When you modify the memory content of a core such as Analog System Builder or RAM with Initialization that is used by a Flash Memory core, the Flash Memory core indicates that one of its dependent components has changed and that it needs to be regenerated. This indication will be shown in the Hierarchy or Files Tab ⚠ .

RAM with Initialization core - You can modify the memory content without invalidating synthesis.

Analog System Builder core - You can modify the following without invalidating synthesis:

- Existing flag settings: threshold levels, assertion/de-assertion counts, OVER/UNDER type
- Modifying sequence order or adding sequence operations

- Changing acquisition times
- Resistor Value for the Current Monitor
- RTC time settings
- Gate Driver source current

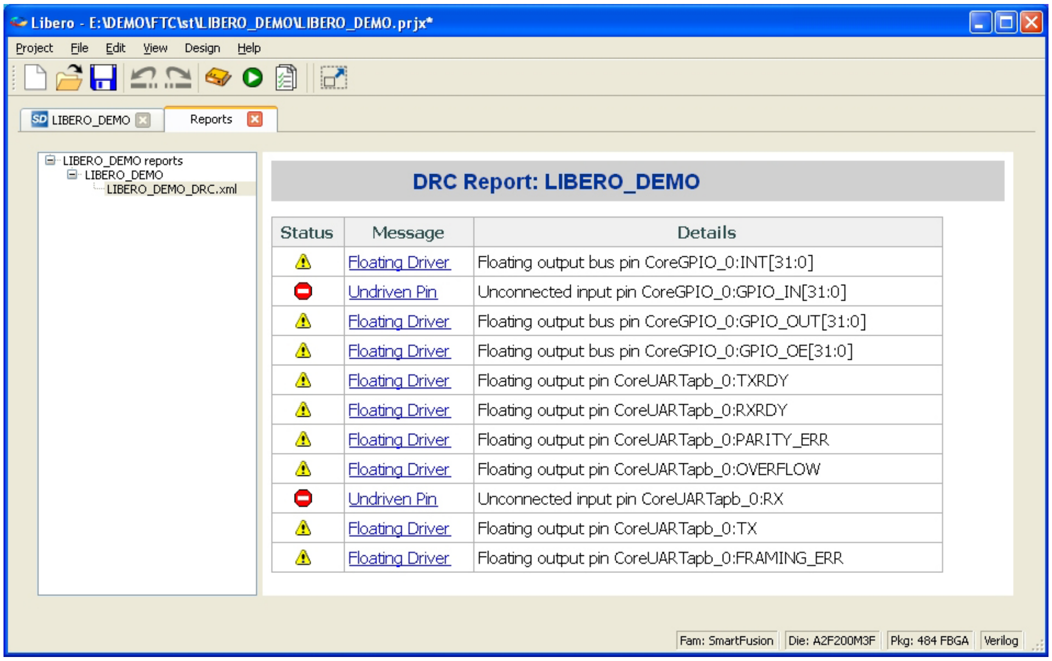
Flash Memory System Builder core - You can modify the following without invalidating synthesis:

- Modifying memory file or memory content for clients
- JTAG protection for Init Clients

Design Rules Check

The Design Rules Check runs automatically when you generate your SmartDesign; the results appear in the Reports tab. To view the results, from the **Design** menu, choose **Reports**.

- **Status** displays an icon to indicate if the message is an error or a warning (as shown in the figure below). Error messages are shown with a small red sign and warning messages with a yellow exclamation point. ⚠
- **Message** identifies the specific error/warning (see list below); click any message to see where it appears on the Canvas
- **Details** provides information related to the Message



LIBERO_DEMO reports
LIBERO_DEMO
LIBERO_DEMO_DRC.xml

DRC Report: LIBERO_DEMO

Status	Message	Details
⚠	Floating Driver	Floating output bus pin CoreGPIO_0:INT[31:0]
⛔	Undriven Pin	Unconnected input pin CoreGPIO_0:GPIO_IN[31:0]
⚠	Floating Driver	Floating output bus pin CoreGPIO_0:GPIO_OUT[31:0]
⚠	Floating Driver	Floating output bus pin CoreGPIO_0:GPIO_OE[31:0]
⚠	Floating Driver	Floating output pin CoreUARTapb_0:TXRDY
⚠	Floating Driver	Floating output pin CoreUARTapb_0:RXRDY
⚠	Floating Driver	Floating output pin CoreUARTapb_0:PARITY_ERR
⚠	Floating Driver	Floating output pin CoreUARTapb_0:OVERFLOW
⛔	Undriven Pin	Unconnected input pin CoreUARTapb_0:RX
⚠	Floating Driver	Floating output pin CoreUARTapb_0:TX
⚠	Floating Driver	Floating output pin CoreUARTapb_0:FRAMING_ERR

Figure 29 · Design Rules Check Results

Message Types:

Unused Instance - You must remove this instance or connect at least one output pin to the rest of the design.

Out-of-date Instance - You must update the instance to reflect a change in the component referenced by this instance; see Fixing an out-of-date instance.

Undriven Pin - To correct the error you must connect the pin to a driver or change the state, i.e. tie low (GND) or tie high (VCC).

Floating Driver - You can mark the pin unused if it is not going to be used in the current design. Pins marked unused are ignored by the Design Rules Check.

Unconnected Bus Interface - You must connect this bus interface to a compatible port because it is required connection.

Required Bus Interface Connection – You must connect this bus interface before you can generate the design. These are typically silicon connection rules.

Exceeded Allowable Instances for Core – Some IP cores can only be instantiated a certain number of times for legal design. For example, there can only be one CortexM1 or CoreMP7 in a design because of silicon rules. You must remove the extra instances.

Incompatible Family Configuration – The instance is not configured to work with this project's Family setting. Either it is not supported by this family or you need to re-instantiate the core.

Incompatible Die Configuration – The instance is not configured to work with this project's Die setting. Either it is not supported or you need to reconfigure the Die configuration.

Incompatible 'Debug' Configuration – You must ensure your CoreMP7 and CoreMP7Bridge have the same 'Debug' configuration. Reconfigure your instances so they are the same.

No RTL License, No Obfuscated License, No Evaluation License – You do not have the proper license to generate this core. Contact Microsemi SoC to obtain the necessary license.

No Top level Ports - There are no ports on the top level. To auto-connect top-level ports, right-click the Canvas and choose Auto-connect

Generating a SmartDesign Component

Before your SmartDesign component can be used by downstream processes, such as synthesis and simulation, you must generate it.

Click the Generate button to generate a SmartDesign component.

This will generate a HDL file in the directory <libero_project>/components/<library>/<yourdesign>.

Note: Note: The generated HDL file will be deleted when your SmartDesign design is modified and saved to ensure synchronization between your SmartDesign component and its generated HDL file.

Generating a SmartDesign component may fail if there are any DRC errors. DRC errors must be corrected before you generate your SmartDesign design.

Generating a Datasheet

If your SmartDesign is the root design in your project, then a Memory Map / Datasheet is produced that contains the information for your design.

Generating Firmware and Software IDE Workspace

If your SmartDesign is the root design in your project, then any compatible firmware drivers for your peripherals are generated to <project>/firmware. Furthermore, if you have specified a Software IDE tool in your profile, then the workspace and projects for that Software IDE are generated into <project>/<SoftwareIDE>.

The datasheet provides all the specifics of the generated firmware drivers and Software IDE workspaces.

Reference

SmartDesign Menu

Command	Icon	Function
Generate Component		Generates the SmartDesign component
Auto Connect		Auto-connects instances
Connection Mode		Toggles connection mode on or off
Add Port		Opens the Add Port dialog box, adds a port to the top SmartDesign component
QuickConnect		Opens the QuickConnect dialog box, enables you to view, find and connect pins
Auto-Arrange Instances		Adds a port to the top of the SmartDesign component
Route All Nets		Re-routes your nets; useful if you are unsatisfied with the default display
Show/Hide Nets		Enables you to show or hide nets on the Canvas
Zoom In	🔍	Zooms in on the Canvas
Zoom Out	🔍	Zooms out on the Canvas
Zoom to Fit		Zooms in or out to include all the elements on the Canvas in the view
Zoom Box	🔍	Zooms in on the selected area
Add Note	*A*	Adds text to your Canvas
Add Line	\	Enables you to add a line to the Canvas
Add Rectangle	□	Enables you to add a rectangle to the Canvas

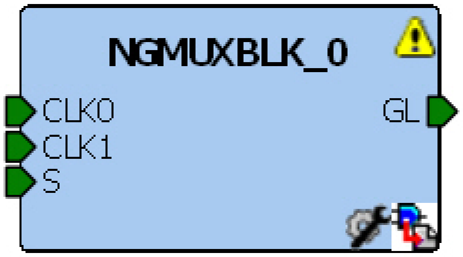
SmartDesign Glossary

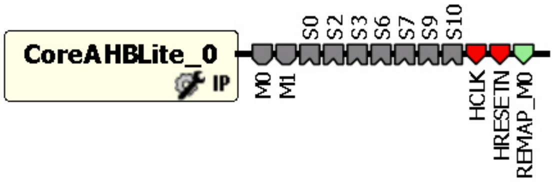
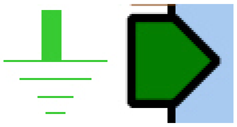
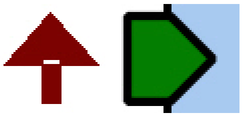
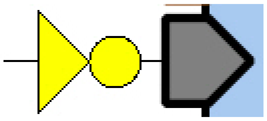
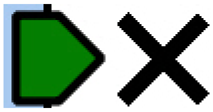
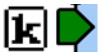
Term	Description
BIF	Abbreviation for bus interface.
bus	An array of scalar ports or pins, where all scalars have a common base name and have unique indexes in the bus.
Bus Definition	Defines the signals that comprise a bus interface. Includes which signals are present on a master, slave, or system interface, signal direction, width, default value, etc. A bus definition is not specific to a logic or design component but is a type or protocol.
Bus Interface	Logical grouping of ports or pins that represent a single functional purpose. May contain both input and output, scalars or busses. A bus interface is a specific mapping of a bus definition onto a component instance.
Bus Interface Net	A connection between 2 or more compatible bus interfaces.
Canvas	Block diagram, connections represent data flow; enables you to connect instances of components in your design.
Component	Design element with a specific functionality that is used as a building block to create a SmartDesign core. A component can be an HDL module, non-IP core generated from the Catalog, SmartDesign core, Designer Block, or IP core. When you add a component to your design, SmartDesign creates a specific instance of that component.
Component Declaration	VHDL construct that refers to a specific component.
Component Port	An individual port on a component definition.
Driver	A driver is the origin of a signal on a net. The input and slave BIF ports of the top-level or the output and Master BIF ports from instances are drivers.
Instance	A specific reference to a component/module that you have added to your design. You may have multiple instances of a single component in your design. For each specific instance, you usually will have custom connections that differ from other instances of the same component.
Master Bus Interface	The bus interface that initiates a transaction (such as a read or write) on a bus.

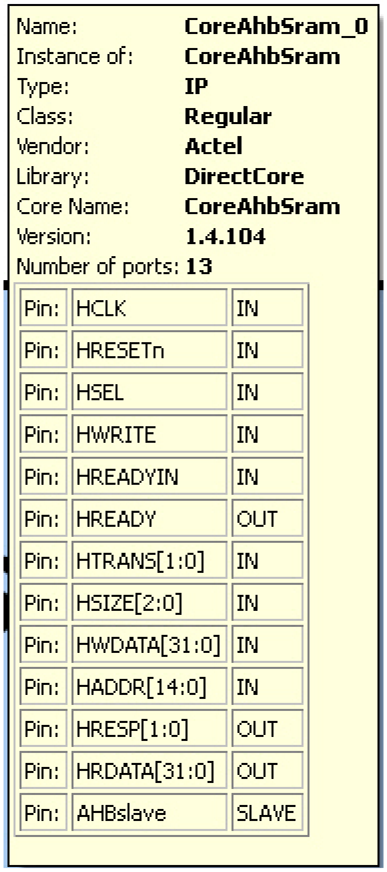
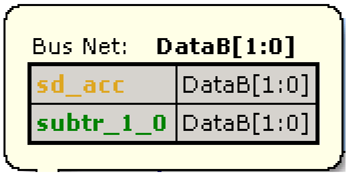
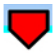
Term	Description
Net	Connection between individual pins. Each net contains a single output pin and one or more input pins, or one or more bi-directional pins. Pins on the net must have the same width.
PAD	The property of a port that must be connected to a design’s top level port. PAD ports will eventually be assigned to a package pin. In SmartDesign, these ports are automatically promoted to the top-level and cannot be modified.
Pin	An individual port on a specific instance of a component.
Port	An individual connection point on a component or instance that allows for an electrical signal to be received or sent. A port has a direction (input, output, bi-directional) and may be referred to as a ‘scalar port’ to indicate that only a single unit-level signal is involved. In contrast, a bus interface on an instance may be considered as a non-scalar, composite port. A component port is defined on a component and an instance port (also known as a ‘pin’) is part of a component instance.
Signal	A net or the electrical message carried on a net.
Slave Bus Interface	Bus interface that terminates a transaction initiated by a master interface.
System Bus Interface	Interface that is neither master nor slave; enables specialized connections to a bus.
Top Level Port	An external interface connection to a component/module. Scalar if a 1-bit port, bus if a multiple-bit port.

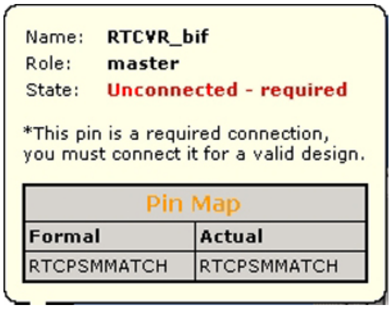
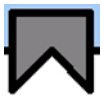
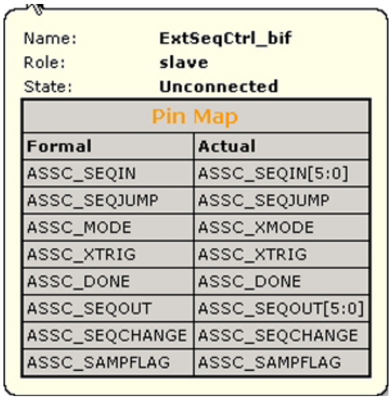
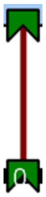
Canvas Icons

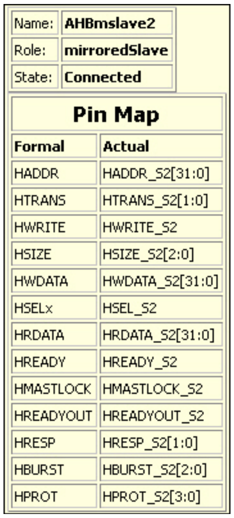
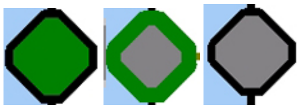
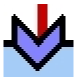
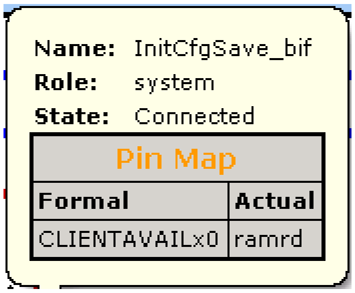
Hover your pointer over any icon in the SmartDesign Canvas view to display details.

Icon	Description
	Representation of an instance in your design. An instance is a component that has been added to your SmartDesign component. The name of the instance appears at the top and the name of the generic component at the bottom. The instance type is indicated by an icon inside the instance. There are specific icons for instances from SmartDesign, HDL,

Icon	Description
	and ViewDraw. The instance icon at left indicates a Microsemi SoC core.
 CoreAHBLite_0 IP M0 M1 S0 S2 S3 S6 S7 S9 S10 HCLK HRESETN REMAP_M0	Bus instance; you can click and drag the end of a bus instance to resize it; also, the bus instance will resize based on the number of instances that you connect to it.
	Optional unconnected pin. Required pins are red.
	Connected pin
	Pin with default Tie Off
	Pin tied low
	Pin tied high
	Pin inverted
	Pin marked as unused
	Pin tied to constant

Icon	Description
 Name: CoreAhbSram_0 Instance of: CoreAhbSram Type: IP Class: Regular Vendor: Actel Library: DirectCore Core Name: CoreAhbSram Version: 1.4.104 Number of ports: 13 Pin: HCLK IN Pin: HRESETn IN Pin: HSEL IN Pin: HWRITE IN Pin: HREADYIN IN Pin: HREADY OUT Pin: HTRANS[1:0] IN Pin: HSIZE[2:0] IN Pin: HWDATA[31:0] IN Pin: HADDR[14:0] IN Pin: HRESP[1:0] OUT Pin: HRDATA[31:0] OUT Pin: AHBslave SLAVE	Instance details. If there are less than twenty ports, they are listed in the details.
 Bus Net: DataB[1:0] sd_acc DataB[1:0] subtr_1_0 DataB[1:0]	Bus Net details.
	Master bus interface icon. A master is a bus interface that initiates a transaction on a bus interface net. An unconnected master BIF with REQUIRED connection is red (shown at left). A master BIF with unconnected OPTIONAL connection is gray.

<table>
<tr><th>Icon</th><th>Description</th></tr>
<tr><td>Name: RTCVR_bif
Role: master
State: Unconnected - required
*This pin is a required connection, you must connect it for a valid design.<table><tr><th colspan="2">Pin Map</th></tr><tr><th>Formal</th><th>Actual</th></tr><tr><td>RTCPSMMATCH</td><td>RTCPSMMATCH</td></tr></table></td><td>Master BIF details, showing name, role, and state.
The Pin Map shows the Formal name of the pin assigned by the component (in this example, RCCLKOUT) and the Actual, or representative name assigned by the user (CLKOUT).</td></tr>
<tr><td></td><td>Slave BIF (shown at left).
Unconnected slave icons with REQUIRED connections are red.
Unconnected slave icons with OPTIONAL connections are gray.</td></tr>
<tr><td>Name: ExtSeqCtrl_bif
Role: slave
State: Unconnected<table><tr><th colspan="2">Pin Map</th></tr><tr><th>Formal</th><th>Actual</th></tr><tr><td>ASSC_SEQIN</td><td>ASSC_SEQIN[5:0]</td></tr><tr><td>ASSC_SEQJUMP</td><td>ASSC_SEQJUMP</td></tr><tr><td>ASSC_MODE</td><td>ASSC_XMODE</td></tr><tr><td>ASSC_XTRIG</td><td>ASSC_XTRIG</td></tr><tr><td>ASSC_DONE</td><td>ASSC_DONE</td></tr><tr><td>ASSC_SEQOUT</td><td>ASSC_SEQOUT[5:0]</td></tr><tr><td>ASSC_SEQCHANGE</td><td>ASSC_SEQCHANGE</td></tr><tr><td>ASSC_SAMPFLAG</td><td>ASSC_SAMPFLAG</td></tr></table></td><td>Slave BIF details, showing name, role, and state.
The Pin Map shows the Formal name of the pin assigned by the component (in this example, RCCLKOUT) and the Actual, or representative name assigned by the user (CLKA).</td></tr>
<tr><td></td><td>Master-slave bus interface connection</td></tr>
</table>

Icon	Description
 Name: AHBmslave2 Role: mirroredSlave State: Connected Pin Map Formal | Actual HADDR | HADDR_S2[31:0] HTRANS | HTRANS_S2[1:0] HWRITE | HWRITE_S2 HSIZE | HSIZE_S2[2:0] HWDATA | HWDATA_S2[31:0] HSELx | HSEL_S2 HRDATA | HRDATA_S2[31:0] HREADY | HREADY_S2 HMASTLOCK | HMASTLOCK_S2 HREADYOUT | HREADYOUT_S2 HRESP | HRESP_S2[1:0] HBURST | HBURST_S2[2:0] HPROT | HPROT_S2[3:0]	Master-slave bus interface connection details.
	Groups of pins in an instance. Fully connected groups are solid green. Partially connected groups are gray with a green outline. Unconnected groups (no connections) are gray with a black outline.
	A system BIF is the bus interface that does not have a simple input/output relationship on both master/slave. This could include signals that only drive the master interface, or only drive the slave interface, or drive both the master and slave interfaces.
 Name: InitCfgSave_bif Role: system State: Connected Pin Map Formal | Actual CLIENTAVAILx0 | ramrd	System BIF details, showing name, role, and state. The Pin Map shows the Formal name of the pin assigned by the component (in this example, CLIENTAVAILx0), and the Actual name assigned by the user (in this example:

Icon	Description
	ramrd).
⊠	Pad port icon; indicates a hardwired chip-level pin

Create Core from HDL

You can instantiate any HDL module and connect it to other blocks inside SmartDesign. However, there are situations where you may want to extend your HDL module with more information before using it inside SmartDesign.

- If you have an HDL module that contains configurable parameters or generics.
- If your HDL module is intended to connect to a processor subsystem and has implemented the appropriate bus protocol, then you can add a bus interface to your HDL module so that it can easily connect to the bus inside of SmartDesign.

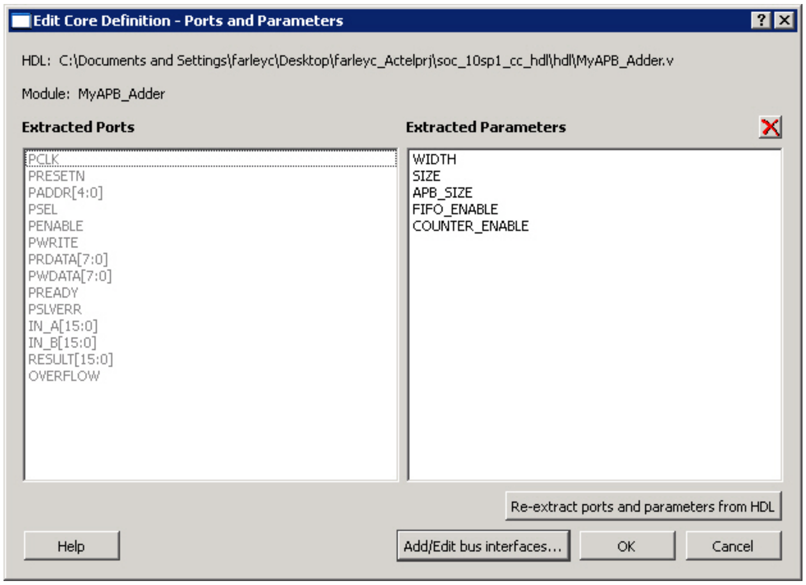
To create a core from your HDL:

1. Import or create a new HDL source file; the HDL file appears in the Design Hierarchy.
2. Select the HDL file in the Design Hierarchy and click the HDL+ icon or right-click the HDL file and choose **Create Core from HDL**.

The Edit Core Definition – Ports and Parameters dialog appears. It shows you which ports and parameters were extracted from your HDL module.

3. Remove parameters that are not intended to be configurable by selecting them from the list and clicking the X icon. Remove parameters that are used for internal variables, such as state machine enumerations.

If you removed a parameter by accident, click **Re-extract ports and parameters from HDL file** to reset the list so it matches your HDL module.



Edit Core Definition - Ports and Parameters

HDL: C:\Documents and Settings\farleyc\Desktop\farleyc_Actelprj\soc_10sp1_cc_hdl\hdl\MyAPB_Adder.v

Module: MyAPB_Adder

Extracted Ports	Extracted Parameters
PCLK PRESETN PADDR[4:0] PSEL PENABLE PWRITE PRDATA[7:0] PWDATA[7:0] PREADY PSLVERR IN_A[15:0] IN_B[15:0] RESULT[15:0] OVERFLOW	WIDTH SIZE APB_SIZE FIFO_ENABLE COUNTER_ENABLE

Re-extract ports and parameters from HDL
Help
Add/Edit bus interfaces...
OK
Cancel

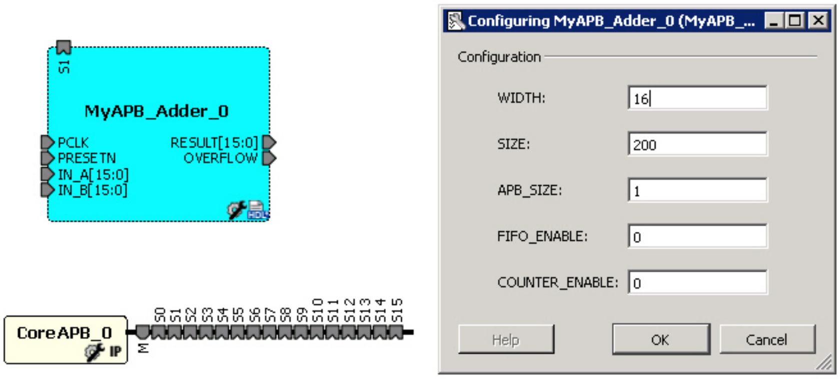
Figure 30 · Edit Core Definition - Ports and Parameters Dialog Box

4. (Optional) Click **Add/Edit Bus Interfaces** to add bus interfaces to your core.

After you have specified the information, your HDL turns into an HDL+ icon in the Design Hierarchy. Click and drag your HDL+ module from the Design Hierarchy to the **Canvas**.

If you added bus interfaces to your HDL+ core, then it will show up in your SmartDesign with a bus interface pin that can be used to easily connect to the appropriate bus IP core.

If your HDL+ has configurable parameters then double-clicking the object on the Canvas invokes a configuration dialog that enables you to set these values. On generation, the specific configuration values per instance are written out to the SmartDesign netlist.



Configuration

WIDTH: 16

SIZE: 200

APB_SIZE: 1

FIFO_ENABLE: 0

COUNTER_ENABLE: 0

Figure 31 · HDL+ Instance and Configuration Dialog Box

You can right-click the instance and choose **Modify HDL** to open the HDL file inside the text editor.

Edit Core Definition

You can edit your core definition after you created it by selecting your HDL+ module in the design hierarchy and clicking the HDL+ icon.

Remove Core Definition

You may decide that you do not want or need the extended information on your HDL module. You can convert it back to a regular HDL module. To do so, right-click the HDL+ in the Design Hierarchy and choose **Remove Core Definition**. After removing your definition, your instances in your SmartDesign that were referencing this core must be updated. Right-click the instance and choose **Replace Component for Instance**.

Create HDL and Create HDL Stimulus

You can use HDL (hardware description language) files to simulate and model your device.

To create an HDL file:

1. Open your project.
2. In the Design Flow window, double-click **Create HDL** or **Create HDL Stimulus**. The Create new Verilog (or VHDL) file dialog box opens.
3. Enter a Name and click **OK**. (Do not enter a file extension; Libero SoC adds one for you.) The HDL Editor workspace opens.
4. After creating your HDL file, click the **Save** button to save your file to the project . Your HDL file is saved to your project in the Files window /hdl directory.

Using the HDL Editor

The HDL Editor is a text editor designed for editing HDL source files. In addition to regular editing features, the editor provides a syntax checker.

You can have multiple files open at one time in the HDL Editor workspace. Click the tabs to move between files.

Editing

Editing functions are available in the **Edit** menu. Available functions include cut, copy, paste, find, and replace. These features are also available in the toolbar.

Saving

You must save your file to add it to your Libero SoC project. Select **Save** in the **File** menu, or click the **Save** icon in the toolbar.

Printing

Print is available from the **File** menu and the toolbar.

Note: Note: To avoid conflicts between changes made in your HDL files, Microsemi recommends that you use one editor for all of your HDL edits.

HDL Syntax Checker

To run the syntax checker:

In the **Files** list, double-click the HDL file to open it. Right-click in the body of the HDL editor and choose **Check HDL File**.

The syntax checker parses the selected HDL file and looks for typographical mistakes and syntactical errors. Warning and error messages for the HDL file appear in the Libero SoC Log Window.

Commenting Text

You can comment text as you type in the HDL Editor, or you can comment out blocks of text by selecting a group of text and applying the Comment command.

To comment or uncomment out text:

1. Type your text.
2. Select the text.
3. Right-click inside the editor and choose **Comment Out** or **Uncomment**.

Importing HDL Source Files

To import an HDL source file:

1. In the Design Flow window, right-click **Create HDL** and choose **Import Files**.
2. In **Look in**, navigate to the drive/folder that contains the file.
3. Select the file to import and click **Open**.

Mixed-HDL Support in Libero SoC

You must have ModelSim PE or SE to use mixed HDL in the Libero SoC. Also, you must have Synplify Pro to synthesize a mixed-HDL design.

When you create a project, you must select a preferred language. The HDL files generated in the flow (such as the post-layout netlist for simulation) are created in the preferred language.

The language used for simulation is the same language as the last compiled testbench. (E.g. if tb_top is in verilog, <fam>.v is compiled.)

If your preferred language is Verilog, the post-synthesis and post-layout netlists are in Verilog 2001.

SmartDesign Testbench

Use a SmartDesign to instantiate and connect stimulus cores or modules to drive your Root design. Double-click **Create SmartDesign Testbench** in the Design Flow window to add a new SmartDesign testbench to your project.

New testbench files appear in the Stimulus Hierarchy.

The SmartDesign Testbench automatically instantiates your root design into the Canvas.

You can also instantiate your own stimulus HDL or simulation models into the SmartDesign Testbench Canvas and connect it to your DUT (design under test), or instantiate Simulation Cores from the Catalog. Simulation cores are basic cores that are useful for stimulus, such as driving clocks, resets, and pulses.

Click the Simulation Mode checkbox in the Catalog to instantiate simulation cores.

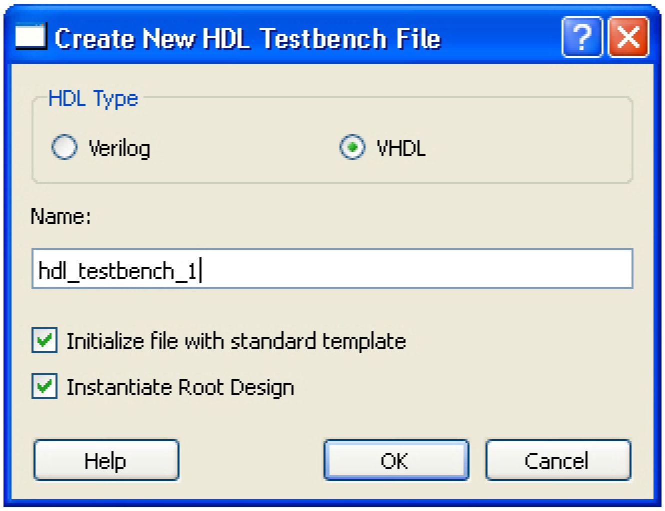
HDL Testbench

Double-click Create HDL Testbench to open the Create New HDL Testbench dialog box. The dialog box enables you to create a new testbench file and gives you the option to include standard testbench content and your design data.

Set your HDL Type, specify a name, select the data options and click OK to create a new testbench.

Initialize file with standard template populates the new HDL file with basic headers and Clock/Reset driver, as in the header of the example file below.

Instantiate Root Design includes your root design information in the new file. It includes architectural, constant, signal, component, clock, and port information.



Create New HDL Testbench File

HDL Type

○ Verilog ● VHDL

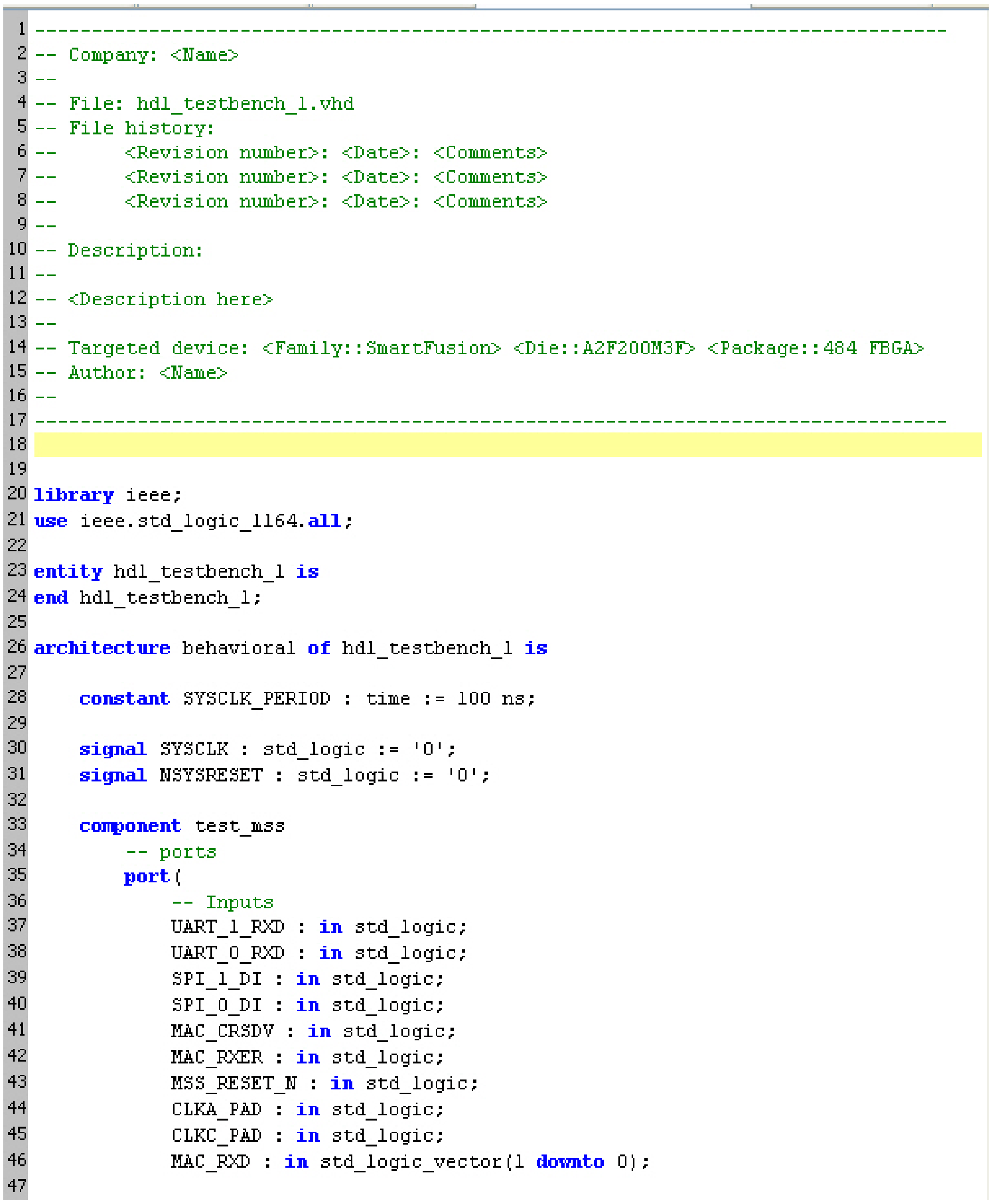
Name:

hdl_testbench_1

☑ Initialize file with standard template

☑ Instantiate Root Design

Figure 32 · Create HDL Testbench Dialog Box



```
----------------------------------------------------------------------------------
-- Company: <Name>
--
-- File: hdl_testbench_1.vhd
-- File history:
--      <Revision number>: <Date>: <Comments>
--      <Revision number>: <Date>: <Comments>
--      <Revision number>: <Date>: <Comments>
--
-- Description:
--
-- <Description here>
--
-- Targeted device: <Family::SmartFusion> <Die::A2F200M3F> <Package::484 FBGA>
-- Author: <Name>
--
----------------------------------------------------------------------------------


library ieee;
use ieee.std_logic_1164.all;

entity hdl_testbench_1 is
end hdl_testbench_1;

architecture behavioral of hdl_testbench_1 is

    constant SYSCLK_PERIOD : time := 100 ns;

    signal SYSCLK : std_logic := '0';
    signal NSYSRESET : std_logic := '0';

    component test_mss
        -- ports
        port(
            -- Inputs
            UART_1_RXD : in std_logic;
            UART_0_RXD : in std_logic;
            SPI_1_DI : in std_logic;
            SPI_0_DI : in std_logic;
            MAC_CRSDV : in std_logic;
            MAC_RXER : in std_logic;
            MSS_RESET_N : in std_logic;
            CLKA_PAD : in std_logic;
            CLKC_PAD : in std_logic;
            MAC_RXD : in std_logic_vector(1 downto 0);

```

Figure 33 · HDL Testbench Example - Standard Template and Root Design Enabled

View/Configure Firmware Cores

The Design Firmware tab lists the compatible firmware for the hardware that you have in your design. In the Design Flow tab, expand **Create Design** and double-click **View/Configure Firmware Cores** to view the DESIGN_FIRMWARE tab.

The Firmware table lists the compatible firmware and drivers based on the hardware peripherals that you have used in your design. Each row represents a compatible firmware core. The columns are:

- **Generate** - Allows you to choose whether you want the files for this firmware core to be generated on disk. You may decide to use your own firmware rather than Microsemi's provided firmware cores.
- **Instance Name** - This is the name of the firmware instance. This maybe helpful in distinguishing firmware cores when you have multiple firmware of the same Vendor:Library:Name:Version (VLNV) in your design.

- **Core Type** - Firmware Core Type is the Name from the VLNV id of the core.
- **Version** - Firmware Core Version
- **Compatible Hardware Instance** - The hardware instance that is compatible with this firmware core.

Generating Firmware

Click the Generate icon to export the Firmware and Software workspace for your project. The firmware is generated into <project>\firmware and the software workspace is exported to <project>\<toolchain>.

The firmware drivers are also copied into the <toolchain> folder so that each workspace is self-contained.

Configuring Firmware

Firmware that have configurable options will have a wrench icon in the row. Click the wrench icon or double-click the row to configure the firmware.

It is important that you check the configuration of your firmware if they have configurable options. They may have options that target your toolchain (Keil, IAR), or your processor that are vital configuration options to getting your system to work properly.

Downloading Firmware

The MSS Configurator attempts to find compatible firmware located in the IP Vault located on your disk, as well as firmware in the IP Repository via the Internet.

If compatible firmware is found in the IP repository, the row will be italicized, indicating that it needs to be downloaded. To download all your firmware click the **Download All Firmware** icon in the vertical toolbar

Changing Versions

There will often be multiple versions of a firmware available for a particular firmware core. For a new design, the MSS Configurator will pick the latest compatible version.

However, once the firmware has been added to your design, the tool will not automatically change to the latest version if one becomes available. You can manually change to the latest version by selecting the drop down in the Version column.

Note: Note: If the latest version is italicized, you will need to download the firmware after selecting it.

Generating Sample Projects

Firmware cores are packaged with sample projects that demonstrate their usage. They are packaged for specific tool chains, such as Keil and IAR.

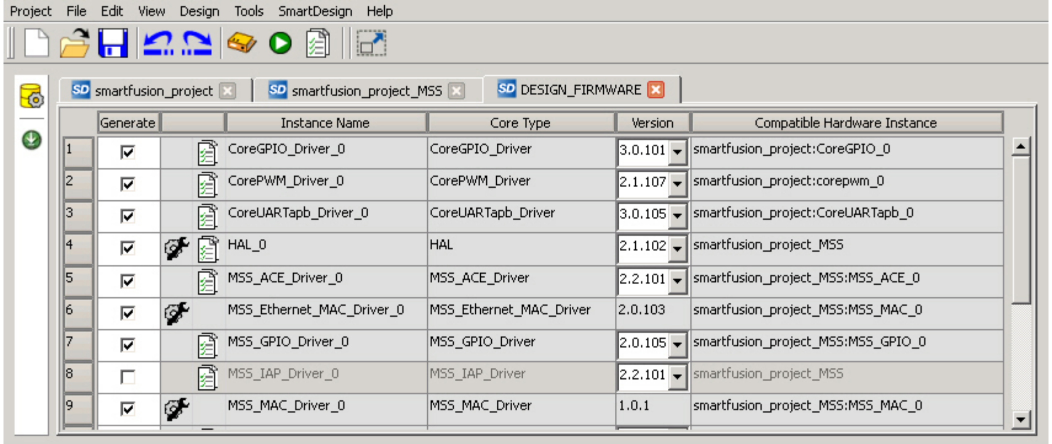
To generate a sample project, click the sample project icon in the row and choose Generate Sample Project followed by the tool chain you are targeting. You will be prompted to select the destination folder for the sample project.

Once this project is generated you can use it as a starting point in your Software IDE tool or use the example project as a basis on how to use the firmware driver.

Peripherals in the Fabric

Libero SoC also attempts to find compatible firmware for soft peripherals that you have added in your top-level SmartDesign.

To enable this, you must set the top level SmartDesign as root in Libero SoC. Right-click your top level design in the Design Hierarchy and choose **Set as Root**. The root component will have its name bolded if it is root.


Project File Edit View Design Tools SmartDesign Help
SD smartfusion_project
SD smartfusion_project_MSS
SD DESIGN_FIRMWARE

	Generate		Instance Name	Core Type	Version	Compatible Hardware Instance
1	☑		CoreGPIO_Driver_0	CoreGPIO_Driver	3.0.101	smartfusion_project:CoreGPIO_0
2	☑		CorePWM_Driver_0	CorePWM_Driver	2.1.107	smartfusion_project:corepwm_0
3	☑		CoreUARTapb_Driver_0	CoreUARTapb_Driver	3.0.105	smartfusion_project:CoreUARTapb_0
4	☑		HAL_0	HAL	2.1.102	smartfusion_project_MSS
5	☑		MSS_ACE_Driver_0	MSS_ACE_Driver	2.2.101	smartfusion_project_MSS:MSS_ACE_0
6	☑		MSS_Ethernet_MAC_Driver_0	MSS_Ethernet_MAC_Driver	2.0.103	smartfusion_project_MSS:MSS_MAC_0
7	☑		MSS_GPIO_Driver_0	MSS_GPIO_Driver	2.0.105	smartfusion_project_MSS:MSS_GPIO_0
8	☐		MSS_IAP_Driver_0	MSS_IAP_Driver	2.2.101	smartfusion_project_MSS
9	☑		MSS_MAC_Driver_0	MSS_MAC_Driver	1.0.1	smartfusion_project_MSS:MSS_MAC_0

Figure 34 · Firmware Cores Tab (DESIGN_FIRMWARE)

See Also

Develop Firmware - Write Application Code

Libero SoC Frequently Asked Questions

Running Libero SoC from your Software Tool Chain

Software IDE Integration

Project Sources

Project sources are any design files that make up your design. These can include schematics, HDL files, simulation files, testbenches, etc. Anything that describes your design or is needed to program the device is a project source.

Source files appear in the Project Flow window. The Design Hierarchy tab displays the structure of the design modules as they relate to each other, while the Files tab displays all the files that make up the project.

The design description for a project is contained within the following types of sources:

- Schematics
- HDL Files (VHDL or Verilog)
- SmartDesign components

One source file in the project is the top-level source for the design. The top-level source defines the inputs and outputs that will be mapped into the devices, and references the logic descriptions contained in lower-level sources. The referencing of another source is called an *instantiation*. Lower-level sources can also instantiate sources to build as many levels of logic as necessary to describe your design.

File Linking

The Project Manager enables you to link to files not managed in your Libero project. Linked files are useful if you want to preserve a file in an archive, or if more than one person is using a file and it is impractical to store it on your local machine. If you link to external files and rename your project, the Project Manager asks if you want to copy the external files into your project or continue using the link. Note that some files (such as schematics) cannot be linked.

Some project sources can be imported.

Sources for your project can include:

Source	File Extension
Schematic	*.1-9

Source	File Extension
Verilog Module	*.v
VHDL Entity	*.vhd
SmartDesign Component	*.vhd
Testbench	*.vhd
Stimulus	*.tim
Programming Files	*.afm; *.prb

See Also

Creating HDL Sources

Generating a Bitstream file

Generating Programming files

Designing with Designer Block Components

Designer Blocks (also generically called "components") enable you to partition a design and optimize critical sections. You can reuse them later in new applications, ensuring consistent performance. Designing with blocks enables multiple designers to work independently on parts of a single design.

Designer Block Advantages

- You can focus on the timing of critical blocks and ensure the timing across the blocks meets requirements before proceeding to the top-level flow.
- Changes in other blocks have no impact on your own block, you can re-use your block without re-calculating the timing.
- The block can be re-used in multiple designs
- Shorter verification time. You need to re-verify only the portion of the design that has changed.

Designer Block Features

- You can create a Designer Block with or without I/Os.
- A Designer Block can be synthesized, simulated, and placed-and-routed the same way as a regular design.
- You can lock the place-and-route of the Designer Block to ensure performance does not change.
- Performance and place-and-route can be fixed absolutely; however these rules can be relaxed gradually, if necessary, to ensure that you can integrate the Designer Block into your <top> project.
- You can use all the features in Designer Blocks in SmartDesign.

Use Designer Blocks When

- The design is congested (uses 90% of the resources on a given die).
- You have difficulty meeting timing by doing the design in its entirety. Blocks enable you to compartmentalize the design and optimize sections before you optimize the entire design.
- You want to re-use some elements of your design.
- You want to use the identical elements multiple times in a single design.

You cannot use Designer Blocks with all families, they are family and die specific; if your Designer Block has I/Os it is also package specific.

Supported families

IGLOO, ProASIC3, SmartFusion2, SmartFusion, Fusion

Designer Blocks and Synthesis

You must run the synthesis tool in No I/O mode when you create your component. The Designer Block is not a full design; Libero SoC sets this option for Synplify if you Enable Designer Block creation.

When you Publish a Designer Block, the Project Manager creates a timing shell that enables the synthesis tool to better synthesize the <top> project. The timing shell is named <blockname>_syn.v(.vhd) if you are using Synplify or <blockname>_pre.v(.vhd) if you are using Precision.

When you are working in your <top> project, the synthesis tool does not know how many globals you have in your Designer Block, or if there will be clock sharing. The synthesis tool promotes as many globals as it can and if you have globals in the Designer Block you will exceed the total number of globals allowed in your device.

In this case, you must limit the number of globals added by the synthesis tool so that the total number (Designer Block plus <top> project) does not exceed the number available on your device.

To add an internal global, you can use either the Synplify constraints editor (SCOPE) or an SDC file.

For example, to add a CLKINT after a CLK port, the command is:

```
define_attribute {n:CLK} syn_insert_buffer {CLKINT}
```

See Also

Creating a component in Designer

Creating a component in Libero SoC

Managing I/Os in a Designer Block Component

If you use I/Os in your Designer Block, use the following rules:

- If the I/O is placed in the block, placement and VCCI of the I/O cannot be changed in the <top> design.
- The register combining option cannot be changed in the <top> design.
- Attributes and Vref pins can be changed if the values are legal (the I/O will not be unplaced).

Globals and Designer Block Components

You must manage your globals when creating a Designer Block to ensure that you have some available after you import the Block into your <top> project.

There is no limit to the number of globals you can use in a Designer Block.

Global Sharing

You can share a global between the Designer Block and the <top> project. You must:

- Use an internal global in the Designer Block.
- Drive the global port in the <top> project with a global net.

Libero SoC removes the internal global and re-routes the entire net.

You can use other global macros in the Designer Block, but you cannot share them with the <top> project.

Global Sharing with SmartFusion, IGLOO, ProASIC3 and Fusion - Use CLKINT in the Designer Block to share the global in the component with the <top> project.

See the list of Physical Design Constraint (PDC) files for more information on how to assign constraints.

Local Clock

You can use local clocks in your component to save on globals, but you may need to do some floorplanning in your <top> project.

Limitations

When you create your block, you cannot assign a port-connected net to a local clock.

The routing for local clocks from the blocks cannot always be preserved.

For all other families, local clocks are rerouted only if they are used in more than one block. The local clock constraint is preserved and the only difference in the routing is from the driver to the entry point of the clock network (when it gets to the clock network you end up with the same routing since the macros are locked in the same location).

Designer Block Compile Report

If you instantiate Designer Blocks in your design, the Compile report includes a description of the blocks you used. The report appears in the Log window after Compile is complete.

The report lists the name of the module, the name of the instance, the number of macros and nets used in the blocks, and information on how conflicts between blocks were resolved by the Compile options or PDC commands (if any). For example:

```
Block Information Report :
==========================
Conflict resolution from Compile options :
==========================================
  Placement : Resolve conflict/Keep and Lock non conflicting placement
   Routing : Resolve conflict/Keep and Lock non conflicting routing
  -----------------------------------------------------
   Block Name : core1
  Instance Name : core1_inst
  | Locked | Total
   ---------------------------
   Instances | 4 | 4 (100.00%)
   Nets | 3 | 3 (100.00%)
  -----------------------------------------------------
  Block Name : core1
  Instance Name : core11_inst
  PDC Constraints :
   =================
   Move : move_block -inst_name {core11_inst} -left 10 -up 0 -non_logic UNPLACE
  | Locked | Total
   ---------------------------
   Instances | 4 | 4 (100.00%)
  Nets | 0 | 3 (0.00%)
```

Designer Block Component Limitations

If you instantiate the same Designer Block many times in the <top> design, only the first instance retains the place-and-route information (if it has any); the others do not. Only the netlist is preserved.

To preserve the relative placement and routing of other blocks you must move the blocks using a PDC command. This PDC file must be imported as a source file along with the netlist(s) and CDB files. If possible, routing is preserved when you move the blocks with a PDC command.

See the move_block PDC command for more information.

Creating a Designer Block Component in Libero SoC

Creating a Designer Block Component in Libero SoC

You must create two Libero SoC projects in order to instantiate your Designer Block in Libero SoC: one to create and publish your Designer Block, and another in which to instantiate your Designer Block. This section describes how to create your Designer Block.

See Instantiating a Designer Block in Libero SoC for more information.

The general design flow for creating a Designer Block in Libero SoC is shown in the figure below.

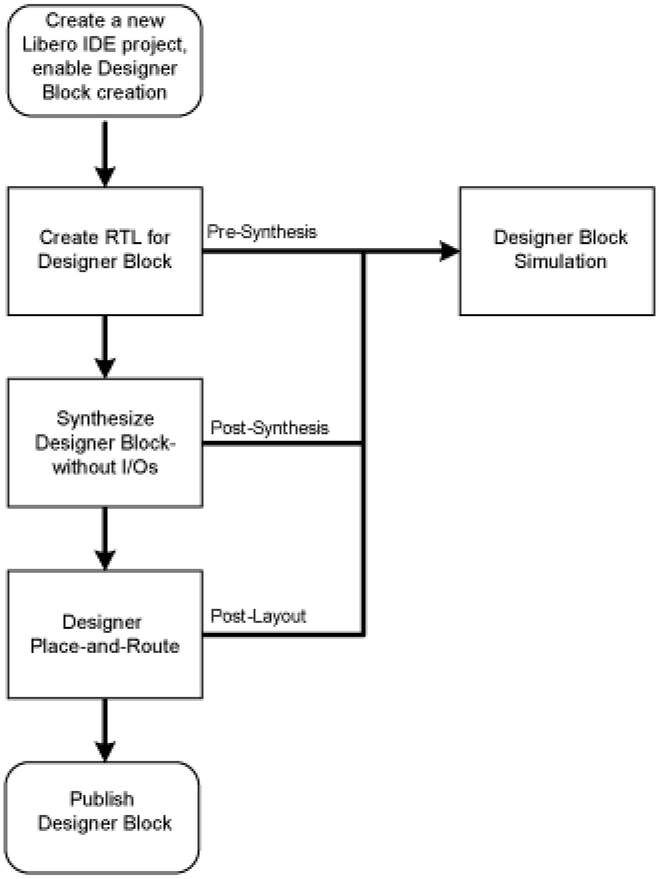

Create a new Libero IDE project, enable Designer Block creation
Create RTL for Designer Block
Pre-Synthesis
Designer Block Simulation
Synthesize Designer Block-without I/Os
Post-Synthesis
Designer Place-and-Route
Post-Layout
Publish Designer Block

Figure 35 · Create a Designer Block Flow in Libero SoC

To create a Designer Block in Libero SoC with a new design:

1. Start a new project. You must select a family that supports Block designs (IGLOO, ProASIC3, SmartFusion2, SmartFusion, Fusion). After your project opens, from the **Project** menu, choose **Settings > Flow**, and click the **Enable Designer Block creation** checkbox.
2. Create a design in Libero SoC (standard design flow - create RTL, synthesize, run place-and-route and generate the block using Designer).

To create a Designer Block in the Libero SoC with an existing design, open your design and from the **Project** menu, choose **Setting > Flow**, and click the **Enable Designer Block creation** checkbox. Note that your design must use a device family that supports Designer Blocks (IGLOO, ProASIC3, SmartFusion2, SmartFusion, Fusion).

Instantiating a Designer Block in Libero SoC

You must have two projects in order to instantiate your Designer Block in Libero SoC: one to create and publish your Designer Blocks, and another in which to instantiate your Designer Block. This topic and the flow shown in the figure below describe how to instantiate your Designer Block in the Libero SoC.

See Creating a Designer Block in Libero SoC for information on how to create a Designer Block. You can also import your Designer Blocks into SmartDesign.

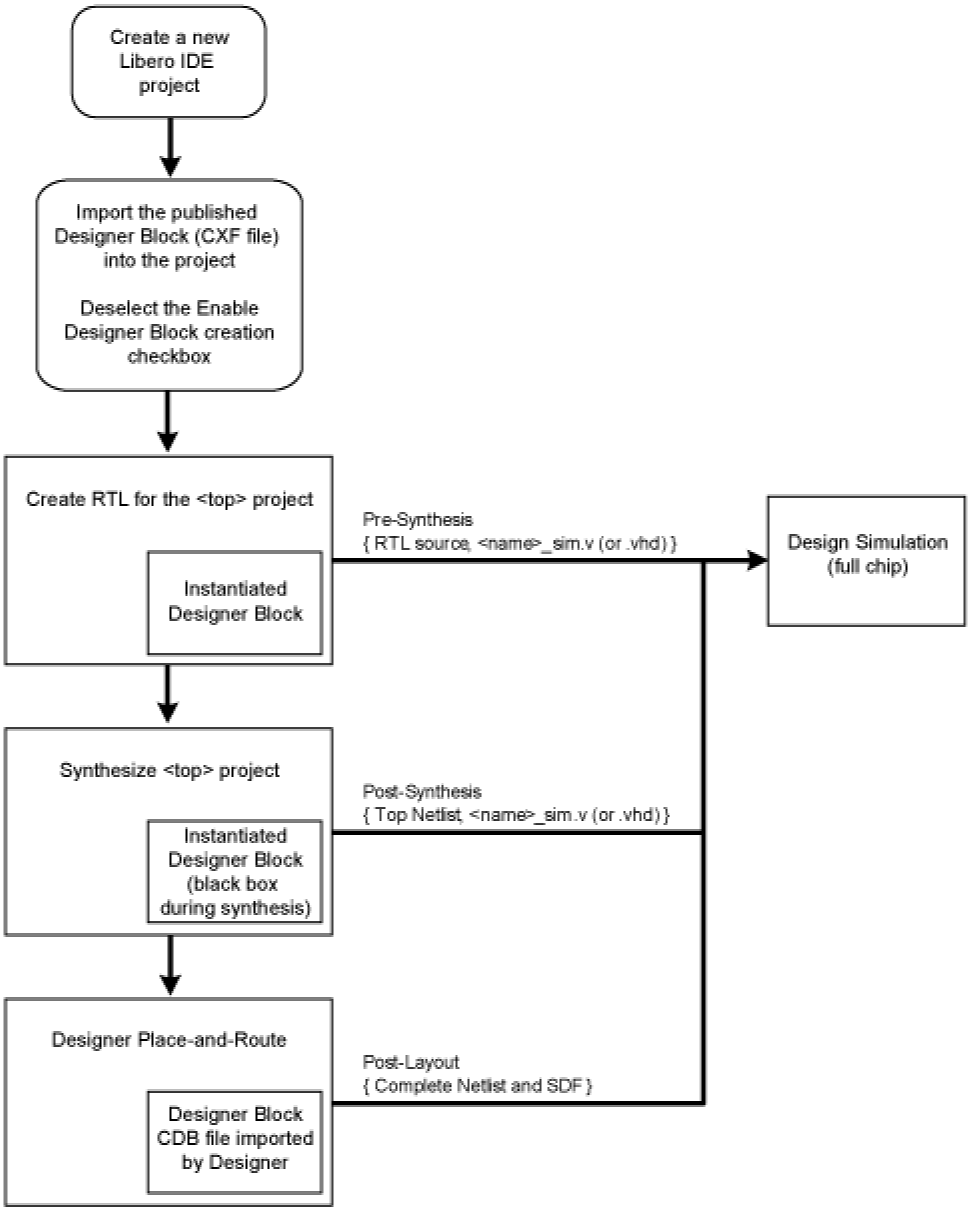

Create a new Libero IDE project
Import the published Designer Block (CXF file) into the project
Deselect the Enable Designer Block creation checkbox
Create RTL for the <top> project
Instantiated Designer Block
Pre-Synthesis
{ RTL source, <name>_sim.v (or .vhd) }
Design Simulation (full chip)
Synthesize <top> project
Instantiated Designer Block (black box during synthesis)
Post-Synthesis
{ Top Netlist, <name>_sim.v (or .vhd) }
Designer Place-and-Route
Designer Block CDB file imported by Designer
Post-Layout
{ Complete Netlist and SDF }

Figure 36 · Libero SoC Designer Block Instantiation Flow

To instantiate (import) a Designer Block in Libero SoC, import your design netlist and CXF file(s). The CXF file imports all the files you need for your Designer Block. After you import your files, the design flow is the same as regular Libero SoC designs. There is no limit to the number of CXF files you can import, but you cannot import the same Designer Block more than once, and the family and device for your imported block must match your project.

After you import the CXF file, the Project Manager displays the imported files in the Design Hierarchy tab.

The Designer Block(s) you instantiate must have the same family and die (and package, if it contains I/Os) as your current <top> project. If the family, die, and package do not match, Libero SoC asks if you want to change the current setting to match the one from the Designer Block.

The Project Manager passes all the Designer Block files to Designer automatically.

Note: Note

- Disable Designer Block creation when you import a component into your <top> project. If you are using a Designer Block component to create another Designer Block, leave it enabled.
- If you already have an HDL component with the same name as the one you imported, the new Designer Block component is not be used by default. You must and right-click the Designer Block component in the Project Manager and choose **Use this file** to make it use your Designer Block.

RTL Simulation

If you wish to perform pre-layout simulation, in the Design Flow Window, under Verify Pre-Synthesized design, double-click Simulate.

The default tool for RTL simulation in Libero SoC is ModelSim AE.

ModelSimTM AE is a custom edition of ModelSim PE that is integrated into Libero SoC's design environment. ModelSim for Microsemi is an OEM edition of Model Technology Incorporated's (MTI) tools. ModelSim for Microsemi supports VHDL or Verilog. It only works with Microsemi libraries and is supported by Microsemi.

Other editions of ModelSim are supported by Libero SoC. To use other editions of ModelSim , simply do not install ModelSim AE from the Libero SoC CD.

Note: Note: ModelSim for Microsemi comes with its own online help and documentation. After starting ModelSim, click the *Help* menu.

See the following topics for more information on simulation in Libero SoC:

- Simulation Options
- Selecting a Stimulus File for Simulation
- Selecting additional modules for simulation
- Performing Functional Simulation

Simulation Options

You can set a variety of simulation options for your project.

To set your simulation options:

1. From the **Project** menu, choose **Project Settings**.
2. Click the simulation option you wish to edit: **DO file**, **Waveforms**, or **Vsim commands**.
3. Click **Close** to save your settings.

DO File

- **Use automatic Do file -** Select to execute the wave.do or other specified Do file. Use the wave.do file to customize the ModelSim Waveform window display settings.
- **Simulation Run Time -** Specify how long the simulation should run in nanoseconds. If the value is 0, or if the field is empty, there will not be a run command included in the run.do file.
- **Testbench module name -** Specify the name of your testbench entity name. Default is "testbench," the value used by WaveFormer Pro.
- **Top Level instance name -** Default is <top_0>, the value used by WaveFormer Pro. The Libero SoC replaces <top> with the actual top level macro when you run ModelSim.
- **Generate VCD file** - Select this checkbox to have ModelSim automatically generate a VCD file based on the current simulation. VCD files can be used in SmartPower. For best results, we recommend that a postlayout simulation be used to generate the VCD.
- **VCD filename** - Specify the name of the VCD file that will be automatically generated by ModelSim
- **User defined DO file** - Available if you opt not to use the automatic DO file. Input the path or browse to your user-defined DO file.
- **DO Command parameters** - Text in this field is added to the DO command.

Waveforms

- **Include DO file** - Including a DO file enables you to customize the set of signal waveforms that will be displayed in ModelSim.
- **Display waveforms for** - You can display signal waveforms for either the top-level testbench or for the design under test. If you select top-level testbench then Libero SoC outputs the line 'add wave /testbench/*' in the DO file run.do. If you select DUT then Libero SoC outputs the line 'add wave /testbench/*' in the run.do file.
- **Log all signals in the design** - Saves and logs all signals during simulation.

Vsim Commands

- **SDF timing delays** - Select Minimum, Typical, or Maximum timing delays in the back-annotated SDF file.
- **Resolution**: The default is family-specific, but you can customize it to fit your needs.
  Some custom simulation resolutions may not work with your simulation library. For example, simulation resolutions above 1 ps will cause errors if you are using ProASIC3 devices (the simulation errors out because of an infinite zero-delay loop). Consult your simulation help for more information on how to work with your simulation library and detect infinite zero-delay loops caused by high resolution values.

Family	Default Resolution
ProASIC3	1 ps
IGLOO	1 ps
SmartFusion and Fusion	1 ps

- **Additional options:** Text entered in this field is added to the vsim command.

Simulation Libraries

- **Verilog (or VhDL) library path** - Enables you to choose the default library for your device, or to specify your own library. Enter the full pathname of your own library to use it for simulation.
- **Restore Defaults**: Restores factory settings.

Selecting a Stimulus File for Simulation

Before running simulation, you must associate a testbench. If you attempt to run simulation without an associated testbench, the Libero SoC Project Manager asks you to associate a testbench or open Model*Sim* without a testbench.

To associate a stimulus:

1. Run simulation or in the Design Flow window under Verify Pre-Synthesized Design right-click **Simulate** and choose **Organize Input Files > Organize Stimulus Files**. The Organize Stimulus Files dialog box appears.
2. Associate your testbench(es):

In the Organize Stimulus Files dialog box, all the stimulus files in the current project appear in the Source Files in the Project list box. Files already associated with the block appear in the Associated Source Files list box.

In most cases you will only have one testbench associated with your block. However, if you want simultaneous association of multiple testbench files for one simulation session, as in the case of PCI cores, add multiple files to the Associated Source Files list.

To add a testbench: Select the testbench you want to associate with the block in the Source Files in the Project list box and click **Add** to add it to the Associated Source Files list.

To remove a testbench: To remove or change the file(s) in the Associated Source Files list box, select the file(s) and click **Remove**.

To order testbenches: Use the up and down arrows to define the order you want the testbenches compiled. The top level-entity should be at the bottom of the list.

3. When you are satisfied with the Associated Source Files list, click **OK**.

Selecting Additional Modules for Simulation

Libero SoC passes all the source files related to the top-level module to simulation .

If you need additional modules in simulation, in the Design Flow window right-click **Simulate** and choose **Organize Input Files > Organize Source Files**. The Organize Files for Simulation dialog box appears.

Select the HDL modules you wish to add from the Simulation Files in the Project list and click **Add** to add them to the Associated Stimulus Files list

Performing Functional Simulation

To perform functional simulation:

1. Create your testbench.
2. Right-click **Simulate** (in Design Flow window, Implement Design > Verify Post-Synthesis Implementation > Simulate) and choose **Organize Input Files > Organize Source Files** from the right-click menu.

In the Organize Files for Source dialog box, all the stimulus files in the current project appear in the Source Files in the Project list box. Files already associated with the block appear in the Associated Source Files list box.

In most cases you will only have one testbench associated with your block. However, if you want simultaneous association of multiple testbench files for one simulation session, as in the case of PCI cores, add multiple files to the Associated Source Files list.

To add a testbench: Select the testbench you want to associate with the block in the Source Files in the Project list box and click **Add** to add it to the Associated Source Files list.

To remove a testbench: To remove or change the file(s) in the Associated Source Files list box, select the file(s) and click **Remove**.

3. When you are satisfied with the Associated Simulation Files list, click **OK**.
4. To start ModelSim AE, right-click **Simulate** in the Design Hierarchy window and choose **Open Interactively**.

ModelSim starts and compiles the appropriate source files. When the compilation completes, the simulator runs for 1 µs and the Wave window opens to display the simulation results.

5. Scroll in the Wave window to verify that the logic of your design functions as intended. Use the zoom buttons to zoom in and out as necessary.
6. From the **File** menu, select **Quit**.

Performing DirectCore Functional Simulation

Libero SoC overwrites all the existing files of the Core when you import a DirectCore project (including testbenches). Save copies of your project stimulus files with new names if you wish to keep them.

You must import a DirectCore BFM file into the Libero SoC in order to complete functional simulation (the BFM is a stimulus file that you can edit to extend the testbench). VEC files are generated automatically from the BFM when you run ModelSim.

The SoC Project Manager overwrites your BFM file if you re-import your project. Edit and save your BFM outside the Libero SoC project to prevent losing your changes. After you re-import your DirectCore project, you can import your modified BFM again.

To perform functional simulation of a DirectCore project:

1. Right-click a stitched module of the DirectCore project and select **Set as root**.
2. To start ModelSim AE, right-click **Simulate** in the Design Hierarchy window and choose **Open Interactively**.

ModelSim starts and compiles the appropriate source files. When the compilation completes, the simulator runs for 1 µs and the Wave window opens to display the simulation results.

3. Scroll in the Wave window to verify that the logic of your design functions as intended. Use the zoom buttons to zoom in and out as necessary.
4. From the **File** menu, select **Quit**.

I/O Constraints - SmartFusion2

SmartFusion2 I/O constraints are PDC files. Note that for SmartFusion2 I/O constraint PDC files are separate from Floorplan constraint PDC files; if you have a PDC file that contains both I/O and Floorplan constraints then Libero SoC errors out with an invalid constraint error.

Libero SoC generates an I/O PDC file automatically if you explicitly add/modify your I/O Constraints in the post-Compile I/O Editor. Your new I/O PDC file is added to the project and marked as Used.

I/O Constraints enables you to:

Import Files - If you do not have a compiled project, double-click I/O Constraints to open the Import Files dialog box and import I/O constraint files (*.pdc files).

Create a New Constraint from Your Root Module - Double-click to create a new constraint if you already have a compiled project.

Link Files - Right-click **I/O Constraints** and choose **Link Files** to link PDC constraint files from other projects. Linked files are not copied into your local project directory; instead the path is stored in your project, enabling you (or others) to update the file separately from Libero SoC. If your linked file is updated then the Project Manager indicates that the file has been changed and asks you if you wish to recompile, as appropriate.

Linked files appear in your **Files window** (**View > Windows > Files**), where they can be opened, deleted from the project, updated, or unlinked and copied to your local project.

Once you import or generate an I/O Constraint file you can double-click the file in the **Design Flow** window (**Create Constraints > I/O Constraints > <filename>**) to open it in the I/O Constraint Editor, or right-click the file to:

Use for Compile - Includes the constraint file when you run Compile.

Open in I/O Editor - Opens the file in the I/O Editor.

Open in the Text Editor - Opens the file in the Text Editor so that you can update the code manually.

Delete from Project - Removes the file from the project.

Delete from Disk and Project - Removes the file from the project and deletes it from the disk.

Timing Constraints

Timing Constraints enables you to:

Import Files - Double-click Timing Constraints to open the Import Files dialog box and import timing constraint files (*.sdc files).

Link Files - Right-click **Timing Constraints** and choose **Link Files** to link SDC constraint files from other projects. Linked files are not copied into your local project directory; instead the path is stored in your project, enabling you (or others) to update the file separately from Libero SoC. If your linked file is updated then the Project Manager indicates that the file has been changed and asks you if you wish to recompile, as appropriate.

Linked files appear in your Files window (**View > Windows > Files**), where they can be opened, deleted from the project, updated, or unlinked and copied to your local project.

Once you import or generate a Timing Constraint file, you can double-click the file in the **Design Flow** window (**Create Constraints > Timing Constraints > <filename>**) to open it in the Text Editor, right-click the file to:

Use for Synthesis - Uses the file for synthesis.

Use for Compile - Includes the file during Compile.

Open in Text Editor - Opens the file in the Project Manager Text Editor.

Save as - Opens the Save As dialog box, enables you to save the constraint in a different location and/or filename. This is useful if you want to preserve the settings of a particular constraint, or to save it outside your project.

Delete from Project - Removes the file from the project.

Delete from Disk and Project - Removes the file from the project and deletes it from the disk.

Floorplan Constraints - SmartFusion2

SmartFusion2 Floorplan constraints are PDC files. Note that for SmartFusion2 Floorplan constraint PDC files are separate from I/O constraint PDC files; if you have a PDC file that contains both Floorplan and I/O constraints then Libero SoC errors out with an invalid constraint error.

Floorplan Constraints enables you to:

Import Files - Double-click Floorplan Constraints to open the Import Files dialog box and import Floorplan constraint files (*.pdc files).

Link Files - Right-click **I/O Constraints** and choose **Link Files** to link PDC constraint files from other projects. Linked files are not copied into your local project directory; instead the path is stored in your project, enabling you (or others) to update the file separately from Libero SoC. If your linked file is updated then the Project Manager indicates that the file has been changed and asks you if you wish to recompile, as appropriate.

Linked files appear in your **Files window** (**View > Windows > Files**), where they can be opened, deleted from the project, updated, or unlinked and copied to your local project.

Once you import Floorplan Constraint file you can double-click the file in the **Design Flow** window (**Create Constraints > Floorplan Constraints > <filename>**) to open it in the Text Editor, or right-click the file to:

Use for Compile - Includes the constraint file when you run Compile.

Open in Text Editor - Opens the file in the Project Manager Text Editor.

Save as - Opens the Save As dialog box, enables you to save the constraint in a different location and/or filename. This is useful if you want to preserve the settings of a particular constraint, or to save it outside your project.

Delete from Project - Removes the file from the project.

Delete from Disk and Project - Removes the file from the project and deletes it from the disk.

Constrain Design - Import I/O Constraints and Import Timing Constraints

Import I/O Constraints and Import Timing Constraints opens the Import Files dialog box to import PDC or SDC files, respectively.

Right-click **Import I/O Constraints** and choose **Import Files** to open the Import Files dialog box and import PDC files.

Right-click **Import Timing Constraints** and choose **Import Files** to open the Import Files dialog box and import SDC files.

I/O Constraints (PDC Files)

The software enables you to specify the physical constraints to define the size, shape, utilization, and pin/pad placement of a design. You can specify these constraints based on the utilization, aspect ratio, and dimensions of the die. The pin/pad placement depends on the external physical environment of the design, such as the placement of the device on the board.

Timing Constraints (SDC Files)

Timing constraints represent the performance goals for your designs. Software uses timing constraints to guide the timing-driven optimization tools in order to meet these goals.

You can set timing constraints either globally or to a specific set of paths in your design.

You can apply timing constraints to:

- Specify the required minimum speed of a clock domain
- Set the input and output port timing information
- Define the maximum delay for a specific path
- Identify paths that are considered false and excluded from the analysis
- Identify paths that require more than one clock cycle to propagate the data
- Provide the external load at a specific port

To get the most effective results you need to set the timing constraints close to your design goals. Sometimes slightly tightening the timing constraint helps the optimization process to meet the original specifications.

Synthesize

Double-click **Synthesize** to run synthesis on your design automatically; automatic synthesis uses the default settings in your synthesis tool.

If you wish to run synthesis manually, right-click **Synthesize** and choose **Open Interactively** to open your synthesis tool.

The default synthesis tool included with Libero SoC is Synplify Pro ME. If you wish to use a different synthesis tool you can change the settings in your Tool Profile.

Libero SoC works with the following synthesis tools:

- Synplify Pro ME from Synopsys
- Precision RTL from Mentor Graphics

While Precision RTL is not part of the Libero SoC package, they can be integrated to work with Libero SoC. You can also integrate different versions of Synplify. To integrate tools, add them to your project profile.

Some families enable you to set or change Configuration options for your synthesis tool from the Design Flow window. To do so, in the Design Flow window expand **Implement Design**, right-click **Synthesize** and choose **Configure Options**. This opens the Synthesize Options dialog box.

Verilog Standard - Sets your Verilog Standard to Verilog 2001 and/or System Verilog; make your selection based on your design preferences.

VHDL Standard - Sets your VHDL Standard to VHDL 2008; make your selection based on your design preferences.

Synplify Pro ME

Synplify Pro ME is the default synthesis tool for Libero SoC.

To run synthesis using Synplify Pro ME and default settings, right-click **Synthesize** and choose **Run**.

If you wish to use custom settings you must run synthesis interactively.

To run synthesis using Synplify Pro ME with custom settings:

1. If you have set Synplify as your default synthesis tool, right-click **Synthesize** in the Libero SoC Design Flow and choose **Open Interactively**. Synplify starts and loads the appropriate design files, with a few pre-set default values.
2. From Synplify's **Project** menu, choose **Implementation Options**.
3. Set your specifications and click **OK**.
4. Deactivate synthesis of the defparam statement. The defparam statement is only for simulation tools and is not intended for synthesis. Embed the defparam statement in between **translate_on** and **translate_off** synthesis directives as follows :

```
/* synthesis translate_off */
defparam M0.MEMORYFILE = "meminit.dat"

/*synthesis translate_on */
// rest of the code for synthesis
```

5. Click the **RUN** button. Synplify compiles and synthesizes the design into an EDIF, *.edn, file. Your EDIF netlist is then automatically translated by the software into an HDL netlist. The resulting *edn and *.vhd files are visible in the Files list, under Synthesis Files.

Should any errors appear after you click the Run button, you can edit the file using the Synplify editor. Double-click the file name in the Synplify window showing the loaded design files. Any changes you make are saved to your original design file in your project.

6. From the **File** menu, choose **Exit** to close Synplify. A dialog box asks you if you would like to save any settings that you have made while in Synplify. Click **Yes**.

Note: Note: See the Microsemi Attribute and Directive Summary in the Synplify online help for a list of attributes related to Microsemi devices.

Note: To add a clock constraint in Synplify you must add "n:<net_name>" in your SDC file. If you put the net_name only, it does not work.

Precision RTL

Libero SoC supports Precision RTL from Mentor Graphics.

To run synthesis with Precision RTL default settings, set Precision RTL as the synthesis tool for your project (as outlined below), right-click **Synthesize** and choose **Run**.

To run synthesis with custom settings, right-click **Synthesize** and choose **Open Interactively**. Precision RTL opens and enables you to change settings before you run synthesis.

If your design is not ready for synthesis then Open does not appear in your right-click menu.

To set Precision RTL as the synthesis tool for your project:

1. From the **Project** menu, choose **Tool Profiles**. The Tool Profiles dialog box appears.
2. Click **Synthesis** to choose the synthesis tool profile.
3. Click the **Add** button. The Add Profile dialog box appears.
4. Enter a name. This is the name that appears in the Tool Profile dialog box.
5. In the **Tool integration** dropdown menu choose **Precision RTL**.
6. Enter the location of Precision RTL and any additional parameters.
7. Click **OK**.
8. Select **Precision RTL** in the Tool Profile dialog box and click **OK**.
9. Double-click **Synthesize** in the Design Flow window to start Precision RTL and run synthesis.

Instrument Design with the Identify Debugger

Libero SoC integrates the Identify RTL debugger tool. It enables you to probe and debug your FPGA design directly in the source RTL. Use Identify software when the design behavior after programming is not in accordance with the simulation results.

To open the Identify RTL debugger, in the Design Flow window under Debug Design double-click **Instrument Design**.

Identify features:

- Instrument and debug your FPGA directly from RTL source code .
- Internal design visibility at full speed.
- Incremental iteration - Design changes are made to the device from the Identify environment using incremental compile. You iterate in a fraction of the time it takes route the entire device.
- Debug and display results - You gather only the data you need using unique and complex triggering mechanisms.

You must have both the Identify RTL Debugger and the Identify Instrumentor to run the debugging flow outlined below.

To use the Identify debugger:

1. Create your source file (as usual) and run pre-synthesis simulation.
2. (Optional) Run through an entire flow (Synthesis - Compile - Place and Route - Generate a Programming File) without starting Identify.
3. In Synplify, click **Options > Configure Identify Launch** to setup Identify.
4. Right-click **Synthesize** and choose **Open Interactively** in the Libero SoC to launch Synplify. In Synplify, create an Identify implementation; to do so, click **Project > New Identify Implementation**.
5. In the Implementations Options dialog, make sure the Implementation Results > Results Directory points to a location under <libero project>\synthesis\, otherwise Libero SoC is unable to detect your resulting EDN file
6. From the Instumentor UI specify the sample clock, the breakpoints, and other signals to probe. Synplify creates a new synthesis implementation. Synthesize the design.
7. In Libero SoC, select the edif netlist of the Identify implementation you want to use in the flow. Right-click **Compile** and choose **Organize Input Files > Organize Source Files** and select the edif netlist of your Identify implementation.
8. Run Compile, Place and Route and Generate a Programming File with the edif netlist you created with the Identify implementation.
9. Double-click **Instrument Design** in the Design Flow window to launch the Identify Debugger.

The Identify RTL Debugger, Synplify, and FlashPro must be synchronized in order to work properly. See the Release Notes for more information on which versions of the tools work together.

Verify Post-Synthesis Implementation - Simulate

The steps for performing functional and timing simulation are nearly identical. Functional simulation is performed before place-and-route and simulates only the functionality of the logic in the design. Timing simulation is performed after the design has gone through place-and-route and uses timing information based on the delays in the placed and routed designs.

Timing simulation includes much more detailed timing information for the targeted device. Timing simulation requires a testbench.

To perform timing simulation:

1. If you have not done so, back-annotate your design and create your testbench.
2. Right-click **Simulate** (in Design Flow window, Implement Design > Verify Post-Synthesis Implementation > Simulate) and choose **Organize Input Files > Organize Source Files** from the right-click menu.

In the Organize Files for Source dialog box, all the stimulus files in the current project appear in the Source Files in the Project list box. Files already associated with the block appear in the Associated Source Files list box.

In most cases you will only have one testbench associated with your block. However, if you want simultaneous association of multiple testbench files for one simulation session, as in the case of PCI cores, add multiple files to the Associated Source Files list.

To add a testbench: Select the testbench you want to associate with the block in the Source Files in the Project list box and click **Add** to add it to the Associated Source Files list.

To remove a testbench: To remove or change the file(s) in the Associated Source Files list box, select the file(s) and click **Remove**.

To order testbenches: Use the up and down arrows to define the order you want the testbenches compiled. The top level-entity should be at the bottom of the list.

3. When you are satisfied with the Associated Simulation Files list, click **OK**.
4. To start ModelSim AE, right-click **Simulate** in the Design Hierarchy window and choose **Open Interactively**. ModelSim starts and compiles the appropriate source files. When the compilation completes, the simulator runs for 1 µs and the Wave window opens to display the simulation results.
5. Scroll in the Wave window to verify the logic works as intended. Use the cursor and zoom buttons to zoom in and out and measure timing delays. If you did not create a testbench with WaveFormer Pro, you may get error messages with the vsim command if the instance names of your testbench do not follow the same conventions as WaveFormer Pro. Ignore the error message and type the correct vsim command.
6. When you are done, from the **File** menu, choose **Quit**.

Compile

After you import your netlist files and select your device, you must compile your design. Compile contains a variety of functions that perform legality checking and basic netlist optimization. Compile checks for netlist errors (bad connections and fan-out problems), removes unused logic (gobbling), and combines functions to reduce logic count and improve performance. Compile also verifies that the design fits into the selected device.

To compile your device with default settings, right-click **Compile** in the Design Flow window and choose **Run**.

To compile your design with custom settings, right-click **Compile** in the Design Flow window and choose **Configure Options**. You can merge your PDC or SDC files with existing physical or timing constraints, respectively; see the Configure Options in Compile description below for more information.

During compile, the Log window displays information about your design, including warnings and errors. Libero SoC issues warnings when your design violates recommended Microsemi design rules. Microsemi recommends that you address all warnings, if possible, by modifying your design before you continue.

If the design fails to compile due to errors in your input files (netlist, constraints, etc.), you must modify the design to remove the errors. You must then re-import and re-compile the files.

Configure Options in Compile

Right-click **Compile** in the Design Flow window and choose **Configure Options** to view compile options.

Merge User SDC file(s) with Existing Timing Constraints

Select **Merge User SDC file(s) with existing timing constraints**. to preserve all existing timing constraints that you have made using the Timer GUI or previously imported file. If you import a SDC file and you have this checkbox selected, the software merges the existing constraints and the constraints existing in the SDC file. In case of a conflict, the new constraint has priority over the existing constraint.

The Merge SDC file(s) with existing timing constraints option is **On** by default. With this option **On**, your timing constraints from the imported SDC files are merged with the existing constraints. When this option is **Off**, all the existing timing constraints are replaced by the constraints in the newly imported SDC files.

Merge PDC file(s) with Existing Physical Constraints

Select **Merge PDC file(s) with existing physical constraints** to preserve all existing physical constraints that you have entered either using one of the MVN tools (ChipPlanner, PinEditor, or the I/O Attribute Editor) or a previous GCF or PDC file. The software will resolve any conflicts between new and existing physical constraints and display the appropriate message.

The Merge PDC file(s) with existing physical constraints option is Off by default. When this option is Off, all the physical constraints in the newly imported GCF or PDC files are used. All pre-existing constraints are lost. When this option is On, the physical constraints from the newly imported GCF or PDC files are merged with the existing constraints.

Abort Compile if errors are found in the physical design constraints

Changes the Abort on PDC error behavior. Select this option to stop the flow if any error is reported in reading your PDC file. If you deselect this option, the tool skips errors in reading your PDC file and just reports them as warnings. The default is ON.

Note: Note: The flow always stops even if this option is deselected in the following two cases:

- If there is a Tcl error (for example, the command does not exist or the syntax of the command is incorrect)
- The assign_local_clock command for assigning nets to LocalClocks fails. This may happen if any floor planning DRC check fails, such as, region resource check, fix macro check (one of the load on the net is outside the local clock region). If such an error occurs, then the Compile command fails. Correct your PDC file to proceed.

Note: Note: Every time you invoke this dialog box, this option is reset to its default value ON. This is to ensure that your PDC file is correct.

Reserve Live Probe

Specifies if the pins need to be preserved for Live Probe. Reserve your pins for probing if you intend to debug using SmartDebug.

Compile Report

Limit the number of displayed high fanout nets to: Enables flip-flop net sections in the compile report and defines the number of nets to be displayed in the high fanout. The default value is 10.

Compile Options

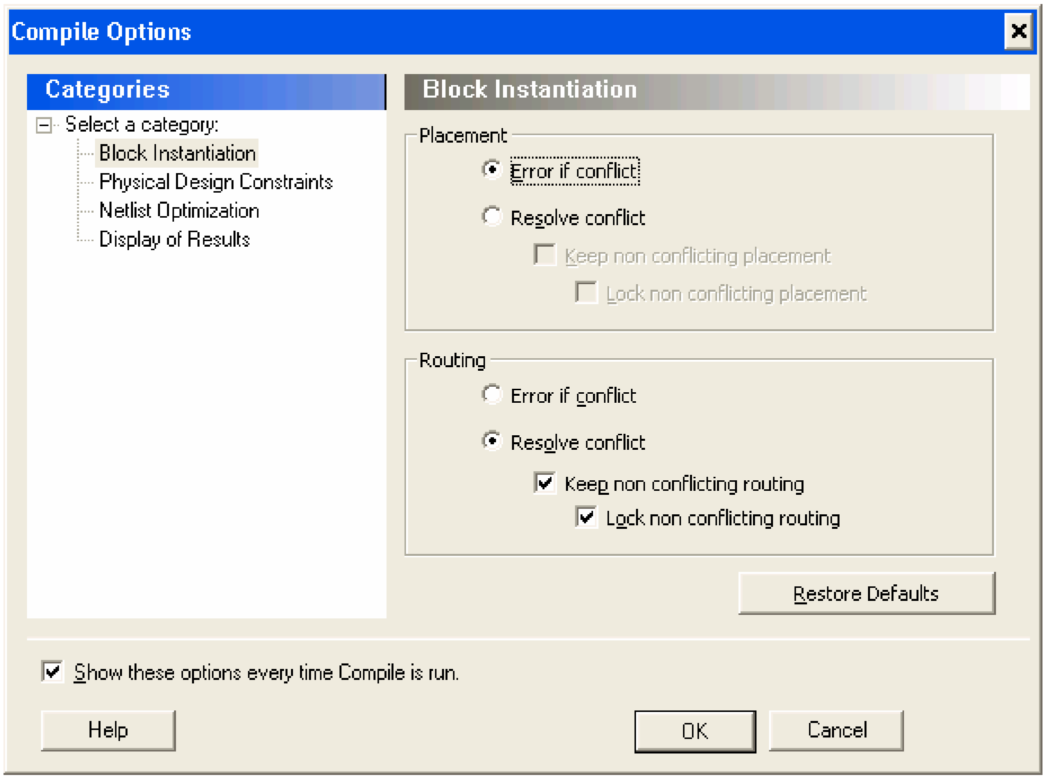
To set custom compile options:

1. Right-click **Compile** and choose **Open Interactively**. Designer opens.
2. Click the **Compile** button. The Compile Options dialog box opens. The Options available are family specific.
3. Select your options, and click **OK**.

The Compile Options dialog box enables you to do the following:

- Set your Block Instantiation options (used for conflict resolution when you instantiate multiple blocks)
- Verify Physical Design Constraints
- Perform Globals Management
- Netlist Optimization
- Generate a Compile report in Display of Results
- Set Block Creation options (available only if you are creating a block)

Block Instantiation



Placement
- ● Error if conflict
- ○ Resolve conflict
  - □ Keep non conflicting placement
    - □ Lock non conflicting placement

Routing
- ○ Error if conflict
- ● Resolve conflict
  - ☑ Keep non conflicting routing
    - ☑ Lock non conflicting routing

☑ Show these options every time Compile is run.

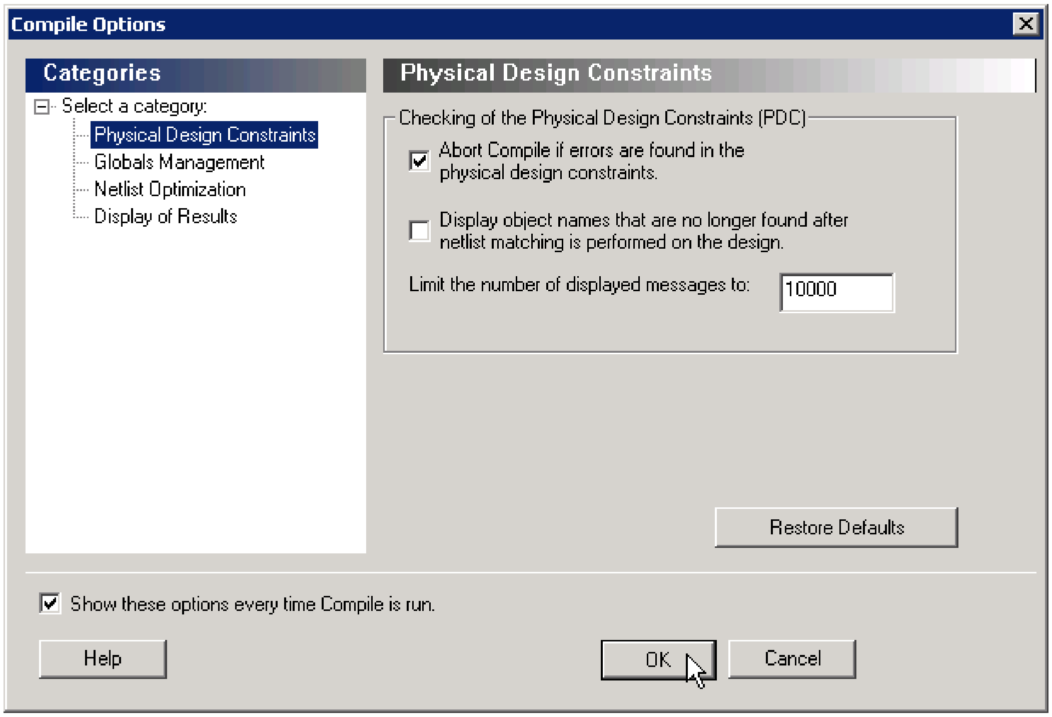
Designer uses the Block Instantiation options to resolve conflicts between multiple blocks in your design. The default options is to return an error if there is overlapping placement between the blocks and resolve any conflict for nets.

This ensures you are aware that the blocks overlap; you can go back and set the placement to resolve the conflicts and it will Compile.

See Conflict resolution in Designer Blocks for more information.

Physical Design Constraints

This interface enables you to verify the Physical Design Constraints (PDC) file.



Checking of the Physical Design Constraints (PDC)

☑ Abort Compile if errors are found in the physical design constraints.

□ Display object names that are no longer found after netlist matching is performed on the design.

Limit the number of displayed messages to: 10000

☑ Show these options every time Compile is run.

Figure 37 ·

Checking the Physical Design Constraint (PDC)

Abort Compile if errors are found in the physical design constraints: Changes the Abort on PDC error behavior. Select this option to stop the flow if any error is reported in reading your PDC file. If you deselect this option, the tool skips errors in reading your PDC file and just reports them as warnings. The default is ON.

Note: Note: The flow always stops even if this option is deselected in the following two cases:

- If there is a Tcl error (for example, the command does not exist or the syntax of the command is incorrect)
- The assign_local_clock command for assigning nets to LocalClocks fails. This may happen if any floor planning DRC check fails, such as, region resource check, fix macro check (one of the load on the net is outside the local clock region). If such an error occurs, then the Compile command fails. Correct your PDC file to proceed.

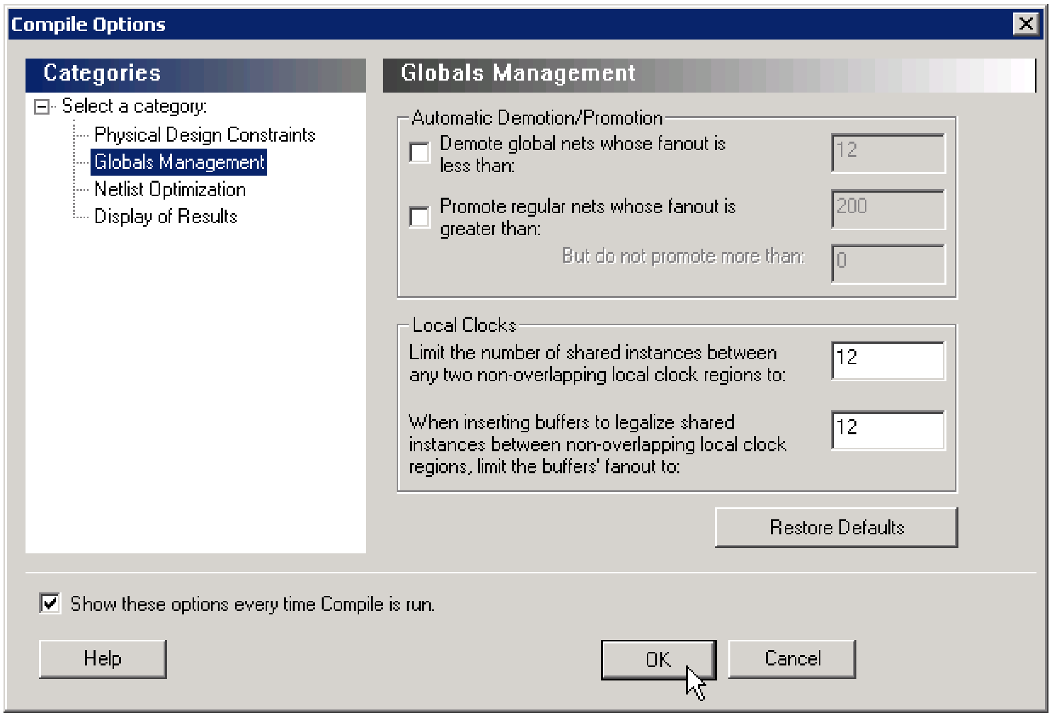
Note: Note: Every time you invoke this dialog box, this option is reset to its default value ON. This is to ensure that your PDC file is correct.

Display object names that are no longer found after netlist matching is performed on the design: Displays netlist objects in the PDC that are not found in the imported netlist during the Compile ECO mode. Select this option to report netlist objects not found in the current netlist when reading the internal ECO PDC constraints. The default is OFF.

Limit the number of displayed messages to: Defines the maximum number of errors/warnings to be displayed in the case of reading ECO constraints. The default is 10000 messages.

Globals Management

The interface provides a global control to the Compile component of the design flow.



Automatic Demotion/Promotion

□ Demote global nets whose fanout is less than: 12

□ Promote regular nets whose fanout is greater than: 200

But do not promote more than: 0

Local Clocks

Limit the number of shared instances between any two non-overlapping local clock regions to: 12

When inserting buffers to legalize shared instances between non-overlapping local clock regions, limit the buffers' fanout to: 12

☑ Show these options every time Compile is run.

Automatic Demotion/Promotion

Demote global nets whose fanout is less than: Enables the global clock demotion of global nets to regular nets.
By default, this option is OFF. The maximum fanout of a demoted net is 12.

Note: Note: A global net is not automatically demoted (assuming the option is selected) if the resulting fanout of the demoted net is greater than the max fanout value. Microsemi recommends that the automatic global demotion only act on small fanout nets. Microsemi recommends that you drive high fanout nets with a clock network in the design to improve timing and routability.

Promote regular nets whose fanout is greater than: Enables global clock promotion of nets to global clock network. By default, this option is OFF. The minimum fanout of a promoted net is 200.

But do not promote more than: Defines the maximum number of nets to be automatically promoted to global. The default value is 0. This is not the total number as nets need to satisfy the minimum fanout constraint to be promoted. The promote_globals_max_limit value does not include globals that may have come from either the netlist or PDC file (quadrant clock assignment or global promotion).

Note: Note: Demotion of globals through PDC or Compile is done before automatic global promotion is done.

Note: You may exceed the number of globals present in the device if you have nets already assigned to globals or quadrants from the netlist or by using a PDC file. The automatic global promotion adds globals on what already exists in the design.

Local clocks

Limit the number of shared instances between any two non-overlapping local clock regions to: Defines the maximum number of shared instances allowed to perform the legalization. It is also for quadrant clocks.

The maximum number of instances allowed to be shared by 2 local clock nets assigned to disjoint regions to perform the legalization (default is 12, range is 0-1000). If the number of shared instances is set to 0, no legalization is performed.

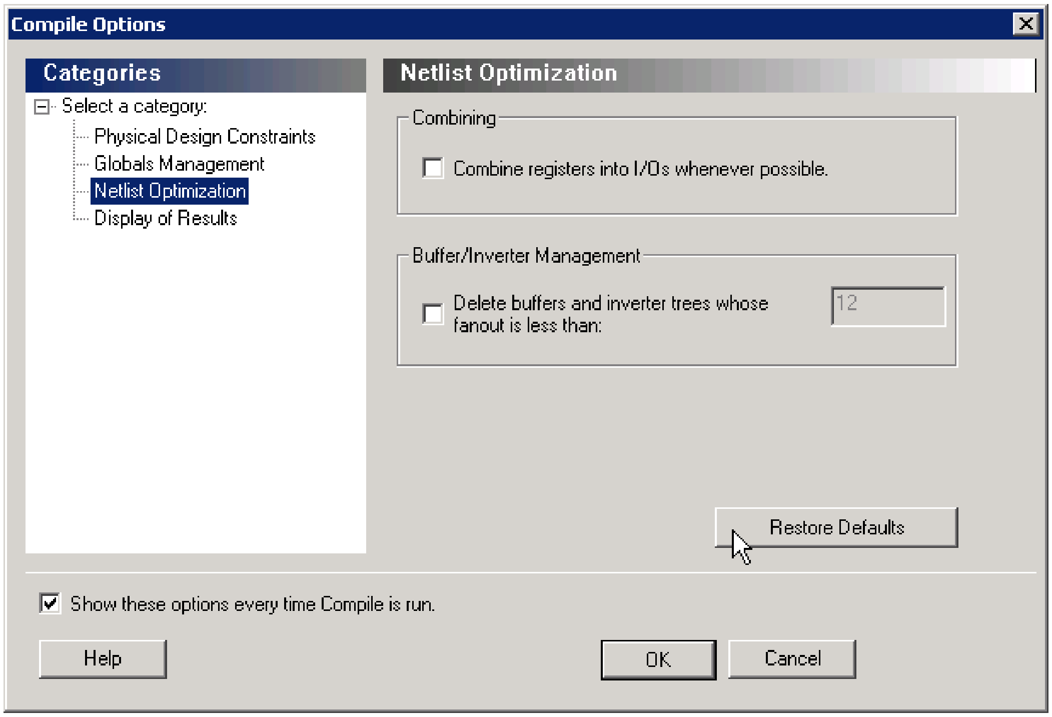
When inserting buffers to legalize shared instances between non-overlapping local clock regions, limit the buffers' fanout to: Defines the maximum fanout value used during buffer insertion for clock legalization. Set the value to 0 to disable this option and prevent legalization (default value is 12, range is 0-

1000). If the value is set to 0, no buffer insertion is performed. If the value is set to 1, there will be one buffer inserted per pin.

Note: Note: If you assign quadrant clock to nets using MultiView Navigator, no legalization is performed.

Netlist Optimization

This interface allows you to perform netlist optimization.



Compile Options

Categories

- Select a category:
  - Physical Design Constraints
  - Globals Management
  - Netlist Optimization
  - Display of Results

Netlist Optimization

Combining

□ Combine registers into I/Os whenever possible.

Buffer/Inverter Management

□ Delete buffers and inverter trees whose fanout is less than: 12

☑ Show these options every time Compile is run.

Combining

Combine registers into I/O wherever possible: Combines registers at the I/O into I/O-Registers. Select this option for optimization to take effect. By default, this option is OFF.

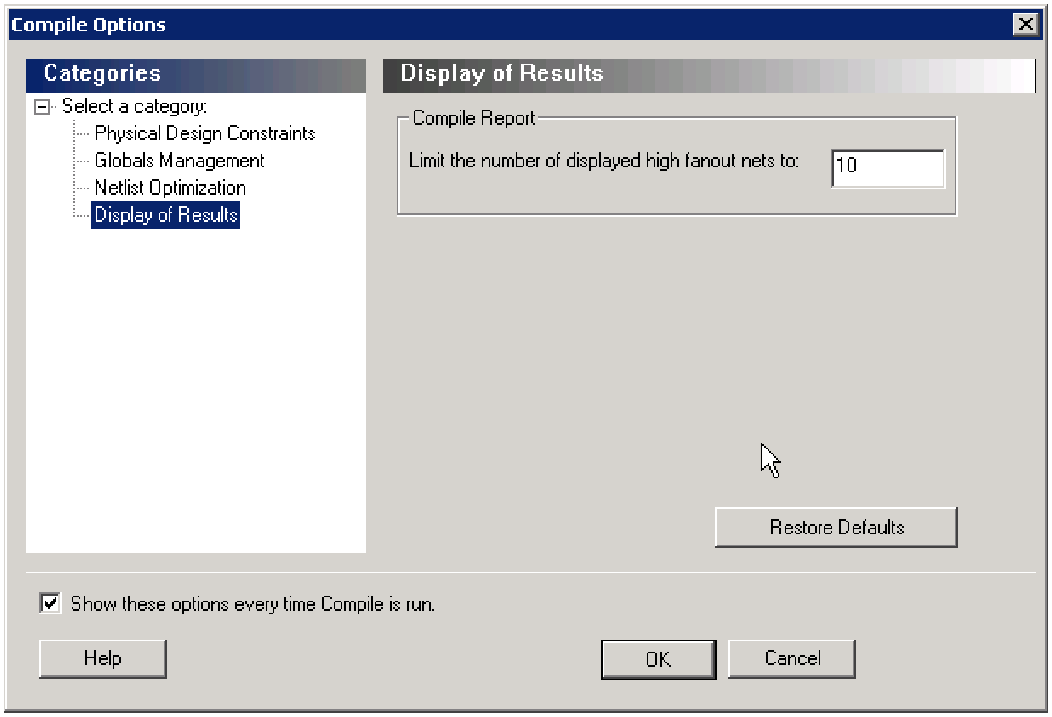
Buffer/Inverter Management

Delete buffers and inverter trees whose fanout is less than: Enables buffer tree deletion on the global signals from the netlist. The buffer and inverter are deleted. By default, this option is OFF. The maximum fanout of a net after buffer tree deletion is 12.

Note: Note: A net does not automatically remove its buffer tree (assuming the option is on) if the resulting fanout of the net (if the buffer tree was removed) is greater than the max fanout value. Microsemi recomends that the automatic buffer tree deletion should only act on small fanout nets. From a routability and timing point of view, it is not recommended to have high fanout nets not driven by a clock network in the design.

Display of Results

This interface lets you generate a Compile report.



Compile Options

Categories

- Select a category:
  - Physical Design Constraints
  - Globals Management
  - Netlist Optimization
  - Display of Results

Display of Results

Compile Report

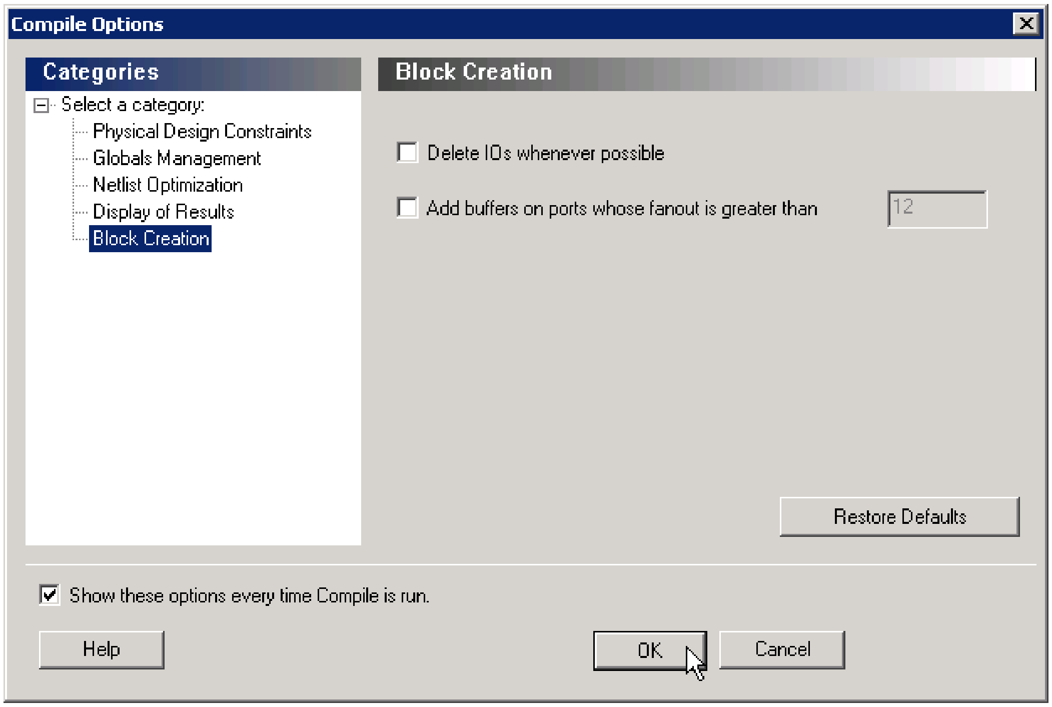
Limit the number of displayed high fanout nets to: 10

☑ Show these options every time Compile is run.

Compile Report

Limit the number of displayed high fanout nets to: Enables flip-flop net sections in the compile report and defines the number of nets to be displayed in the high fanout. The default value is 10.

Block Creation (Available only when creating Designer Blocks)



Compile Options

Categories

- Select a category:
  - Physical Design Constraints
  - Globals Management
  - Netlist Optimization
  - Display of Results
  - Block Creation

Block Creation

☐ Delete IOs whenever possible

☐ Add buffers on ports whose fanout is greater than 12

☑ Show these options every time Compile is run.

Delete I/Os whenever possible - Deletes I/Os in the block during compile (except TRIBUFF and BIBUFF, because they cannot be removed). Useful if you have I/Os in your design but want to create a block anyway.

Add buffers on ports whose fanout is greater than <value> - Adds buffers on ports with a fanout greater than a value you specify. This option enables more predictable block timing. For example, if you have a net with a fanout of 100 the net will be unrouted. If you add a buffer, the output of the buffer is routed and the routing is preserved.

See Also

`compile`

Configure Flash*Freeze

Opens the Flash*Freeze Hardware Settings dialog box. For more information on the Flash*Freeze mode for SmartFusion2 see the SmartFusion2 Low Power User's Guide.

The fabric SRAMs can be put into a Suspend Mode or a Sleep Mode. This applies to both the Large SRAM (LSRAM) instances of RAM1xK18 and the Micro SRAM (uSRAM) instances of RAM64x18. These SRAMs are grouped in rows in Libero® System-on-Chio (SoC) devices and each SRAM row can be configured independently to go into Suspend or Sleep mode during Flash*Freeze mode:

- In Suspend mode: LSRAM and uSRAM contents are retained.
- In Sleep mode: LSRAM and uSRAM contents are not retained.

The SRAM state during Flash*Freeze mode must be configured in a PDC file and can be done on a per row basis. The syntax for configuring SRAM in Sleep or Suspend mode is:

```
set_ram_ff -mode [suspend |sleep] -row [Row Number]
```

uRAM/LSRAM State

Sleep - Sets to Sleep.

Suspend - Sets to Suspend.

MSS Clock Source

The lower the frequency the lower the power will be. But for some peripherals that can remain active (such as SPI or MMUART), you may need a higher MSS clock frequency (such as to meet the baud rate for MMUART).

Options are:

- On-Chip 1 MHz RC Oscillator
- On-Chip 50 MHz RC Oscillator
- External 32 KHz Crystal Oscillator

Place and Route

Place and Route runs automatically with default settings as part of the push-button design flow in Libero SoC.

Custom Layout options are saved when you save your ADB after place and route.

To change your Place and Route settings:

Expand **Implement Design**, right-click **Place and Route** and choose **Configure Options**.

Place and Route Options

Timing-Driven

Select this option to run Timing-Driven Layout. The primary goal of timing-driven layout is to meet timing constraints, specified by you or generated automatically. Timing-driven Layout typically delivers better performance than Standard layout.

If you do not select Timing-driven layout, Designer runs Standard layout. Standard layout targets efficient usage of the chip resources. Chip performance is not optimized. Timing constraints are not considered by the Layout in standard mode, although a delay report based on delay constraints entered in SmartTime can still be generated for the design. This is helpful to determine if timing-driven Layout is required.

If your design has multiple scenarios, you can select a scenario from the pull-down list to perform timing-driven layout.

Place and Route Incrementally

Select this option to use previous placement data as the initial placement for next placement run. Additionally, this will preserve previous placement data during the next incremental placement run.

Router will also be run incrementally. Select to fully route a design when some nets failed to route during a previous run. You can also use it when the incoming netlist has undergone an ECO. (Engineering Change Order). Incremental routing should only be used if a low number of nets fail to route (less than 50 open nets or shorted segments). A high number of failures usually indicates a less than optimal placement (if using manual placement through macros, for example) or a design that is highly connected and does not fit in the device. If a high number of nets fail, relax constraints, remove tight placement constraints, deactivate timing-driven mode, or select a bigger device and rerun Layout. Also, see the Advanced Layout options for your device.

Lock Existing Placement (Fix)

Locks your existing placement. Use this option if you do not want any changes in your layout.

Additional Layout Options Available if you Open Interactively

The I/O Bank Assigner and Global Planner run automatically after you click **OK** in the **Layout Options** dialog box. The I/O Bank Assigner automatically assigns technologies to all I/O banks that have not been assigned a technology. The Global Planner automatically assigns global nets to clock conditioning circuit (CCC) locations on the chip in the design.

Note: Note: All I/O technologies assigned to I/O banks by the I/O Bank Assigner in Layout are unlocked.

Power-Driven

Select this option to run Power-Driven Layout. The primary goal of power-driven layout is to reduce dynamic power while still maintaining timing constraints.

To get the most out of Power-Driven Layout:

1. Enter maximum delay, minimum delay, setup, and hold constraints in SmartTime's constraint editor or in SDC.
2. Set false paths on any paths that have a constraint, but do not need one (this will help layout meet the constraints that are needed).
3. Perform Layout with **Timing-Driven**, **Run Place**, and **Run Route** options checked.
4. Resolve worst case setup and maximum delay violations.
5. Generate an SDF back-annotation file.
6. Perform a post layout back-annotated simulation using this SDF file, and export a VCD (Value Change Dump) file that will capture real activities for each net.
7. Import this VCD file in Designer using the **Import Auxiliary** option from the **File** menu.
8. Perform Layout with **Timing-Driven** and **Power-Driven** checked. Run Place and Route.
9. Verify that your timing constraints are still met with SmartTime.
10. Analyze your power with SmartPower.

In case you do not have simulation vectors for your design, the following alternative flow is recommended:

1. Enter maximum delay, minimum delay, setup, and hold constraints in SmartTime's constraint editor or in SDC.
2. Set false paths on any paths that have a constraint, but do not need one (this will help layout to meet the constraints that are needed).
3. Perform Layout with **Timing-Driven**, **Run Place**, and **Run Route** options checked.
4. Resolve worst case setup and maximum delay violations.
5. Verify that your timing constraints are still met with SmartTime.
6. Open SmartPower and set clock frequencies and toggle rates for the different clocks. Clock frequencies can be imported from your timing constraints. Refer to Initialize Frequencies for more information.
7. Perform Layout with **Timing-Driven**, and **Power-Driven** options checked. Run Place and Route.
8. Verify that your timing constraints are still met with SmartTime.
9. Analyze your power with SmartPower

Run Place

Select this option to run the placer during Layout. By default, it reflects the current Layout state. If you have not run Layout before, Run Place is selected by default. If your design has already been placed but not routed, this box is cleared by default. You can also select the following incremental placement options.

- **Incrementally**: Select to use previous placement data as the initial placement for the next place run.
- **Lock Existing Placement (fix):** Select to preserve previous placement data during the next incremental placement run.

Incremental options apply to the entire design. For more detailed control of the placer behavior (such as, to fix placement of a portion of the design), use the MultiView Navigator tools or set fixed attributes on the placed instances via PDC constraint files.

Run Route

Select to run the router during Layout. By default, it reflects the current Layout state. If you have not run Layout before, Run Route is checked. Run Route is also checked if your previous Layout run completed with routing failures. If your design has been routed successfully, this check box is cleared.

- **Incrementally**: Select to fully route a design when some nets failed to route during a previous run. You can also use it when the incoming netlist has undergone an ECO. (Engineering Change Order). Incremental routing should only be used if a low number of nets fail to route (less than 50 open nets or shorted segments). A high number of failures usually indicates a less than optimal placement (if using manual placement through macros, for example) or a design that is highly connected and does not fit in the device. If a high number of nets fail, relax constraints, remove tight placement constraints, deactivate timing-driven mode, or select a bigger device and rerun Layout. Also, see the Advanced Layout options for your device.

There is no "Fix" option for the router. In incremental mode the router tries to preserve the existing routing; there is no guarantee that it will be preserved. Therefore the timing characteristics of the previously routed portion of the design may change, even if the placement was fixed for that portion of the design. The chance of this is quite small, and the router will print the list of nets that have fixed terminals (i.e. those nets whose every pin's macro has the placement FIX attribute).

Use Multiple Passes

Select to run layout multiple times with different seeds. Multiple Pass Layout attempts to improve layout quality by selecting from a greater number of layout results. Click **Configure** to set your Multiple Pass Configuration.

Click the Advanced button to set Timing-Driven options.

Place and Route - SmartFusion2

Timing-Driven

Select this option to run Timing-Driven Place and Route. The primary goal of timing-driven Place and Route is to meet timing constraints, specified by you or generated automatically. Timing-driven Place and Route typically delivers better performance than Standard.

If you do not select Timing-driven Place and Route, software uses default settings. The default Place and Route targets efficient usage of the chip resources. Chip performance is not optimized. Timing constraints are not considered by the software in standard mode, although a delay report based on delay constraints entered in SmartTime can still be generated for the design. This is helpful to determine if timing-driven Place and Route is required.

If your design has multiple scenarios, you can select a scenario from the pull-down list to perform timing driven Place and Route.

Power-Driven

Select this option to run Power-Driven Layout. The primary goal of power-driven layout is to reduce dynamic power while still maintaining timing constraints.

To get the most out of Power-Driven Layout:

1. Enter maximum delay, minimum delay, setup, and hold constraints in SmartTime constraint editor or in the SDC file.
2. Set false paths on any paths that have a constraint, but do not need one (this will help layout meet the constraints that are needed).
3. Perform Place and Route with **Timing-Driven** checked.
4. Resolve worst case setup and maximum delay violations.
5. Generate an SDF back-annotation file.
6. Perform a post layout back-annotated simulation using this SDF file, and export a VCD (Value Change Dump) file that will capture real activities for each net.
7. Import this VCD file in Designer using the **Import Auxiliary** option from the **File** menu.
8. Perform Place and Route with **Timing-Driven** and **Power-Driven** checked. Run Place and Route.
9. Verify that your timing constraints are still met with SmartTime.
10. Analyze your power with SmartPower.

High Effort Layout

This option turns on netlist optimizations to obtain better performance. Layout runtime will increase when this option is selected. You can also combine this option with the Multi-Pass mode to achieve the best possible performance.

In the regular flow the compile step in Designer would modify the netlist to make use of efficient resources on the chip, such as global networks and special macros. When the **High Effort Layout** option is turned on, the placer could further change the mapping of the logic components, preserving the original functionality of the design. The changed netlist is then used in all post-layout Designer tools including back-annotation.

The names and types of the combinational core logic primitives may change. All other logic cells (such as registers, memory, I/Os or clocks) or combinational logic primitives that are assigned a physical constraint (locked at a location, assigned to a region, or part of a block component), referred in a timing constraint, or have a preserve property, will remain unchanged.

Incremental Layout

Choose Incremental Layout to use previous placement data as the initial placement for the next run. Note that you should use Incremental Layout only if a low number of nets fail to route (less than 50 open nets or shorted segments). A high number of failures usually indicates a less than optimal placement (if using manual placement through macros, for example) or a design that is highly connected and does not fit in the device. If a high number of nets fail, relax constraints, remove tight placement constraints, deactivate timing-driven mode, or select a bigger device and rerun Place and Route.

SmartFusion, IGLOO, ProASIC3 and Fusion Place and Route Advanced Options

To set these advanced options during Layout, click **Advanced** in the Layout dialog box. The Advanced Layout options are only available in timing-driven Layout mode.

High Effort Layout Mode

This option turns on netlist optimizations to obtain better performance. Layout runtime will increase when this option is selected. You can also combine this option with the Multi-Pass mode to achieve the best possible performance.

In the regular flow the compile step in Designer would modify the netlist to make use of efficient resources on the chip, such as global networks and special macros. When the **High Effort Layout** option is turned on, the placer could further change the mapping of the logic components, preserving the original functionality of the design. The changed netlist is then used in all post-layout Designer tools including back-annotation.

The names and types of the combinational core logic primitives may change. All other logic cells (such as registers, memory, I/Os or clocks) or combinational logic primitives that are assigned a physical constraint (locked at a location, assigned to a region, or part of a block component), referred in a timing constraint, or have a preserve property, will remain unchanged.

When the **Lock Existing Placement** option is also turned on, the placer runs in regular effort mode.

Note: Note: If you change the High Effort Setting you must rerun Place and Route to complete Layout.

Sequential Optimization

Applies to SmartFusion, Fusion, ProASIC3/E/L and IGLOO/E families only.

This option turns on optimization of sequential cells in the High Effort Layout mode. This typically enables register retiming without disturbing timing latency. The names of registers may change unless they are assigned a physical constraint (locked at a location, assigned to a region, or part of a block component), referred in a timing constraint, or have a preserve property. Other restrictions may also apply.

The following cases are excluded from sequential optimization:

- Registers that have any timing constraint other than global FMAX, TSU (setup time) or TCO (clock to out). Registers referred by multi-cycle or exception timing constraints are not moved.
- Registers that feed asynchronous control signals on another register.
- Registers feeding the clock of another register.
- Registers feeding a register in another clock domain.
- Registers that are fed by a register in another clock domain.
- Registers connected to PLL.
- Registers that have PDC attribute “preserve”, assigned a physical constraint (locked at a location, assigned to a region, or part of a block component).

- Both registers in a direct connection from input I/O-to-register-to-register if both registers have the same clock and the first register does not fan out to anywhere else. These registers are considered synchronization registers.
- Both registers in a direct connection from register-to-register if both registers have the same clock, the first register does not fanout anywhere else, and the first register is fed by another register in a different clock domain. These registers are considered synchronization registers.

Router

Repair Minimum Delay Violations

With this option selected, layout will perform an additional route that will attempt to repair paths that have minimum delay and hold time violations. This is done by increasing the length of routing paths and inserting routing buffers to add delay to paths. Since placement will remain unchanged and no additional tiles or modules will be inserted, the amount of delay inserted is limited. As a result, this function is best suited to repair paths with small (0 to 3 ns) hold and minimum delay violations. Paths with large violations will likely improve, but for a complete repair of these paths, manual placement or source code modification may be necessary. Every effort will be made to avoid creating max-delay timing violations on worst case paths.

To get the most out of repair minimum delay violations:

1. Enter max-delay, min-delay, setup and hold constraints in SmartTime's constraint editor or in SDC.
2. Set false paths on any paths that have a constraint, but do not need one (this will help layout to meet the constraints that are needed).
3. Perform Layout with **Timing Driven**, **Run Place**, **Run Route** and optionally **Run incrementally** enabled.
4. Resolve worst case setup and max-delay violations before running minimum delay violations repair.
5. After worst case max-delay timing is resolved, evaluate timing in SmartTime’s Timing Analyzer in minimum delay analysis mode to check for hold time and minimum delay violations.
6. Run repair minimum delay violations with incremental route enabled.
   The repair minimum delay violations tool will attempt to fix all hold time and minimum delay violations by lengthening routing delay paths and inserting routing buffers. As delay is added to paths, worst case max-delay timing is verified to avoid creating new max-delay timing violations. Designer will report the worst minimum slack and the number of violating paths in the log window. In some cases, additional improvement can occur by running repair minimum delay violations multiple times with **Run Incrementally** enabled.
7. Perform both maximum and minimum delay timing analysis to check the timing. Manual placement or source code modification may be necessary to repair all minimum delay violations.
8. After making placement or source code changes, run incremental route and repair minimum delay violations, and then analyze timing again.

Additional Factors

Runtime may vary greatly with the number of paths that need repair, the number of nets in those paths, and the resources available for the tool to insert delay. Over-constraining paths will increase runtime, but will not likely improve results .

The tool will only work on paths that have min delay and hold time constraints. However, other paths that share common nets to the constrained paths may be inadvertently affected.

It is recommended to run minimum delay violations repair with incremental route. This will ensure that paths which do not have minimum delay violations are preserved.

Repair will be performed on:

- Register to register paths where both registers are on the same global or non-global clock
- Register to register paths where the registers are on different clock networks and a minimum delay constraint exists
- Input to register, register to output, clock to out, input to output paths with minimum delay or hold constraint.

You may select programmable input delays to increase delay on input to register paths for devices that support the feature.

Restore Defaults

Click Restore Defaults to run the factory default settings for Advanced options.

Simulate - Opens ModelSim AE

The back-annotation functions are used to extract timing delays from your post layout data. These extracted delays are put into a file to be used by your CAE package's timing simulator. The default simulator for Libero SoC is ModelSim AE. You can change your default simulator in your Tool Profile.

If you wish to perform pre-layout simulation: In the Design Flow Window, under Verify Pre-Synthesized design, double-click Simulate.

To perform timing simulation:

1. If you have not done so, back-annotate your design and create your testbench.
2. Right-click **Simulate** (in Design Flow window, Implement Design > Verify Post-Synthesis Implementation > Simulate) and choose **Organize Input Files > Organize Source Files** from the right-click menu.

In the Organize Files for Source dialog box, all the stimulus files in the current project appear in the Source Files in the Project list box. Files already associated with the block appear in the Associated Source Files list box.

In most cases you will only have one testbench associated with your block. However, if you want simultaneous association of multiple testbench files for one simulation session, as in the case of PCI cores, add multiple files to the Associated Source Files list.

To add a testbench: Select the testbench you want to associate with the block in the Source Files in the Project list box and click **Add** to add it to the Associated Source Files list.

To remove a testbench: To remove or change the file(s) in the Associated Source Files list box, select the file(s) and click **Remove**.

To order testbenches: Use the up and down arrows to define the order you want the testbenches compiled. The top level-entity should be at the bottom of the list.

3. When you are satisfied with the Associated Simulation Files list, click **OK**.
4. To start ModelSim AE, right-click **Simulate** in the Design Hierarchy window and choose **Open Interactively**. ModelSim starts and compiles the appropriate source files. When the compilation completes, the simulator runs for 1 µs and the Wave window opens to display the simulation results.
5. Scroll in the Wave window to verify the logic works as intended. Use the cursor and zoom buttons to zoom in and out and measure timing delays. If you did not create a testbench with WaveFormer Pro, you may get error messages with the vsim command if the instance names of your testbench do not follow the same conventions as WaveFormer Pro. Ignore the error message and type the correct vsim command.
6. When you are done, from the **File** menu, choose **Quit**.

Generate Back Annotated Files - SmartFusion2 Only

Generates Back Annotated (post-layout) files for your design.

Post-layout files include:

- *ba.sdf - Standard Delay Format for back-annotation to the simulator.
- *ba.vhd - AFL flattened netlist used exclusively for back-annotated timing simulation. May contain low level macros not immediately recognizable to you; these were added by the software to improve your design performance.

To generate a post-layout file, in the Design Flow window click **Implement Design** and double-click **Generate Back Annotated Files**.

Right-click **Generate Back Annotated Files** and choose **Configure Options** to open the Generate Back Annotated Files Options dialog box.

Simulator Language Type - Set your simulator language type according to your design.

Timing: Export enhanced min delays for best case - Exports your enhanced min delays to include your best-case timing results in your Back Annotated file.

Export Back Annotated Files

Libero SoC uses post-layout files for back-annotated timing simulation.

Post-layout files include:

- *ba.sdf - Standard Delay Format for back-annotation to the simulator.
- *ba.vhd - AFL flattened netlist used exclusively for back-annotated timing simulation. May contain low level macros not immediately recognizable to you; these were added by the software to improve your design performance.

To generate a post-layout file, in the Design Flow window click **Implement Design** and double-click **Export Back Annotated Files**.

If you wish to export the Back Annotated simulation model with options that are different than the default, right-click **Export Back Annotated Simulation Model** and choose **Open Interactively**.

Generate Fabric Programming Data - SmartFusion2 Only

Generates programming data for your design. This operation is completed automatically as the last step if you use the Build button.

When the process is complete a green check appears next to the operation in the Design Flow window and information messages appear in the Log window (shown in the figure below).

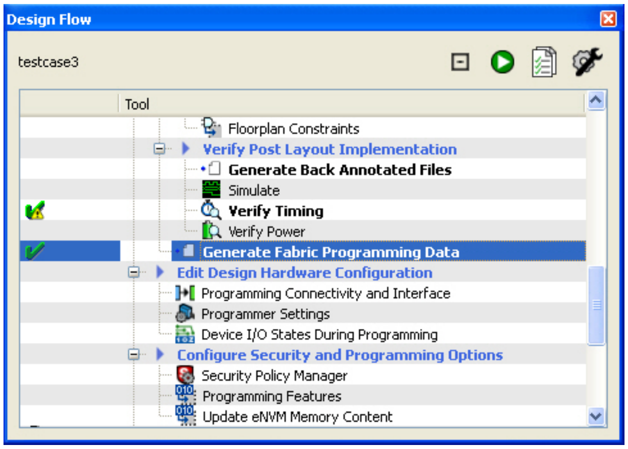

Design Flow
testcase3
Tool
Floorplan Constraints
Verify Post Layout Implementation
Generate Back Annotated Files
Simulate
Verify Timing
Verify Power
Generate Fabric Programming Data
Edit Design Hardware Configuration
Programming Connectivity and Interface
Programmer Settings
Device I/O States During Programming
Configure Security and Programming Options
Security Policy Manager
Programming Features
Update eNVM Memory Content

Figure 38 · Generate Fabric Programming Data (Complete)

Device Programming

Default Programming Data is generated automatically as part of the Libero SoC push-button design flow.

To generate your programming data with custom settings via FlashPoint, expand **Implement Design**, right-click **Generate Programming Data** and choose **Open Interactively**.

FlashPoint enables you to generate other programming files, such as DirectC files (*.dat), IEEE 1532 files (*.bsd, *.isc), programming data files (*.pdb), or Serial Vector Files (*.svf).

You must have completed your design to generate your programming (*.stp or STAPL) file.

SmartFusion, IGLOO, ProASIC3 and Fusion devices use the FlashPoint program file generator to create a programming file. The FlashPoint interface enables the advanced security features in all three device families.

See Also

Generate a DAT file

Programming Connectivity and Interface - SmartFusion2 Only

In the Libero SoC Design Flow window expand **Edit Design Hardware Configuration** and double-click **Programming Connectivity and Interface** to open the Programming Connectivity and Interface window. The Programming Connectivity and Interface window displays the physical chain from TDI to TDO.

The Programming Connectivity and Interface view enables the following actions:

Construct Chain Automatically - Attempts to automatically construct the chain from the physical chain connected to the programmer

Manual Chain Construction Buttons

- **Add Microsemi Device** – Add a Microsemi Device to the chain
- **Add Non-Microsemi Device** – Add a non-Microsemi Device to the chain
- **Add Microsemi Devices From Files** – Add a Microsemi Device from a programming file
- **Delete Selected Device** – Delete selected devices in the grid
- **Scan and Check Chain** – Scan the physical chain connected to the programmer and check if it matches the chain constructed in the grid
- **Zoom In** – Zoom into the grid
- **Zoom Out** – Zoom out of the grid

Hover Information

The device tooltip displays the following information if you hover your pointer over a device in the grid:

- **Device** - Device name
- **Name** - Editable field for a user-specified device name. If you have two or more identical devices in your chain you can use this field to give them unique names.
- **File** - Path to programming file
- **ID** - The unique ID location within the chain
- **IR Length** - Device instruction length
- **Max TCK (MHz**) - Maximum clock frequency to program a specific device; FlashPro uses this information to ensure that the programmer operates at a frequency lower than the slowest device in the chain

Device Chain Details

The device within the chain has the following details:

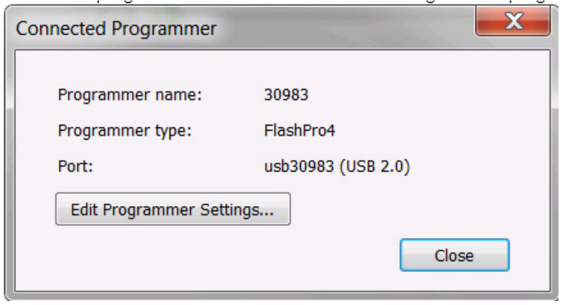
- **Enable Device** - Select to enable the device for programming; enabled devices are green, disabled devices are gray.
- **Name** - Displays your specified device name.
- **File** - Path to programming file.

Right-Click Properties

- **Enable Device** - Select to enable the device for programming; enabled devices are green, disabled devices are gray.
- **HIGH-Z** - Sets disabled Microsemi SoC SmartFusion2 devices in the chain to HIGH-Z (tri-states all the I/Os) during chain programming of enabled Microsemi devices in the daisy chain (Not supported for Libero SoC target design device)
- **Configure Device** – Ability to reconfigure the device (for a Libero SoC target device the dialog appears but only the device name is editable)
- **Load Programming File** – Load programming file for selected device (Not supported for Libero SoC target design device)
- **Enable Serial** - Select to enable serialization when you have loaded a serialization programming file (not supported in software version 11.0)
- **Serial Data** - Opens the Serial Settings dialog box; enables you to set your serialization data.
- **Select Program Procedure/Actions** (Not supported for Libero SoC target design device):
  - **Actions** - List of programming actions for your device.
  - **Procedures** - Advanced option; enables you to customize the list of recommended and optional procedures for the selected Action.

Programmer Settings - SmartFusion2 Only

In the Libero SoC Design Flow window expand **Edit Design Hardware Configuration** and double-click **Programmer Settings** to view the name, type, and port. The dialog box displays information about your programmer if it is connected.



Connected Programmer

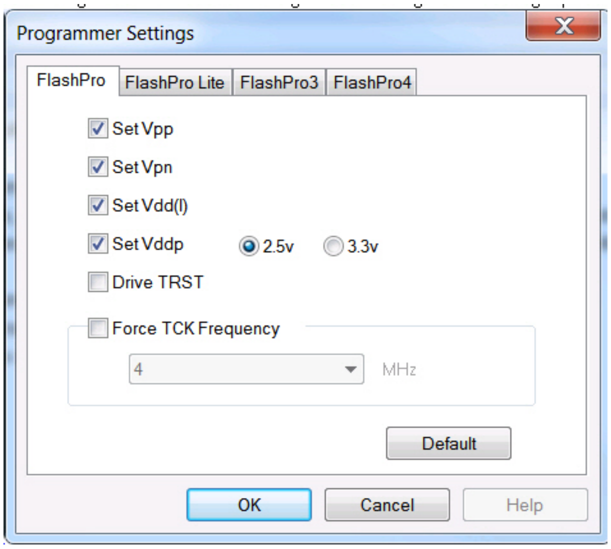
Programmer name: 30983

Programmer type: FlashPro4

Port: usb30983 (USB 2.0)

Figure 39 · Programmer Settings for Connected Programmer

Click Edit Programmer Settings to view the Programmer Settings Dialog box. It enables you to set specific voltage and force TCK frequency values for your programmer.



☑ Set Vpp

☑ Set Vpn

☑ Set Vdd(l)

☑ Set Vddp ● 2.5v ○ 3.3v

□ Drive TRST

□ Force TCK Frequency

4 MHz

Figure 40 · Programmer Settings Dialog Box

The Programmer Settings dialog box includes setting options for FlashPro4/3/3X, FlashPro Lite and FlashPro.

Set the TCK setting in your PDB/STAPL file by selecting the TCK frequency in the Programmer Settings dialog box. TCK frequency limits by programmer:

- FlashPro supports 1-4 MHz
- FlashPro Lite is limited to 1, 2, or 4 MHz only.
- FlashPro4/3/3X supports 1-4 MHz.

TCK frequency limits by target device:

• IGLOO, ProASIC3, Fusion, SmartFusion and SmartFusion2 – 10MHz to 20MHz

• ProASICPLUS and ProASIC – 10 MHz.

During execution, the frequency set by the FREQUENCY statement in the PDB/STAPL file overrides the TCK frequency setting selected by you in the Programmer Settings dialog box unless you also select the Force TCK Frequency checkbox.

FlashPro Programmer Settings

Choose your programmer settings for FlashPro (see above figure). If you choose to add the Force TCK Frequency, select the appropriate MHz frequency. After you have made your selection(s), click **OK**.

Default Settings

- The Vpp, Vpn, Vdd(l), and Vddp options are checked (Vddp is set to 2.5V) to instruct the FlashPro programmer(s) to supply Vpp, Vpn, Vdd(l) and Vddp.
- The Driver TRST option is unchecked to instruct the FlashPro programmer(s) NOT to drive the TRST pin.
- The Force TCK Frequency option is unchecked to instruct FlashPro to use the TCK frequency specified by the Frequency statement in the STAPL file(s).

FlashPro Lite Programmer Settings

If you choose to add the Force TCK Frequency, select the appropriate MHz frequency. After you have made your selection(s), click **OK**.

Default Settings

- The Vpp and Vpn options are checked to instruct the FlashPro Lite programmer(s) to supply Vpp and Vpn.

- The Driver TRST option is unchecked to instruct the FlashPro Lite programmer(s) NOT to drive the TRST pin.
- The Force TCK Frequency option is unchecked to instruct the FlashPro Lite to use the TCK frequency specified by the Frequency statement in the STAPL file(s).

FlashPro4/3/3X Programmer Settings

For FlashPro3, you have the option of choosing the Set Vpump setting or the Force TCK Frequency. If you choose the Force TCK Frequency, select the appropriate MHz frequency. For FlashPro4/3X settings, you have the option of switching the TCK mode between Free running clock and Discrete clocking. After you have made your selections(s), click **OK**.

Default Settings

- The Vpump option is checked to instruct the FlashPro3 programmer(s) to supply Vpump to the device.
- The Force TCK Frequency option is unchecked to instruct the FlashPro3 to use the TCK frequency specified by the Frequency statement in the PDB/STAPL file(s).
- FlashPro3x default TCK mode setting is Free running clock

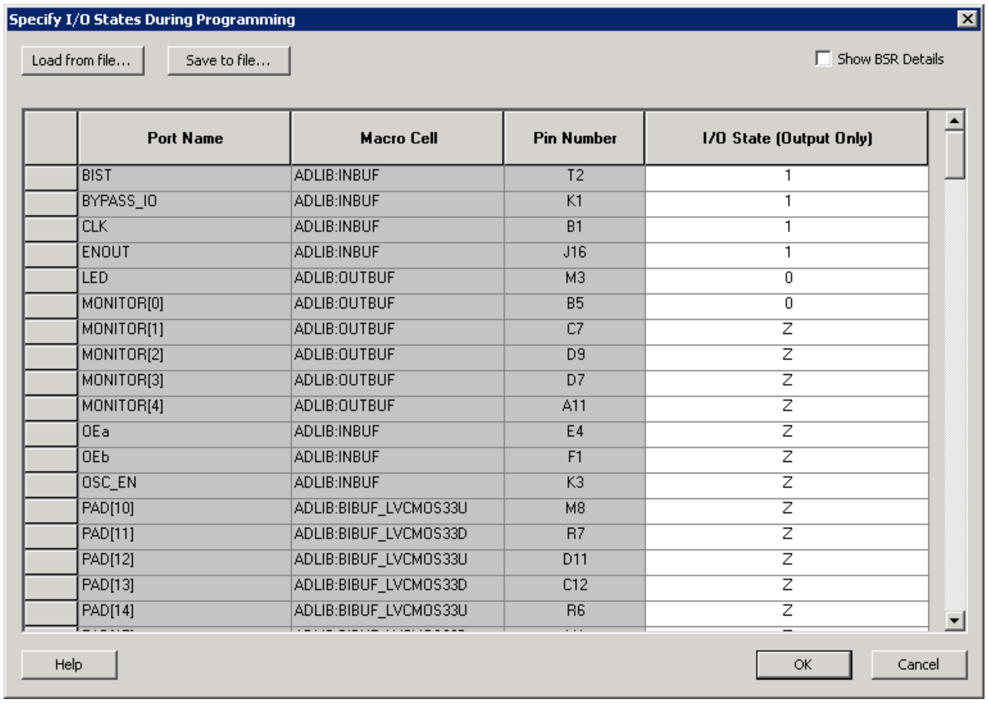
Device I/O States During Programming

In the Libero SoC Design Flow window expand **Edit Design Hardware Configuration** and double-click **Device I/O states during programming** to specify the I/O states prior to programming. In Libero SoC, this feature is only available once Layout is completed.

The default state for all I/Os is Tri-state.

To specify I/O statues during programming:

1. Sort the pins as desired by clicking any of the column headers to sort the entries by that header. Select the I/Os you wish to modify (as shown in the figure below).
2. Set the I/O Output state. You can set Basic I/O settings if you want to use the default I/O settings for your pins, or use Custom I/O settings to customize the settings for each pin. See the Specifying I/O States During Programming - I/O States and BSR Details help topic for more information on setting your I/O state and the corresponding pin values. Basic I/O state settings are:
    - 1 – I/O is set to drive out logic High
    - 0 – I/O is set to drive out logic Low
    - Last Known State: I/O is set to the last value that was driven out prior to entering the programming mode, and then held at that value during programming
    - Z - Tri-State: I/O is tristated



Port Name	Macro Cell	Pin Number	I/O State (Output Only)
BIST	ADLIB:INBUF	T2	1
BYPASS_IO	ADLIB:INBUF	K1	1
CLK	ADLIB:INBUF	B1	1
ENOUT	ADLIB:INBUF	J16	1
LED	ADLIB:OUTBUF	M3	0
MONITOR[0]	ADLIB:OUTBUF	B5	0
MONITOR[1]	ADLIB:OUTBUF	C7	Z
MONITOR[2]	ADLIB:OUTBUF	D9	Z
MONITOR[3]	ADLIB:OUTBUF	D7	Z
MONITOR[4]	ADLIB:OUTBUF	A11	Z
OEa	ADLIB:INBUF	E4	Z
OEb	ADLIB:INBUF	F1	Z
OSC_EN	ADLIB:INBUF	K3	Z
PAD[10]	ADLIB:BIBUF_LVCMOS33U	M8	Z
PAD[11]	ADLIB:BIBUF_LVCMOS33D	R7	Z
PAD[12]	ADLIB:BIBUF_LVCMOS33U	D11	Z
PAD[13]	ADLIB:BIBUF_LVCMOS33D	C12	Z
PAD[14]	ADLIB:BIBUF_LVCMOS33U	R6	Z

Figure 41 · I/O States During Programming Window

6. Click **OK** to save your settings.

   Note: NOTE: I/O States During programming will be used during programming or when exporting programming files.

Security Features Frequently Asked Questions

The following Frequently Asked Questions address the most common queries related to managing and programming SmartFusion2 Security Features.

I have configured the Security Policy Manager and enabled security in my design but I do not want to program my design with the Security Policy Manager features enabled. What do I do?

Go to Programming Features and un-check Security.

What is programmed when I click Program Device?

All features configured in your design and enabled in the Programming Features tool. Any features you have configured (such as eNVM or Security) are enabled for programming by default.

When I click Program Device is the programming file encrypted?

All programming files are encrypted. To generate programming files encrypted with UEK1 or UEK2 you must generate them from Export Programming File for field updates.

Note: NOTE: Once security is programmed, you must erase the security before attempting to reprogram the security.

How do I generate encrypted programming files with User Encryption Key 1/2?

- Configure the Security Policy Manager and specify a User Key Set 1 and User Key Set 2 (User Key Set 2 is available if you select Field Update Broadcast mode). Ensure the Security programming feature is enabled in Programming Features; it is enabled by default once you configure the Security Policy Manager.
- Export Programming File from Handoff Design for Production - <filename>_uek1.(stp/svf/spi/dat) and <filename>_uek2.(stp/svf/spi/dat) files are encrypted with UEK1 and UEK2 respectively. See Security Programming File Descriptions below for more information on programming files.

What are Security Programming Files?

See the Security Programming Files topic for more information.

Security Programming Files

Export Programming File (expand Handoff Design for Production in the Design Flow window) creates the following files:

<filename>_master.(stp/svf/spi/dat) - Created when Program Security and Design at Trusted Facility use model is specified in the Security Policy Manager. This is the master programming file; it includes all programming features enabled, User Key Set 1, User Key Set 2 (optionally if specified), and your security policy settings.

<filename>_security_only_master.(stp/svf/spi/dat) – Created when Program Security at Trusted Facility and Design at untrusted facility use model is specified in the Security Policy Manager. Master security programming file; includes User Key Set 1, User Key Set 2 (optionally if specified), and your security policy settings.

<filename>_uek1.(stp/svf/spi/dat) – Programming file encrypted with User Encryption Key 1 used for field updates; includes all your features for programming except security .

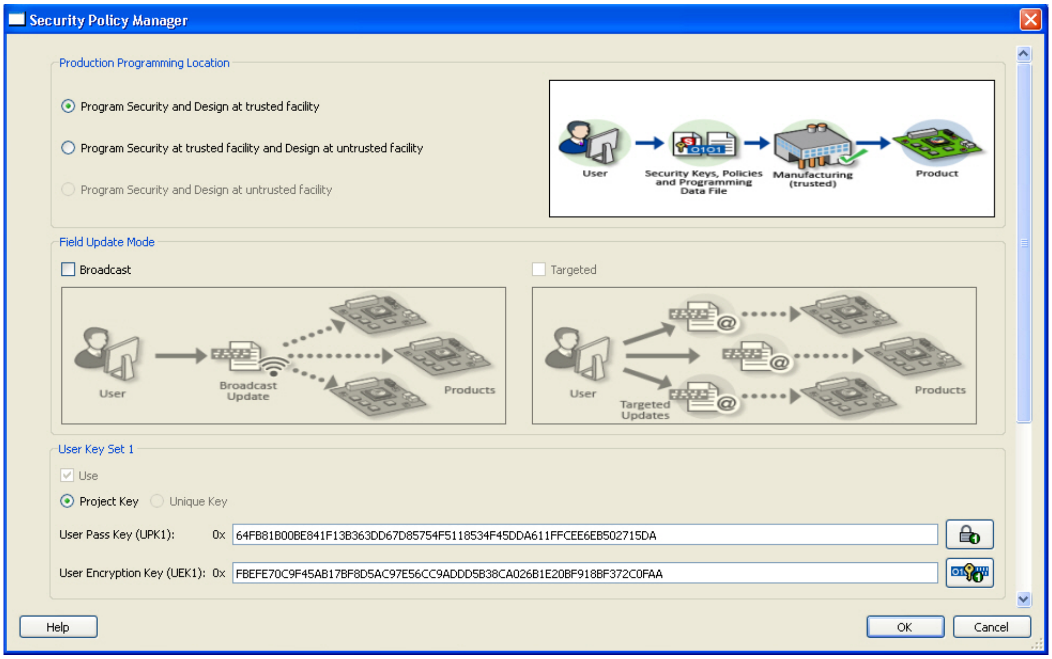
<filename>_uek2.(stp/svf/spi/dat) – Programming file encrypted with User Encryption Key 2 used for field updates; includes all your features for programming except security.

Security Policy Manager (SPM)

Expand **Configure Security and Programming Options**, double-click **Security Policy Manager** to customize the security settings in your design.

In this dialog box you can set your Secured Programming Use Model, User Key Entry and Security Policies. The security settings can be implemented in three different Production Programming locations.

Note: You must complete Place and Route prior to setting security using Security Policy Manager.



Production Programming Location

● Program Security and Design at trusted facility

○ Program Security at trusted facility and Design at untrusted facility

○ Program Security and Design at untrusted facility

Field Update Mode

□ Broadcast

□ Targeted

User Key Set 1

☑ Use

● Project Key ○ Unique Key

User Pass Key (UPK1): 0x 64FB81B00BE841F13B363DD67D85754F5118534F45DDA611FFCEE6EB502715DA

User Encryption Key (UEK1): 0x FBEFE70C9F45AB17BF8D5AC97E56CC9ADDD5B38CA026B1E20BF918BF372C0FAA

Figure 42 · Security Policy Manager Dialog Box (Partial)

Production Programming Location

Program Security and Design at trusted facility – Your design and security are programmed in a trusted facility.

Program Security at trusted facility and Design at un-trusted facility – Security is programmed at a trusted location and your design is programmed at an untrusted location. The Design bitstream is encrypted with a User Encryption Key (UEK).

Program Security and Design at un-trusted facility - Security and design are both programmed at an untrusted location.

Field Update Mode

Broadcast - Generic field design update intended for all users. User Encryption Key 1 (UEK1) or User Encryption Key 2 (UEK2) can be utilized for this update, as long as either key is common for all devices.

Targeted - An additional update feature that can be used to update the design data for a subset of the customer database.

Configuring User Keys

There are two sets of user keys:

- User Key Set 1 (UPK1, UEK1) is selected by default. User Passkey 1 (UPK1) protects the User Encryption Key 1 (UEK1) and all user security policy settings.
- User Key Set 2 (UPK2, UEK2) is an optional set of keys. These keys are available only if an update option is specified; click the checkbox to enable it.

Project Key for broadcast update and **Unique Key** for targeted update

Note that User Pass Key 2 (UPK2) protects only User Encryption Key 2 (UEK2).

Security Policies

Update Policy - Sets your Fabric, eNVM and Back Level protections. See the Update Policy topic for more information.

Debug Policy - Enables and sets your Debug Pass Key and debug options. See the Debug Policy topic for more information.

Protocol Policy - Configures the programming protocol to enable or disable. See the Programming Protocol Policy topic for more information.

Operational Integrity Policy - Enables you to check the Digest on power-up for Fabric and eNVM0. See the Operational Integrity Policy topic for more information.

Update Policy - Programming

This dialog box enables you to specify Components that can be updated in the field, and their field-update protection parameters.

Choose your protection options from the drop-down menus; click the appropriate checkbox to set your programming protection preferences.

Fabric update protection

- Erase/Write/Verify protected by UPK1 - Select this option to require UPK1 to erase, write, or verify the Fabric.
- Open for encrypted update with for UEK1 or UEK2 - Encrypted update is allowed with either UEK1 or UEK2 (if enabled).
- Custom

Custom

- OTP - One time programmable; you cannot reprogram the Fabric again.
- Protect Write/Erase by Pass Key - Select this option to require a UPK1 to write or erase your fabric.
- Protect Verify by Pass Key - Select this option to require a UPK1 to verify your fabric.

Note: SmartFusion2 always enforces encrypted programming, whether or not the Fabric/eNVM is protected by UPK1.

eNVM update protection

Erase/Write protected by UPK1 - Select this option to require UPK1 to erase or write to eNVM.

Custom

- OTP - eNVM is one time programmable; you cannot reprogram the eNVM.
- Protect Write/Erase by Pass Key - Select this option to require a passkey to write or erase your eNVM.
- Protect Verify/Read by Pass Key - Select this option to require a passkey to read and verify your eNVM through the JTAG/SPI-Slave.

Back Level protection - If enabled, this provides bitstream replay protection. The BACKLEVEL value limits the design versions that the device can update. So, only programming bistreams with DESIGNER > BACKLEVEL are allowed for programming.

Protect Programming Interface with Pass Key

These settings protect the following programing interfaces with UPK1:

- Auto Programming
- Cortex M3 IAP
- JTAG
- SPI Slave

For more technical information on the Protect Programming Interface with Pass Key option see the SmartFusion2 Programming User's Guide.

Permanent Security - Permanently secures all security settings.

Note: The dialog box informs you of the security settings and features that are no longer reprogrammable.

Debug Security Policy

Debug Security Policy is disabled in this release.

Protocol Policy

Protect programming protocols with UPK1. If a programming protocol is disabled, then UPK1 is required to program with that programming protocol.

Two protocols can be disabled:

- User Encryption Key 1
- User Encryption Key 2

If both protocols are disabled then device update is impossible.

Operational Integrity Policy

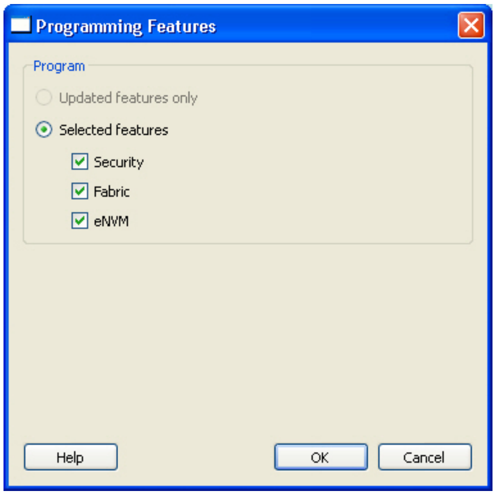
Add digest calculation and verification on power-up. Only features in the design can be selected. The available features are:

- Fabric
- eNVM

Note: NOTE: Only clients with Use as ROM selected are included in this check. Verify/Read eNVM must not be protected by UPK1 within the Update Policy to enable eNVM digest check.

Programming Features

Enables you to select which options you wish to program. Only features that have been added to your design are available for programming. For example, you cannot select eNVM for programming if you do not have an eNVM in your design.


Programming Features

Program

○ Updated features only

● Selected features

☑ Security

☑ Fabric

☑ eNVM

Figure 43 · Programming Features Dialog Box

Updated features only (not supported in this release) - Updates only the features that have changed since your last programming.

Selected features - Updates the features you select, regardless of whether or not they have changed since your last programming.

Update eNVM Memory Content

Double-click Update eNVM Memory Content to open the dialog box and modify your eNVM content.

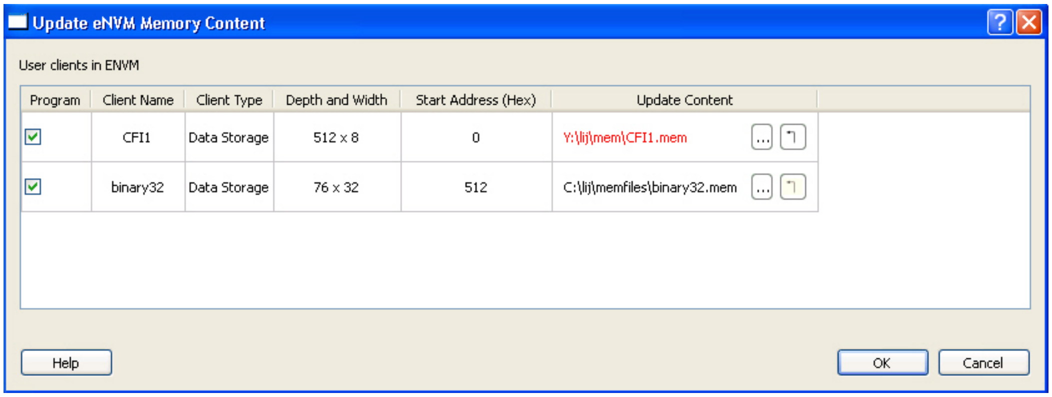
The Update eNVM Memory Content dialog box enables you to change your eNVM content for programming without having to rerun Compile and Place and Route. It is useful if you have reserved space in the eNVM configurator within the MSS for firmware development. Use the eNVM Memory Content dialog box when you have completed your firmware development and wish to incorporate your updated MEM file into the project. If you import your memory file into a project that has completed Place and Route you do not have to rerun Compile or Place and Route - you can program or export your programming file directly.

Click the checkbox in the **Program** column to enable or disable programming for each eNVM user client.

Click the **Browse** button in the Update Content column to change the memory file for a specific client.

The dialog box displays updated memory files in red text.

Click the **Reset** button in the Update Content column to return to the original file.


Update eNVM Memory Content

User clients in ENVM

Program	Client Name	Client Type	Depth and Width	Start Address (Hex)	Update Content
☑	CFI1	Data Storage	512 x 8	0	Y:\lij\mem\CFI1.mem
☑	binary32	Data Storage	76 x 32	512	C:\lij\memfiles\binary32.mem

Figure 44 · Update eNVM Memory Content Dialog Box

Program Device

If you have a device programmer connected you can double-click **Program Device** to execute your programming in batch mode with default settings.

If your programmer is not connected, or if your default settings are invalid, the Reports view lists the error(s).

See Also

Right-click menu options for Program Device

SmartFusion2 Programming - Default Settings

To view your default settings, from the **Project** menu choose **Project Settings**.

To program your SmartFusion2 device with default settings:

1. Create a Libero Soc project using any SmartDesign component. For example, you can create a project using a SmartDesign component, such as a simple fabric module and a MSS block with Flash Memory module (eNVM).
2. Click the Build button to complete Synthesis, Place and Route and program the device with default settings. The default settings do not contain any security settings; use the Security Policy Manager (SPM) to manage your settings prior to programming your device.

SmartFusion2 Programming - Custom Settings

Custom Programming Settings enable you to build the JTAG chain, define programmer settings, set I/O states during programming and run scan chain.

1. To create a JTAG chain, in the Design Flow window expand **Edit Design Hardware Configuration**, right-click **Programming Connectivity and Interface** and choose **Open Interactively**. It opens a schematic view of the devices connected in a JTAG chain; all the devices are targeted by default.

The Programming and Connectivity Interface detects and constructs the JTAG chain automatically. Use the interface to add devices manually.

When you add Microsemi devices you can either load the STP or PDB file or add the device from a drop-down list. You must provide the IR length and Max TCK frequency OR load the BSDL file for non-Microsemi devices.

2. Right-click **Programmer Settings** and choose **Open Interactively** to view your programmer settings. If necessary, click Edit Programmer Settings to specify custom settings for your programmer.
3. Right-click **Device I/O States During Programming** and choose **Open Interactively** to open the Specify I/O States During Programming dialog box and set your device I/O states. Click OK to save your settings and continue.
4. Expand Configure Security, right-click **Security Policy Manager** and choose **Open Interactively** to specify your Secured Programming Use Model, User Key Entry and Security Policies.

Exit Codes

Exit Code	Exit Message	Possible Cause	Possible Solution
0	Passed (no error)	-	-
5	Invalid, corrupted bitstream	Unstable voltage level Signal integrity issues on JTAG pins	Monitor related power supplies that cause the issue during programming; check for transients outside of Microsemi specifications. See your device datasheet for more information on transient specifications. Monitor JTAG supply pins during programming; measure JTAG signals for noise or reflection
	Failed to set programming voltage		
	Device is busy		
	Failed to read		

Exit Code	Exit Message	Possible Cause	Possible Solution
	design information		
	Failed to enter programming mode		
	Failed to set programming mode		
	Failed to read programming information		
6	Failed to verify IDCODE	Incorrect programming file Incorrect device in chain Signal integrity issues on JTAG pins	Choose the correct programming file and select the correct device in the chain. Measure JTAG pins and noise for reflection. If TRST is left floating then add pull-up to pin. Reduce the length of Ground connection.
10	Authentication Error - See Authentication Error Codes If Authentication Error is not displayed, see Error Codes		
-35	Failed to unlock User Pass Key 1	Pass key in file does not match device	Provide a programming file with a pass key that matches pass key programmed into the device
	Failed to unlock User Pass Key 2		

SmartFusion2 Programming Authentication Error Codes (AUTHERRCODE)

The table below lists authentication error codes for SmartFusion2 devices.

Errors related to programming failures (ERRORCODE errors) are summarized in SmartFusion2 Programming Error Codes.

Table 3 · SmartFusion2 Programming Authentication Error Codes

AUTHERRCODE	Description	Possible Cause	Possible Solution
0	Passed (no error)	-	-

AUTHERRCODE	Description	Possible Cause	Possible Solution
1, 2	Invalid, corrupted bitstream	Programming file has been corrupted	Regenerate programming file
3	Invalid, corrupt encryption key	File contains an encrypted key that does not match the device File contains user encryption key, but device has not been programmed with the user encryption key Device has user encryption key 1/2 enforced and you are attempting to reprogram security settings	Provide a programming file with an encryption key that matches that on the device First program security with master programming file, then program with user encryption 1/2 field update programming files You must first ERASE security with the master security file, then you can reprogram new security settings
4	Invalid, corrupted bitstream	Programming file has been corrupted	Regenerate the programming file
5	Back level not satisfied	Design version is not higher than the back-level programmed device	Generate a programming file with a design version higher than the back level version
7	DSN binding mismatch	DSN specified in programming file does not match the device being programmed	Use the correct programming file with a DSN that matches the DSN of the target device being programmed
8	Invalid, corrupted bitstream	Programming file has been corrupted	Regenerate the programming file
9	Insufficient device capabilities	Device does not support the capabilities specified in programming file	Generate a programming file with the correct capabilities for the target device
10	Incorrect DEVICEID	Incorrect programming file Incorrect device in chain Signal integrity issues on JTAG pins	Choose the correct programming file and select the correct device in chain Measure JTAG pins and noise or reflection. If TRST is left floating, then add pull-up to pin

AUTHERRCODE	Description	Possible Cause	Possible Solution
			Reduce the length of ground connection
11	Unsupported bitstream protocol version	Old programming file	Generate programming file with latest version of Libero SoC
12	Verify not permitted on this bitstream		

SmartFusion2 Programming Error Codes (ERRORCODE)

The table below lists authentication error codes for SmartFusion2 devices.

Errors related to authentication failures are summarized in SmartFusion2 Programming Authentication Error Codes.

Table 4 · SmartFusion2 Programming Error Codes

ERRORCODE	Description
0	Passed (no error)
1	Fabric verification failed
2	Device security prevented operation
3	Programming mode not enabled
4	eNVM programing operation failed
5	eNVM verify operation failed

Programming File Actions - SmartFusion2

Libero SoC enables you to program security settings, FPGA Array, and eNVM features for SmartFusion2 device support. You can program these features separately using different programming files or you can combine them into one programming file.

Table 5 · Programming File Actions

Action	Description
PROGRAM	Programs all selected family features: FPGA Array, targeted eNVM clients, and security settings.
VERIFY	Verifies all selected family features: FPGA Array, targeted eNVM clients, and security settings.
ERASE	Erases the selected family features: FPGA Array and Security settings.

Action	Description
DEVICE_INFO	Displays the IDCODE, the design name, the checksum, and device security settings and programming environment information programmed into the device.
READ_IDCODE	Reads the device ID code from the device
ENC_DATA_AUTHENTICATOIN	Encrypted bitstream authentication data.

Options Available in Programming Actions

The table below shows the options available for specific programming actions.

Table 6 · Programming File Actions - Options

Action	Option and Description
PROGRAM	DO_VERIFY - Enables or disables programming verification

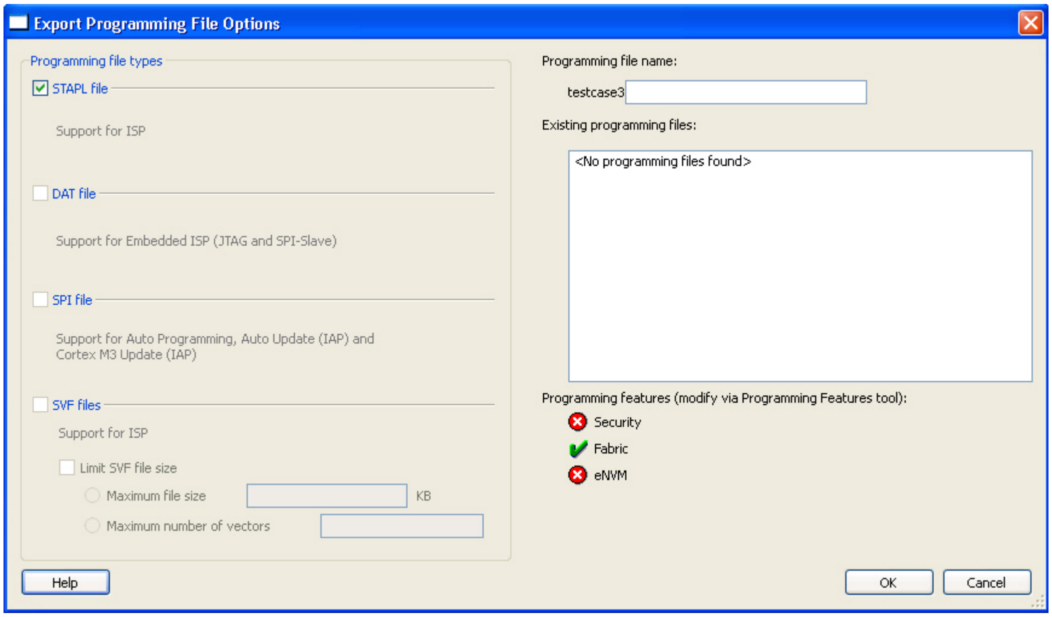
Export Programming Files - SmartFusion2

Export Programming Files enables you to export STAPL, DAT, SPI and SVF programming files. Go to Programming Features to change the features selected for programming.

To generate a STAPL file with default settings, in the Libero SoC Design Flow window expand Handoff Design for Production and double-click **Export Programming File**.

To modify your programming file settings before you export:

1. Right-click **Export Programming File** and choose **Configure Options**. The Export Programming File Options dialog box opens.
2. Choose your options, such as **DAT file** if you wish to include support for Embedded ISP, or **SPI file** if you need support for IAP.
3. Enter your **Programming file name** and click **OK**.



Export Programming File Options

Programming file types

☑ STAPL file

Support for ISP

□ DAT file

Support for Embedded ISP (JTAG and SPI-Slave)

□ SPI file

Support for Auto Programming, Auto Update (IAP) and Cortex M3 Update (IAP)

□ SVF files

Support for ISP

□ Limit SVF file size

○ Maximum file size KB

○ Maximum number of vectors

Programming file name:

testcase3

Existing programming files:

<No programming files found>

Programming features (modify via Programming Features tool):

☒ Security

✓ Fabric

☒ eNVM

Figure 45 · Export Programming File Options Dialog Box

Limit File Size

Some testers may have memory size restrictions for a single SVF file. The SVF limit file option enables you to limit the size of each SVF file by either file size or vectors.

The generated SVF files append an index to the file name indicating the sequence of files. The format is:

```
<SVF_filename>_XXXXX.svf
```

where XXXXX is the index of the SVF file. The first SVF file begins with <SVF_filename>_00000.svf and increments by 1 until file generation is complete.

Maximum file size: Max file size limit for the SVF file; use this option to limit your SVF file size based on number of kB.

Maximum number of vectors: Max vector limit for the SVF file; use this option to limit the size of your SVF based on number of vectors.

Programming features – Lists which programming features will be included in the exported programming files.

Note: NOTE: SPI master files will not include the Security programming feature.

Programming File Types

STAPL Files

The Standard Test And Programming Language (STAPL) is designed to support the programming of programmable devices and testing of electronic systems, using the IEEE Standard 1149.1: “Standard Test Access Port and Boundary Scan Architecture” (commonly referred to as JTAG) interface. As a STAPL file is executed, signals are produced on the IEEE 1149.1 interface, as described in the STAPL file. STAPL operates on a single IEEE 1149.1 chain. STAPL supports the programming of any IEEE 1149.1-compliant programmable device.

STAPL has support for programming and test systems with user interface features. A single STAPL file may perform several different functions, such as programming, verifying, and erasing a programmable device.

DAT Files

Export a DAT file if you want to include support for the features listed in the dialog box.

SPI File

Export SPI (serial peripheral interface) file if you want to include support for the features listed in the dialog box.

SVF Files

Courtesy Serial Vector Format Specification from ASSET InterTech, 1999:

Serial Vector Format (SVF) is the media for exchanging descriptions of high-level IEEE 1149.1 bus operations. In general, IEEE 1149.1 bus operations consist of scan operations and movements between different stable states on the IEEE 1149.1 state diagram. SVF does not explicitly describe the state of the IEEE 1149.1 bus at every Test Clock.

The SVF file is defined as an ASCII file that consists of a set of SVF statements. The maximum number of characters on a line is 256, although one SVF statement can span more than one line. Each statement consists of a command and associated parameters. Each SVF statement is terminated by a semicolon. SVF is not case sensitive.

SmartFusion2 Programming Tutorial

The SmartFusion2 Programming Tutorial describes the basic steps for SmartFusion2 programming.

Only the bold steps in the Design Flow window are required to complete and program your design. Note that the bold steps are completed automatically if you use the Build button.

1. MSS Configuration - eNVM

eNVM configuration enables you to configure eNVM as a ROM so that it can be included in the eNVM digest calculations.

Data Security Configuration controls which masters have access to which memory region within the MSS.

2. Generate Fabric Programming Data

Generates the map and DCA file; it does not generate the programming file.

3. Edit Design Hardware Configuration

Configures Device I/O States During Programming.

4. Configure Security and Programming Options

- Security Policy Manager
- Programming Features
- Update eNVM Memory Content

5. Program Design

Configure Actions/Procedures (sets programming options) and programs your device.

5. Handoff Design for Production

- Export Programming File
- Export BSDL

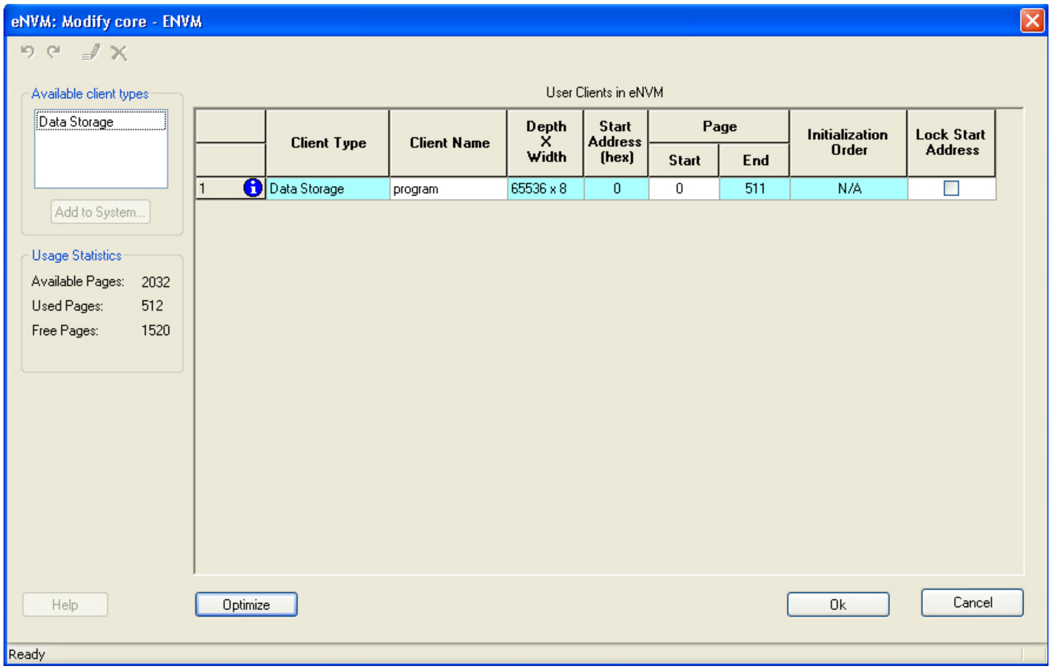
MSS Configuration - eNVM

eNVM Configuration

You must create a MSS to configure your eNVM.

eNVM configuration enables you to configure eNVM as a ROM so that it can be included in the eNVM digest calculations. To do so:

1. Open your MSS and double-click the **eNVM block** to open the **eNVM configuration dialog box**, as shown in the figure below.



eNVM: Modify core - ENVM

Available client types

Data Storage

Add to System...

Usage Statistics

Available Pages:	2032
Used Pages:	512
Free Pages:	1520

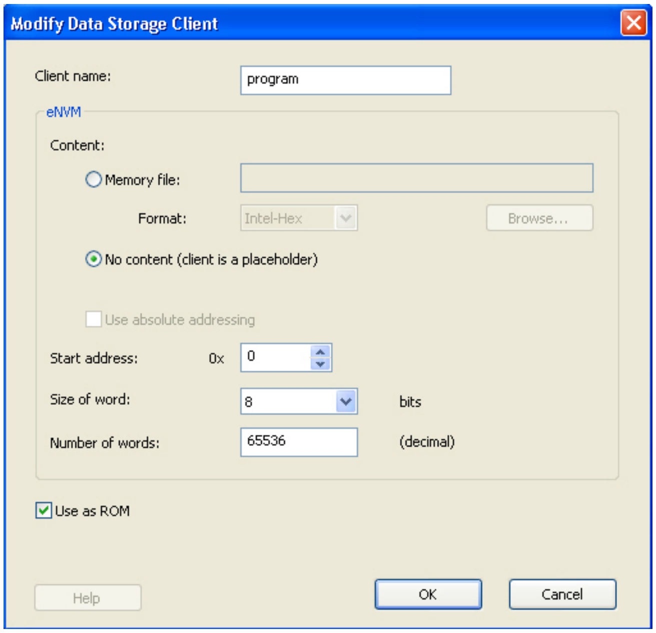
User Clients in eNVM

	Client Type	Client Name	Depth X Width	Start Address (hex)	Page		Initialization Order	Lock Start Address
					Start	End		
1 ℹ	Data Storage	program	65536 x 8	0	0	511	N/A	□

Figure 46 · eNVM Configuration Dialog Box

The example design shown in the figure above already has a Data Storage Client.

2. Double-click the **Data Storage Client** to open the **Modify Data Storage Client dialog box**, as shown in the figure below.



Modify Data Storage Client

Client name: program

eNVM

Content:

○ Memory file:

Format: Intel-Hex

● No content (client is a placeholder)

□ Use absolute addressing

Start address: 0x 0

Size of word: 8 bits

Number of words: 65536 (decimal)

☑ Use as ROM

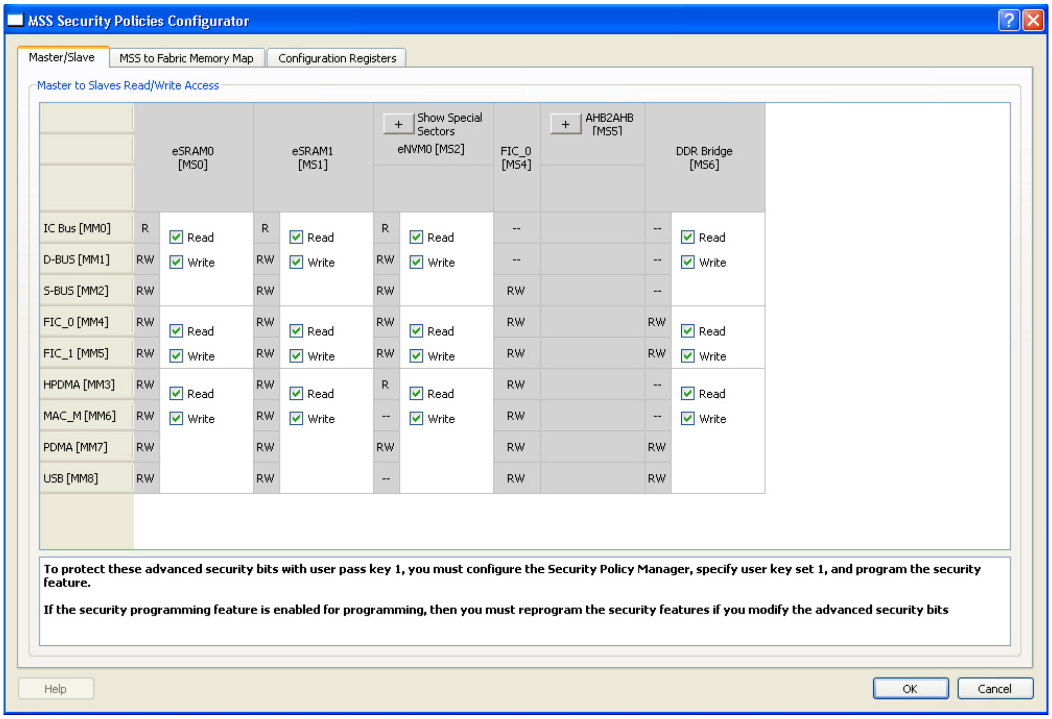
Figure 47 · Modify Data Storage Client Dialog Box - Use as ROM Selected

3. Click **Use as ROM** (as shown in the figure above) to configure the memory region as a ROM and include the eNVM digest calculations.
4. Click **OK** in the Modify Data Storage Client dialog box to continue.
5. Click **OK** in the eNVM Configuration dialog box to return to the MSS.

Enable Data Security

The Data Security Configuration controls which masters have access to which memory region within the MSS. To configure your data security:

Double-click the **Security** block in the MSS to open the **MSS Security Policies Configurator**, as shown in the figure below.



	eSRAM0 [MS0]		eSRAM1 [MS1]		+ Show Special Sectors eNVM0 [MS2]		FIC_0 [MS4]	+ AHB2AHB [MS5]	DDR Bridge [MS6]	
IC Bus [MM0]	R	☑ Read ☑ Write	R	☑ Read ☑ Write	R	☑ Read ☑ Write	--		--	☑ Read ☑ Write
D-BUS [MM1]	RW		RW		RW		--		--	
S-BUS [MM2]	RW		RW		RW		RW		--	
FIC_0 [MM4]	RW	☑ Read ☑ Write	RW	☑ Read ☑ Write	RW	☑ Read ☑ Write	RW		RW	☑ Read ☑ Write
FIC_1 [MM5]	RW		RW		RW		RW		RW	
HPDMA [MM3]	RW	☑ Read ☑ Write	RW	☑ Read ☑ Write	R	☑ Read ☑ Write	RW		--	☑ Read ☑ Write
MAC_M [MM6]	RW		RW		--		RW		--	
PDMA [MM7]	RW		RW		RW		RW		RW	
USB [MM8]	RW		RW		--		RW		RW	

To protect these advanced security bits with user pass key 1, you must configure the Security Policy Manager, specify user key set 1, and program the security feature.

If the security programming feature is enabled for programming, then you must reprogram the security features if you modify the advanced security bits

Figure 48 · MSS Security Policies Configurator

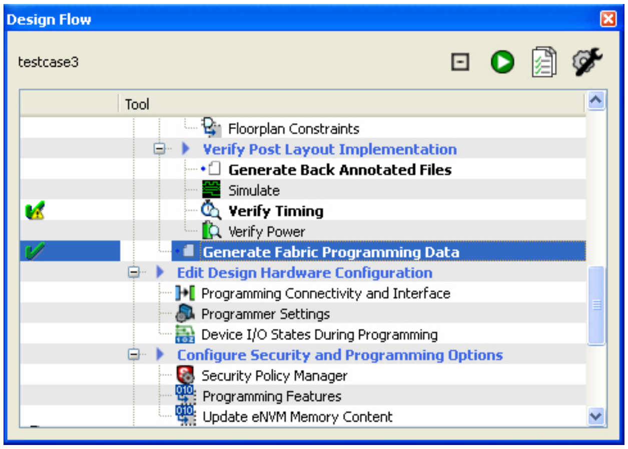
Masters are listed on the left, Slaves are shown at the top. All Masters have access to all Slaves by default; click to enable or disable Read/Write access for specific Masters and Slaves.

Restrict your Master/Slave Read/Write access according to your preference and click **OK** to continue.

Generate Fabric Programming Data - SmartFusion2 Only

Generates programming data for your design. This operation is completed automatically as the last step if you use the Build button.

When the process is complete a green check appears next to the operation in the Design Flow window and information messages appear in the Log window (shown in the figure below).



Design Flow

testcase3

Tool

- Floorplan Constraints
- Verify Post Layout Implementation
  - Generate Back Annotated Files
  - Simulate
  - ✓ Verify Timing
  - Verify Power
- ✓ Generate Fabric Programming Data
- Edit Design Hardware Configuration
  - Programming Connectivity and Interface
  - Programmer Settings
  - Device I/O States During Programming
- Configure Security and Programming Options
  - Security Policy Manager
  - Programming Features
  - Update eNVM Memory Content

Figure 49 · Generate Fabric Programming Data (Complete)

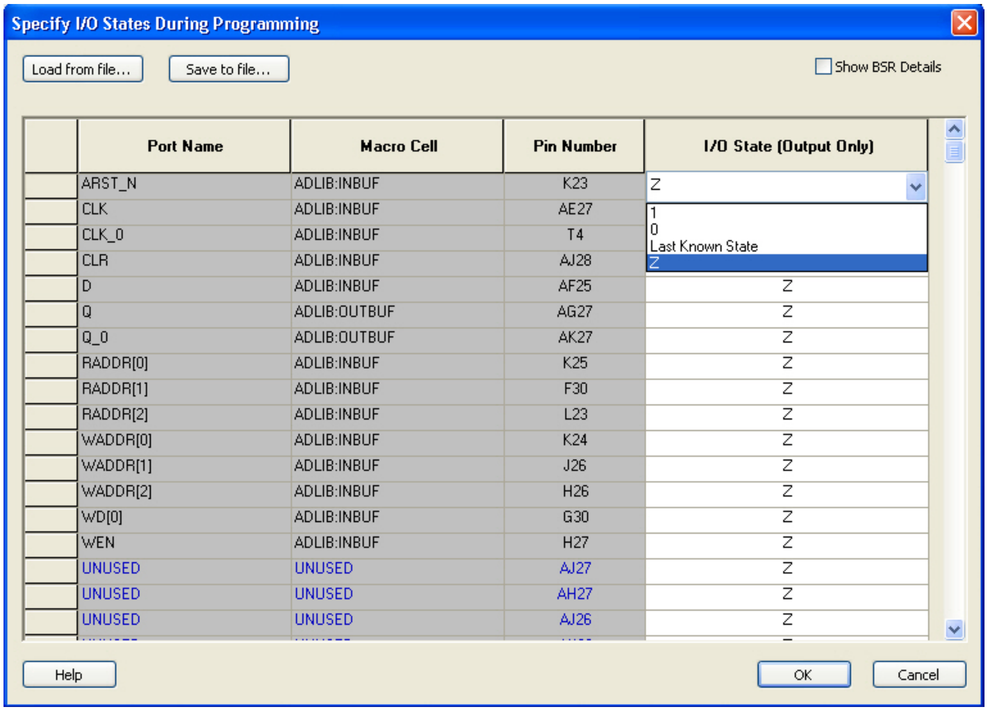
Edit Design Hardware Configuration - Device I/O States During Programming

You can configure your FPGA I/Os while the device is being programmed using Device I/O States During Programming.

In the Design Flow window, expand **Edit Design Hardware Configuration** and double-click **Device I/O States During Programming**.

The Device I/O States During Programming dialog box appears.

Click a value in the I/O State (Output Only) column to set your I/O State options according to your preference, as shown in the figure below.



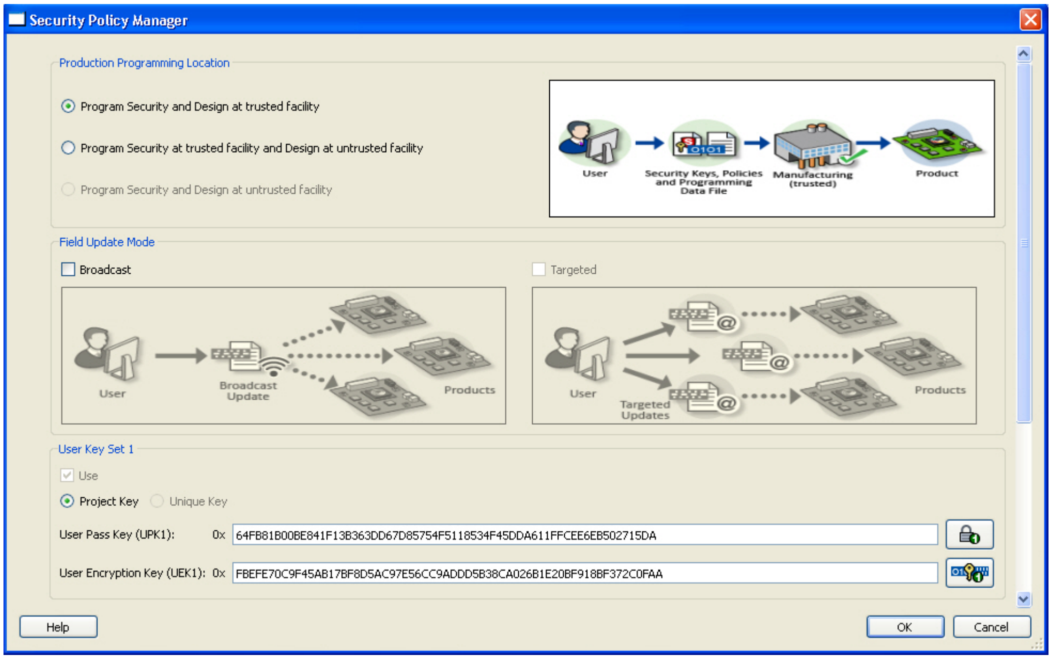
Specify I/O States During Programming

Port Name	Macro Cell	Pin Number	I/O State (Output Only)
ARST_N	ADLIB:INBUF	K23	Z
CLK	ADLIB:INBUF	AE27	1
CLK_0	ADLIB:INBUF	T4	0
CLR	ADLIB:INBUF	AJ28	Last Known State Z
D	ADLIB:INBUF	AF25	Z
Q	ADLIB:OUTBUF	AG27	Z
Q_0	ADLIB:OUTBUF	AK27	Z
RADDR[0]	ADLIB:INBUF	K25	Z
RADDR[1]	ADLIB:INBUF	F30	Z
RADDR[2]	ADLIB:INBUF	L23	Z
WADDR[0]	ADLIB:INBUF	K24	Z
WADDR[1]	ADLIB:INBUF	J26	Z
WADDR[2]	ADLIB:INBUF	H26	Z
WD[0]	ADLIB:INBUF	G30	Z
WEN	ADLIB:INBUF	H27	Z
UNUSED	UNUSED	AJ27	Z
UNUSED	UNUSED	AH27	Z
UNUSED	UNUSED	AJ26	Z

Figure 50 · Set I/O State

Configure Security Policy Manager

To configure your security settings, expand **Configure Security and Programming Options** and double-click **Security Policy Manager**. This opens the Security Policy Manager dialog box, as shown in the figure below.



Security Policy Manager

Production Programming Location

● Program Security and Design at trusted facility

○ Program Security at trusted facility and Design at untrusted facility

○ Program Security and Design at untrusted facility

Field Update Mode

□ Broadcast

□ Targeted

User Key Set 1

☑ Use

● Project Key ○ Unique Key

User Pass Key (UPK1): 0x 64FB81B00BE841F13B363DD67D85754F5118534F45DDA611FFCEE6EB502715DA

User Encryption Key (UEK1): 0x FBEFE70C9F45AB17BF8D5AC97E56CC9ADDD5B38CA026B1E20BF918BF372C0FAA

Figure 51 · Security Policy Manager Dialog Box (Partial)

The Security Policy Manager has scenarios that enable security settings based on your likely needs. Choose the scenario that best describes your process. Note that you can further customize each security option individually after you select your scenario.

Field Update Mode enables you to program all devices with a single key (Broadcast) or program individual devices with unique keys (Targeted).

Click the **Generate Key** button to generate a new **User Pass Key** (UPK) or **User Encryption Key** (UEK).

User Key Set 2 is available only if you are using a Field Update Mode.

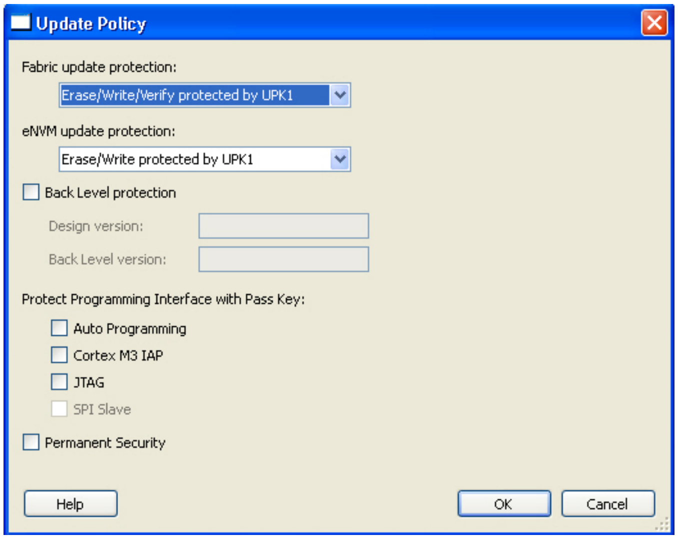
After you create your User Keys you can set your Security Policies. Note that each Security Policy can be enabled or disabled any time by clicking the **Use** checkbox underneath it.

Click **Update Policy** to open the Update Policy dialog box. The Update Policy dialog box enables you to set your Fabric and eNVM Update Protection, as shown in the figure below.

Back Level protection limits how many versions back a design can be programmed. For example, if you are on version 5 of a design and want to enable programming only back to version 4 you can set that option here.

Protect your **Programming Interface** with a pass key according to your design preferences. This requires a UPK1 match in order to utilize the protected programming interface.

Click **Permanent Security** to make the security settings permanent (they cannot be modified once programmed). After you configure your settings click **OK** to continue.



Update Policy

Fabric update protection:

Erase/Write/Verify protected by UPK1

eNVM update protection:

Erase/Write protected by UPK1

□ Back Level protection

Design version:

Back Level version:

Protect Programming Interface with Pass Key:

□ Auto Programming

□ Cortex M3 IAP

□ JTAG

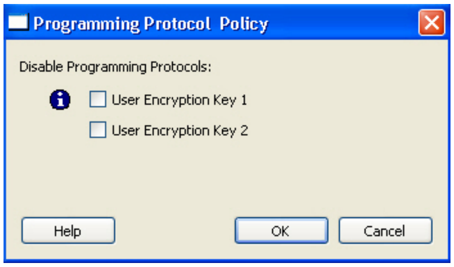
□ SPI Slave

□ Permanent Security

Figure 52 · Update Policy Dialog Box

Click **Protocol Policy** to open the Protocol Policy dialog box, as shown in the figure below.

Protocol policy allows you to disable programming protocols. Click the checkbox adjacent to your User Encryption Key to disable it. If a programming protocol is disabled, then UPK1 is required to program with that programming protocol.



Programming Protocol Policy

Disable Programming Protocols:

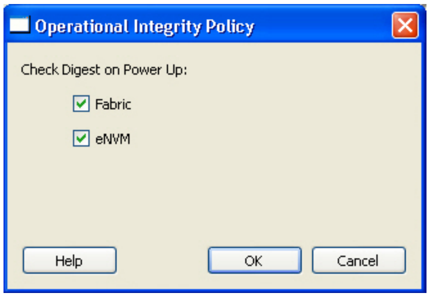
□ User Encryption Key 1

□ User Encryption Key 2

Figure 53 · Protocol Policy Dialog Box

Click **Operational Integrity Policy** to open the Operational Integrity Policy dialog box. Click the checkbox to Check Digest on Power Up (as shown in the figure below).

Note: NOTE: eNVM is only enabled if the feature has been configured, at least 1 client has Use as ROM selected, and eNVM verify/read is NOT protected by pass key within the update policy.



Operational Integrity Policy

Check Digest on Power Up:

☑ Fabric

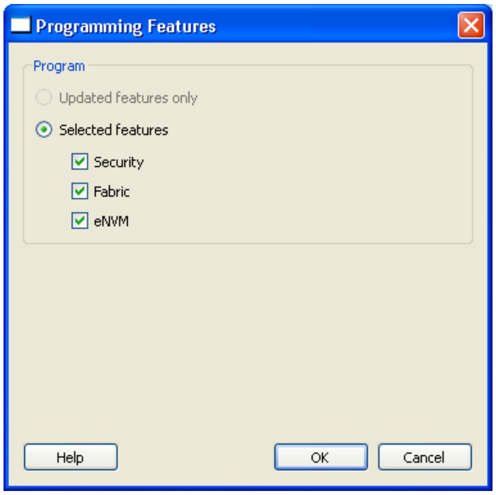
☑ eNVM

Figure 54 · Operational Integrity Policy Dialog Box

Click **OK** to save your policies and proceed.

Programming Features

Expand Configure Security and Programming Options and double-click Programming Features to open the Programming Features dialog box, as shown in the figure below.



Programming Features

Program

○ Updated features only

● Selected features

☑ Security

☑ Fabric

☑ eNVM

Figure 55 · Programming Features Dialog Box

You can program your Security, Fabric, eNVM or any combination.

Features (Security, Fabric, eNVM) are enabled for programming by default when you add them to your design. If you manually disable a feature in this dialog box then you must re-enable it here if you want it included during programming.

Update eNVM Memory Content

Double-click Update eNVM Memory Content to open the dialog box and modify your eNVM content.

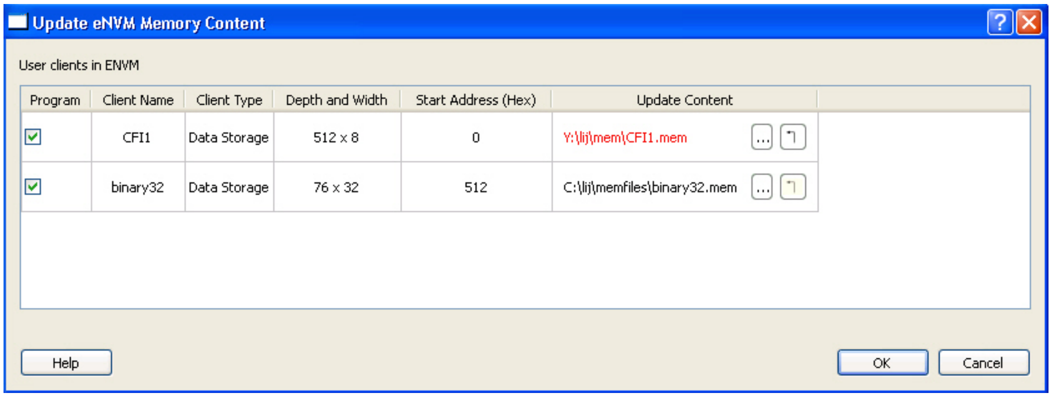
The Update eNVM Memory Content dialog box enables you to change your eNVM content for programming without having to rerun Compile and Place and Route. It is useful if you have reserved space in the eNVM configurator within the MSS for firmware development. Use the eNVM Memory Content dialog box when you have completed your firmware development and wish to incorporate your updated MEM file into the project. If you import your memory file into a project that has completed Place and Route you do not have to rerun Compile or Place and Route - you can program or export your programming file directly.

Click the checkbox in the **Program** column to enable or disable programming for each eNVM user client.

Click the **Browse** button in the Update Content column to change the memory file for a specific client.

The dialog box displays updated memory files in red text.

Click the **Reset** button ↰ in the Update Content column to return to the original file.



Update eNVM Memory Content

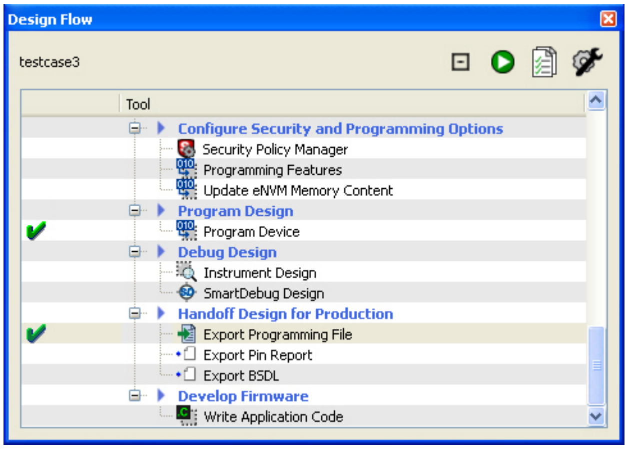
User clients in ENVM

Program	Client Name	Client Type	Depth and Width	Start Address (Hex)	Update Content
☑	CFI1	Data Storage	512 x 8	0	Y:\lij\mem\CFI1.mem
☑	binary32	Data Storage	76 x 32	512	C:\lij\memfiles\binary32.mem

Figure 56 · Update eNVM Memory Content Dialog Box

Program Design - Program Device

Expand Program Design and double-click Program Device to program your device with default settings, as shown in the figure below.



Design Flow

testcase3

Tool

- Configure Security and Programming Options
  - Security Policy Manager
  - Programming Features
  - Update eNVM Memory Content
- Program Design
  - ✓ Program Device
- Debug Design
  - Instrument Design
  - SmartDebug Design
- Handoff Design for Production
  - ✓ Export Programming File
  - Export Pin Report
  - Export BSDL
- Develop Firmware
  - Write Application Code

Figure 57 · Program Device in the Design Flow Window

Right-click **Program Device** and choose from the following menu options:

- **Clean and Run All** - Cleans all tools, deletes all reports and output files and runs through programming. All logs will be updated with new files.
- **Clean** - deletes only reports and output files associated with the Program Device; the other tool files and reports are unaffected.
- **Configure Actions/Procedures** - Enables you to set the specific Action you wish to program. Select your programming action from the dropdown menu.

Handoff Design for Production

In order to handoff your design for production you must export a programming file or generate a BSDL file.

Export Programming File

Expand **Handoff Design for Production** and double-click **Export Programming File** to create a file that you can use to program your part.

Right-click Export Programming file to:

- **Run** - Same as double-click behavior; exports your device programming file for production.
- **Clean and Run All** - Removes all data and output from tools run previously and runs back through to the current step.
- **Clean** - Cleans the output for the current tool.
- **Configure Options** - Opens the Export Programming File Options dialog box. It enables you to choose your Programming file type (STAPL, DAT, SPI, SVFJ), create a unique Programming file name and view your Programming features. The exported files are listed below; note that <filename>_master and <filename>_security_master_only are NOT exported for the SPI file type.

When the file is exported successfully a green check appears next to Export Programming File in the Design Flow Window (as shown in the figure below).

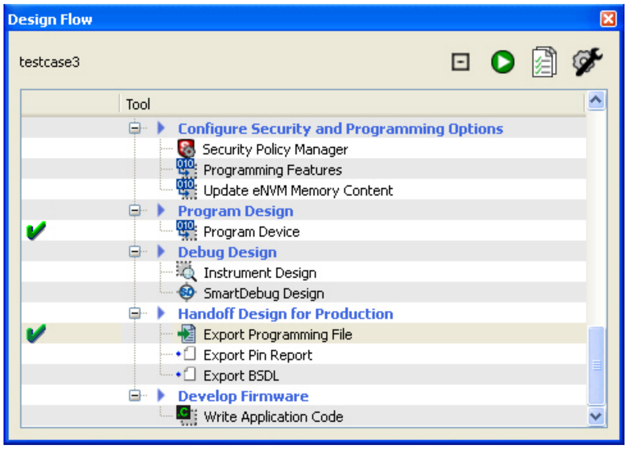


Figure 58 · Export Programming File Complete (Design Flow Window)

The file is exported to the directory <project>/designer/<project_name>/export. Exported programming files vary depending on whether or not security is specified/enabled for programming and which Programming Use Model you select.

Security Feature Not Enabled for Programming

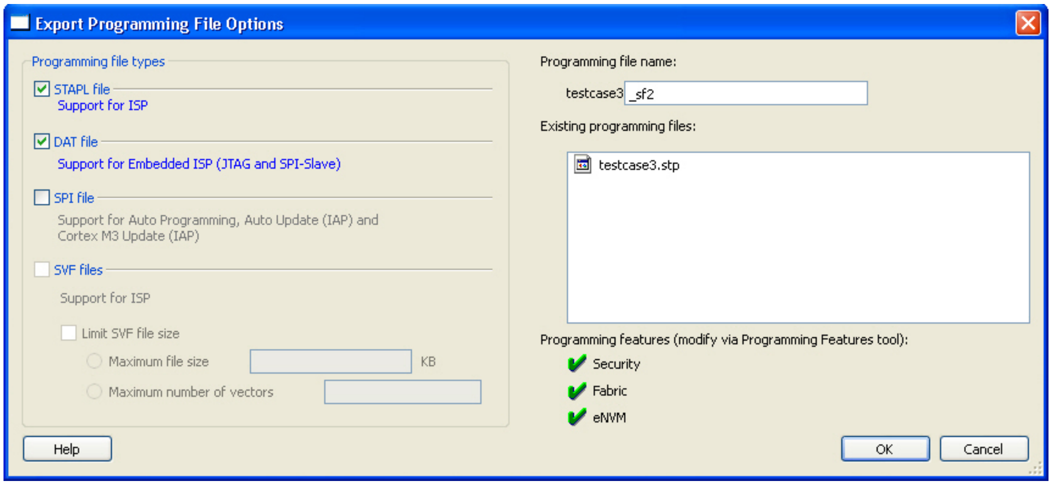
Software exports <filename>.stp.

Security Feature Enabled for Programming with the Use Model Program Security and Design at Trusted Facility

- <filename>_master.stp – Master programming file; includes all programming features enabled, User Key Set 1, User Key Set 2 (if Field Update mode is enabled), and security policy settings.
- <filename>_secured_uek1.stp – Programming file encrypted with User Encryption Key 1 used for field updates; includes all your selected Programming Features except Security
- <filename>_secured_uek2.stp (if Field Update mode is enabled) – Programming file encrypted with User Encryption Key 2 used for field updates; includes all Programming Features except Security

Security feature Enabled for Programming with the Use Model Program Security at Trusted Facility and Design at Untrusted Facility

- <filename>_security_only_master.stp - Master programming file ; includes User Key Set 1, User Key Set 2 (if Field Update mode is enabled), and Security Policy settings.
- <filename>_secured_uek1.stp – Programming file encrypted with User Encryption Key 1 used for field updates; includes all selected Programming Features except Security
- <filename>_secured_uek2.stp (optionally, if Field Update mode is enabled) – Programming file encrypted with User Encryption Key 2 used for field updates; includes all your selected Programming Features except Security



Export Programming File Options

Programming file types

☑ STAPL file
Support for ISP

☑ DAT file
Support for Embedded ISP (JTAG and SPI-Slave)

□ SPI file
Support for Auto Programming, Auto Update (IAP) and Cortex M3 Update (IAP)

□ SVF files
Support for ISP

□ Limit SVF file size

○ Maximum file size KB

○ Maximum number of vectors

Programming file name:

testcase3 _sf2

Existing programming files:

testcase3.stp

Programming features (modify via Programming Features tool):

✓ Security

✓ Fabric

✓ eNVM

Figure 59 · Export Programming File Options Dialog Box

Export BSDL File

Double-click **Export BSDL File** to generate a BSDL file for your project.

Right-click **Export BSDL File** and choose **Clean and Run All** to remove all data and output from tools run previously and rerun the Design Flow up through this point.

Programming SmartFusion in the Libero SoC

Double-click **Program Device** to create a programming file (if necessary) and program your device with default settings.

Right-click **Program Device** and choose **Open Interactively** to open FlashPro.

Generating Programming Files

Generate a Programming File in FlashPoint

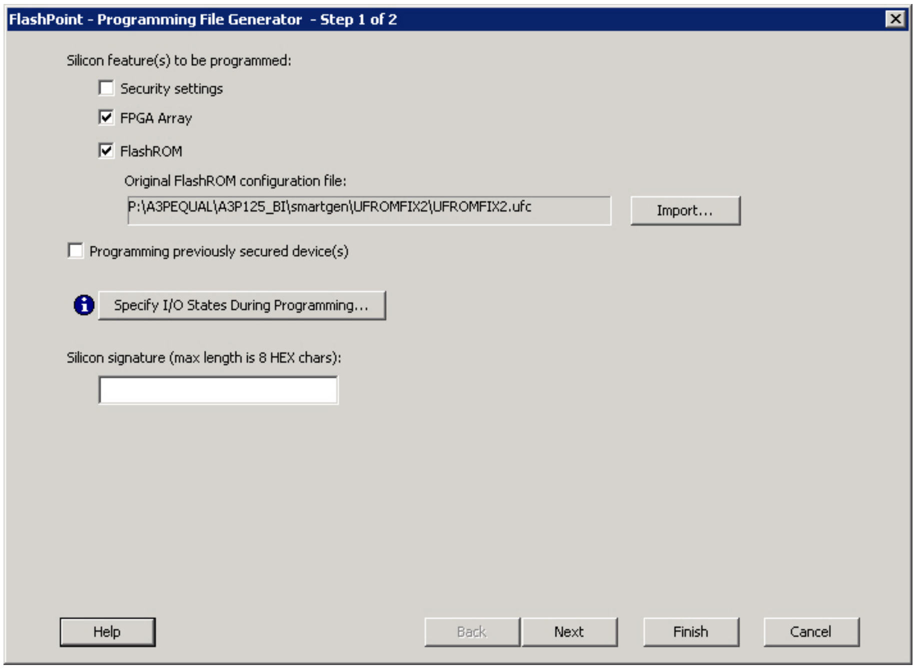
FlashPoint enables you to program security settings, FPGA Array, and FlashROM features for IGLOO, ProASIC3, SmartFusion, Fusion family devices. You can program these features separately using different programming files or you can combine them into one programming file. Each feature is listed as a silicon feature in the GUI.

Note: You can generate a programming file with one, two, or all of the silicon features from the Programming File Generator first page.

To generate a programming file:

1. Select the **Silicon feature(s)** you want to program.

- Security settings
- FPGA Array
- FlashROM



FlashPoint - Programming File Generator - Step 1 of 2

Silicon feature(s) to be programmed:

☐ Security settings

☑ FPGA Array

☑ FlashROM

Original FlashROM configuration file:

P:\A3PEQUAL\A3P125_BI\smartgen\UFROMFIX2\UFROMFIX2.ufc

☐ Programming previously secured device(s)

Silicon signature (max length is 8 HEX chars):

Figure 60 · Programming File Generator – Step 1 of 2

Note: Note: When FlashPoint is invoked for the first time, after netlist files are imported and the design is in post-layout state, the software retrieves the FlashROM and EFM blocks configuration files from the imported netlists and imports the configuration files. Otherwise, you need to import configuration files.

2. Click the **Programming previously secured device(s)** check box if you are reprogramming a device that has been secured.

Because the IGLOO, ProASIC3, SmartFusion, Fusion families enable you to program the Security Settings separately from the FPGA Array and/or FlashROM, you must indicate if the Security Settings were previously programmed into the target device. This requirement also applies when you generate programming files for reprogramming.

3. Enter the silicon signature (0-8 HEX characters). See Silicon Signature for more information.

4. Depending upon the Silicon features you selected, click **Next** or **Finish**.

If you click **Next**, follow the instructions in the appropriate dialog box. If you click **Finish**, the **Generate Programming Files** dialog box appears (as shown in the figure below). Use this dialog box box to specify the programming file name, location, output format (STAPL file, SVF file, PDB file, DirectC DAT file, 1532 file), and, if necessary, limit the file size (as explained below).

Some testers may have memory size restrictions for a single SVF file. The SVF limit file option enables you to limit the size of each SVF file by either file size or vectors.

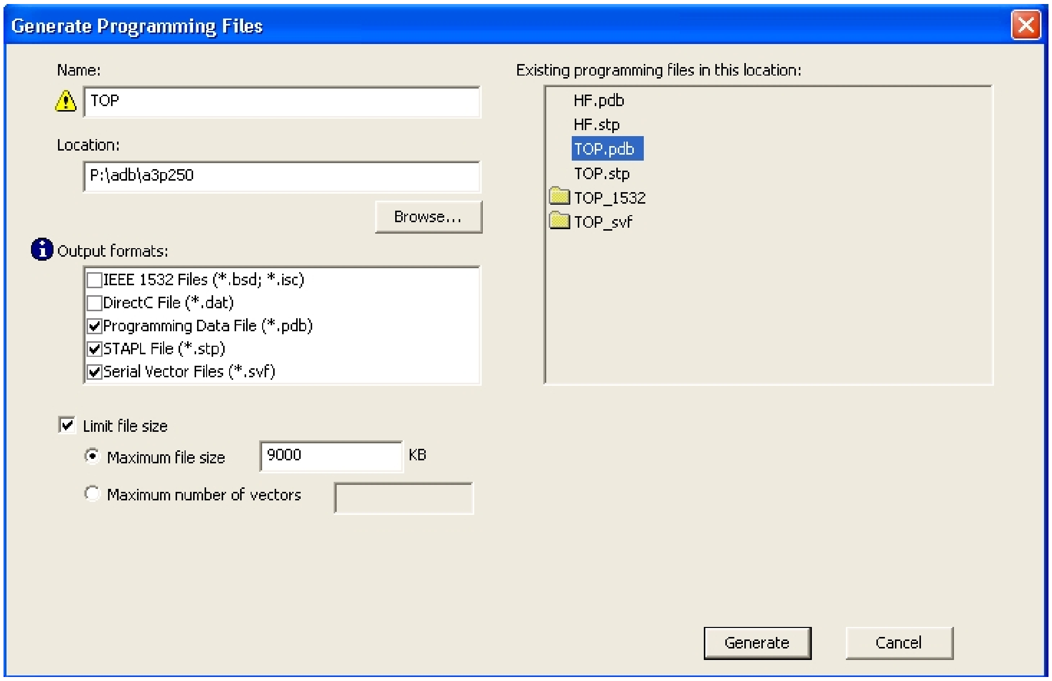
The generated SVF files append an index to the file name indicating the sequence of files. The format is:

```
<SVF_filename>_XXXXX.svf
```

where XXXXX is the index of the SVF file. The first SVF file begins with <SVF_filename>_00000.svf and increments by 1 until file generation is complete.

Maximum file size: Max file size limit for the SVF file; use this option to limit your SVF file size based on number of kB.

Maximum number of vectors: Max vector limit for the SVF file; use this option to limit the size of your SVF based on number of vectors.



Generate Programming Files

Name: TOP

Location: P:\adb\a3p250

Output formats:

- ☐ IEEE 1532 Files (*.bsd; *.isc)
- ☐ DirectC File (*.dat)
- ☑ Programming Data File (*.pdb)
- ☑ STAPL File (*.stp)
- ☑ Serial Vector Files (*.svf)

Existing programming files in this location:

HF.pdb
HF.stp
TOP.pdb
TOP.stp
TOP_1532
TOP_svf

☑ Limit file size

● Maximum file size 9000 KB

○ Maximum number of vectors

Figure 61 · Generate Programming Files Dialog Box (Flashpoint)

Programming File Types

The table below summarizes the Microsemi SoC programming file types and programmers.

Unless otherwise noted, listing an individual device indicates the device family and all its derivatives. For example, IGLOO indicates IGLOO, IGLOOe, IGLOO nano and IGLOO plus. See the Supported Families topic for more information. See the list of programming file type descriptions below for more details.

Programming File Type	Device Support	Programmer
PDB (*.pdb)	See device specifications	FlashPro 4/3/3x
STAPL (*.stp)		FlashPro 4/3/3x, FlashPro Lite, FlashPro,

Programming File Type	Device Support	Programmer
		Silicon Sculptor III/II
SVF (*.svf)		Third party programmer
IEEE 1532 (*.isc or *.bsd)		Third party programmer

The following programming-related files are required if you use the related functional block elements in your enabled devices. See the appropriate sections of the FlashPro help for more information on creating these files.

File Type	Device Support	Function
FDB (*.fdb)	See device specifications	Contains your FPGA array data
UFC (*.ufc)		Contains your FlashROM data
EFC (*.efc)		Contains your Embedded Flash Memory file

PDB Files

A proprietary Microsemi and Actel programming data file.

STAPL Files

The Standard Test And Programming Language (STAPL) is designed to support the programming of programmable devices and testing of electronic systems, using the IEEE Standard 1149.1: “Standard Test Access Port and Boundary Scan Architecture” (commonly referred to as JTAG) interface. As a STAPL file is executed, signals are produced on the IEEE 1149.1 interface, as described in the STAPL file. STAPL operates on a single IEEE 1149.1 chain. STAPL supports the programming of any IEEE 1149.1-compliant programmable device.

STAPL has support for programming and test systems with user interface features. A single STAPL file may perform several different functions, such as programming, verifying, and erasing a programmable device.

Bitstream Files

Proprietary Microsemi and Actel programming data file.

SVF Files

Courtesy Serial Vector Format Specification from ASSET InterTech, 1999:

Serial Vector Format (SVF) is the media for exchanging descriptions of high-level IEEE 1149.1 bus operations. In general, IEEE 1149.1 bus operations consist of scan operations and movements between different stable states on the IEEE 1149.1 state diagram. SVF does not explicitly describe the state of the IEEE 1149.1 bus at every Test Clock.

The SFV file is defined as an ASCII file that consists of a set of SVF statements. The maximum number of characters on a line is 256, although one SVF statement can span more htan one line. Each statement consists of a command and associated parameters. Each SVF statement is terminated by a semicolon. SVF is not case sensitive.

IEEE 1532 Files

Courtesy ieee.org:

The IEEE 1532 files implement programming capabilities within programmable integrated circuit devices, utilizing (and compatible with) the 1149.1 communication protocol. This standard allows the programming of one or more compliant devices concurrently, while mounted on a board or embedded in a system, known as In-System Configuration.

Generate a Programming File for SmartFusion

You can configure and generate a new PDB file from FlashPoint.

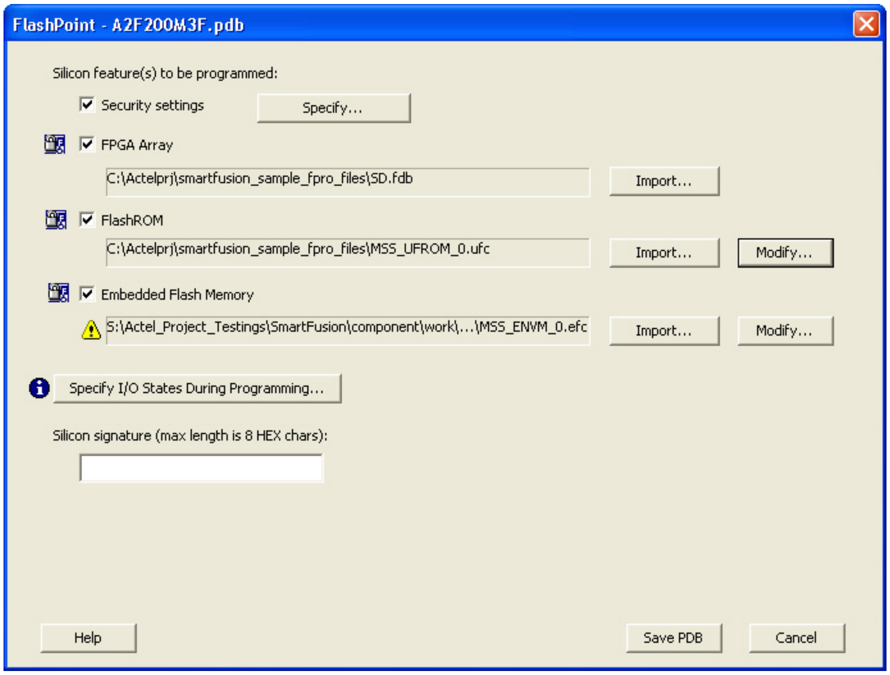
If you are using Single Mode, click **Create** to add a new PDB, or click **Modify** to make changes to a loaded PDB.

In Chain Mode, if you have not already done so, construct a chain and click **Create PDB** to create a new PDB for programming, or click **Modify PDB** to make changes to a loaded PDB.

FlashPoint enables you to specify your security settings and silicon features when you generate your programming file in SmartFusion. You can specify your FPGA Array, FlashROM, and Embedded Flash Memory by importing FDB, UFC and EFC files, respectively (as shown in the figure below). If you have imported a FlashROM and Embedded Flash Memory file you can click **Modify** to configure these feature before saving your PDB file.

Click Specify I/O States During Programming to set custom I/O states.

Note: NOTE: You must import an FDB to populate Port Name and Macro Cell columns.



FlashPoint - A2F200M3F.pdb

Silicon feature(s) to be programmed:

☑ Security settings

☑ FPGA Array

C:\Actelprj\smartfusion_sample_fpro_files\SD.fdb

☑ FlashROM

C:\Actelprj\smartfusion_sample_fpro_files\MSS_UFROM_0.ufc

☑ Embedded Flash Memory

S:\Actel_Project_Testings\SmartFusion\component\work\...\MSS_ENVM_0.efc

Silicon signature (max length is 8 HEX chars):

Figure 62 · FlashPoint Programming Settings for SmartFusion

Generate a Programming File for CoreMP7/Cortex-M1 Device Support

FlashPoint enables you to program FPGA Array and FlashROM features for CoreMP7/Cortex-M1 devices. You can program these features separately using different programming files or you can combine them into one programming file. Each feature is listed as a silicon feature in the GUI. You can generate a programming file with one, two, or all of the silicon features from the **Programming File Generator** first page. For CoreMP7/Cortex-M1 device support, you cannot select your own security settings. The generated programming file always has the encrypted FPGA Array content. The programming file generation is the same as the ProASIC3 family devices.

To generate a programming file:

1. Select the Silicon feature(s) you want to program.

FPGA Array

FlashROM

2. Click **Next** or **Finished** depending on the silicon features you selected.

If you click **Next**, follow the instructions in the appropriate dialog box. If you click **Finish**, the **Generate Programming Files** dialog box appears. Use this dialog box box to specify the programming file name, location, and output format (STAPL file, SVF file, PDB file, DirectC DAT file, 1532 file).

For more information on DAT files, refer to the Data File Generator (DatGen) section of the *DirectC User's Guide*.

CoreMP7/Cortex-M1 Device Security

CoreMP7/Cortex-M1 devices are shipped with the following security enabled:

- FPGA Array enabled for AES encrypted programming and verification.
- FlashROM enabled for plain text read and write.

You cannot select your own security settings. The generated programming file includes the encrypted FPGA Array content.

Programming FlashROM and FPGA Array

For CoreMP7/Cortex-M1 device support, the programming generation for FlashROM and FPGA Array is the same as the programming generation for ProASIC3 and ProASIC family devices.

Generate a Programming File for AFS Device Support - Designer Only

FlashPoint enables you to program Security Settings, FPGA Array, Embedded Flash Memory Blocks, and FlashROM features for AFS device support. You can program these features separately using different programming files or you can combine them into one programming file. Each feature is listed as a silicon feature in the GUI. You can generate a programming file with one, two, or all of the silicon features from the **Programming File Generator** first page.

AFS Programming

In addition to FPGA Array, FlashROM and security setting, the Fusion devices provide Embedded Flash Memory

Blocks (FB) for both Analog configuration initialization and regular memory storage. Depending on the targeted AFS device, you may have one, two, or four FBs available to you. FlashPoint enables you to initialize the FB Instance(s), as desribed in the Embedded Flash Memory help.

To generate a programming file:

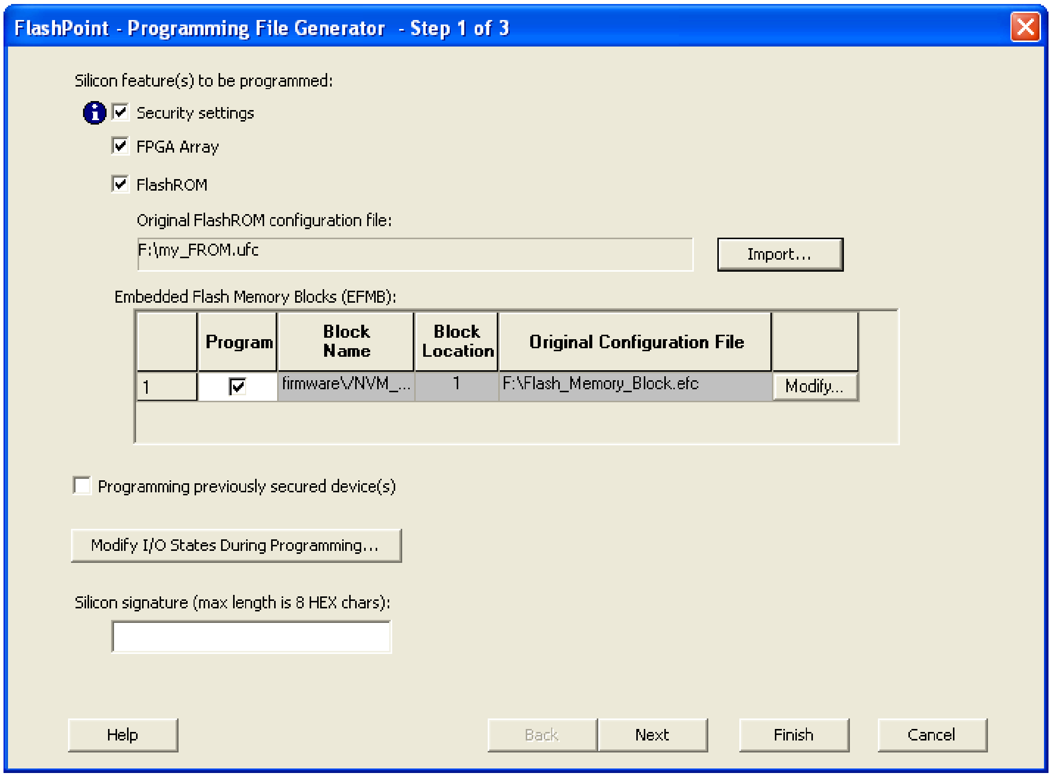
1. Select the **Silicon feature(s)** you want to program.

Security Settings

FPGA Array

FlashROM

Embedded Flash Memory Block



FlashPoint - Programming File Generator - Step 1 of 3

Silicon feature(s) to be programmed:

☑ Security settings

☑ FPGA Array

☑ FlashROM

Original FlashROM configuration file:

F:\my_FROM.ufc

Embedded Flash Memory Blocks (EFMB):

	Program	Block Name	Block Location	Original Configuration File	
1	☑	firmware\NVM_...	1	F:\Flash_Memory_Block.efc	Modify...

☐ Programming previously secured device(s)

Silicon signature (max length is 8 HEX chars):

Figure 63 · FlashPoint- Programming File Generator for AFS

Note: Note: Check the check box in the Program column to enable block modification.

2. Check the **Programming previously secured devices(s)** box if you want to program previously secured devices.
3. Enter the **Silicon signature**.
4. Depending upon the Silicon features you selected, click **Finish** or **Next**.

If you click **Next**, follow the instructions in the appropriate dialog box. If you click **Finish**, the **Generate Programming Files** dialog box appears. Use this dialog box box to specify the programming file name, location, and output format (STAPL file, SVF file, PDB file, DirectC DAT file, 1532 file).

For more information on DAT files, refer to the Data File Generator (DatGen) section of the *DirectC User's Guide*.

Programming Security Settings, FlashROM, and FPGA Array

For AFS device support, the programming generation for Security Settings, FlashROM and FPGA Array is the same as the programming generation for ProASIC3 family devices.

Generate a Programming File for Serialization Support in In House Programming (IHP)

FlashPoint allows you to program security settings, FPGA Array, and FlashROM features for IGLOO, ProASIC3, SmartFusion, Fusion family devices. You can program these features separately using different programming files or you can combine them into one programming file. Each feature is listed as a silicon feature in the GUI.

SVF Serialization Support in IHP

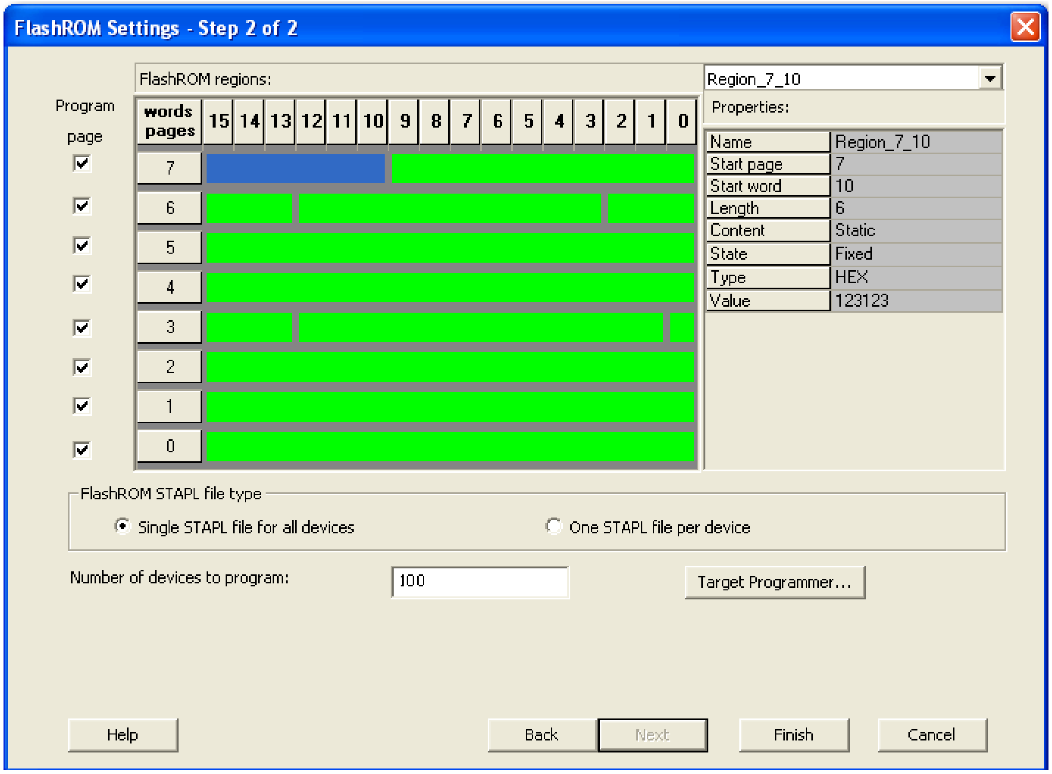
In addition to FPGA Array, FlashROM, and security setting, FlashPoint supports generating SVF files with serialization support in IHP.

To generate SVF with serialization support:

1. Select the **Silicon feature(s)** you want to program.

- Security settings
- FPGA Array
- FlashROM
- Programming Embedded Flash Memory Block

2. Import the UFC file which contains serialization data to FlashROM. Click **Next**.
3. Type in the number of devices to program (as shown in the figure below).



FlashROM Settings - Step 2 of 2

FlashROM regions: Region_7_10

Program page: ☑ 7, ☑ 6, ☑ 5, ☑ 4, ☑ 3, ☑ 2, ☑ 1, ☑ 0

Properties:

Name	Region_7_10
Start page	7
Start word	10
Length	6
Content	Static
State	Fixed
Type	HEX
Value	123123

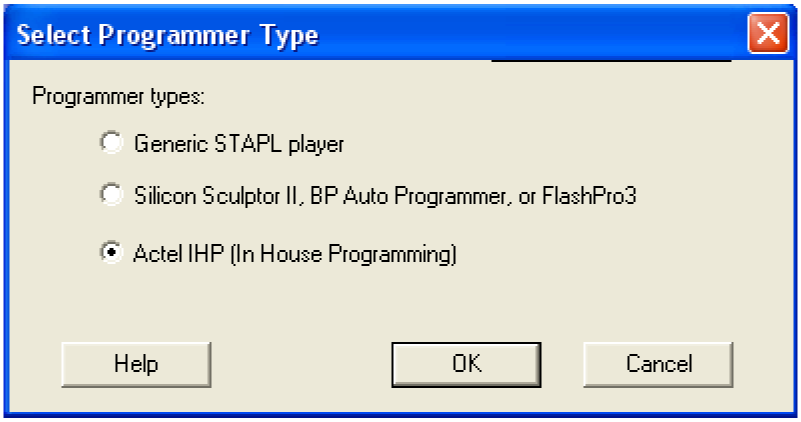
FlashROM STAPL file type

● Single STAPL file for all devices ○ One STAPL file per device

Number of devices to program: 100

Figure 64 · Type Number of Devices

4. Click **Target Programmer** and select **Actel IHP**.



Select Programmer Type

Programmer types:

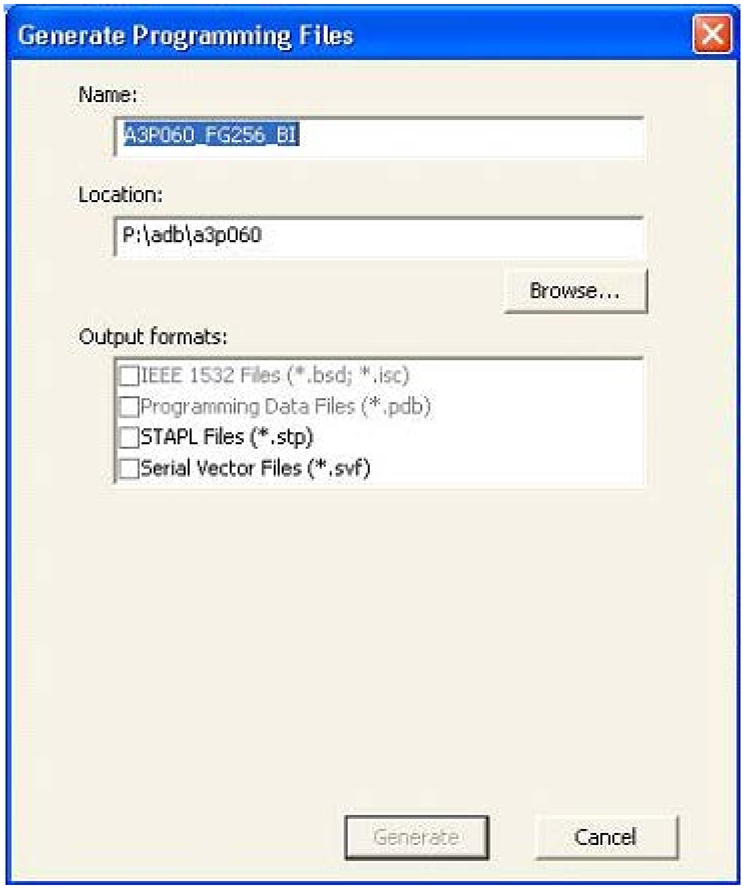
○ Generic STAPL player

○ Silicon Sculptor II, BP Auto Programmer, or FlashPro3

● Actel IHP (In House Programming)

Figure 65 · Select Actel IHP

5. Click **OK**. The Generate Programming Files window appears (as shown in the figure below). Select **Serial Vector Files (*.svf)**.



Generate Programming Files

Name: A3P060_FG256_BI

Location: P:\adb\a3p060

Output formats:

□ IEEE 1532 Files (*.bsd; *.isc)
□ Programming Data Files (*.pdb)
□ STAPL Files (*.stp)
□ Serial Vector Files (*.svf)

Select Serial Vector Files

6. Click **Generate**. An Actel-specific SVF file will be generated with a corresponding serialization data file.

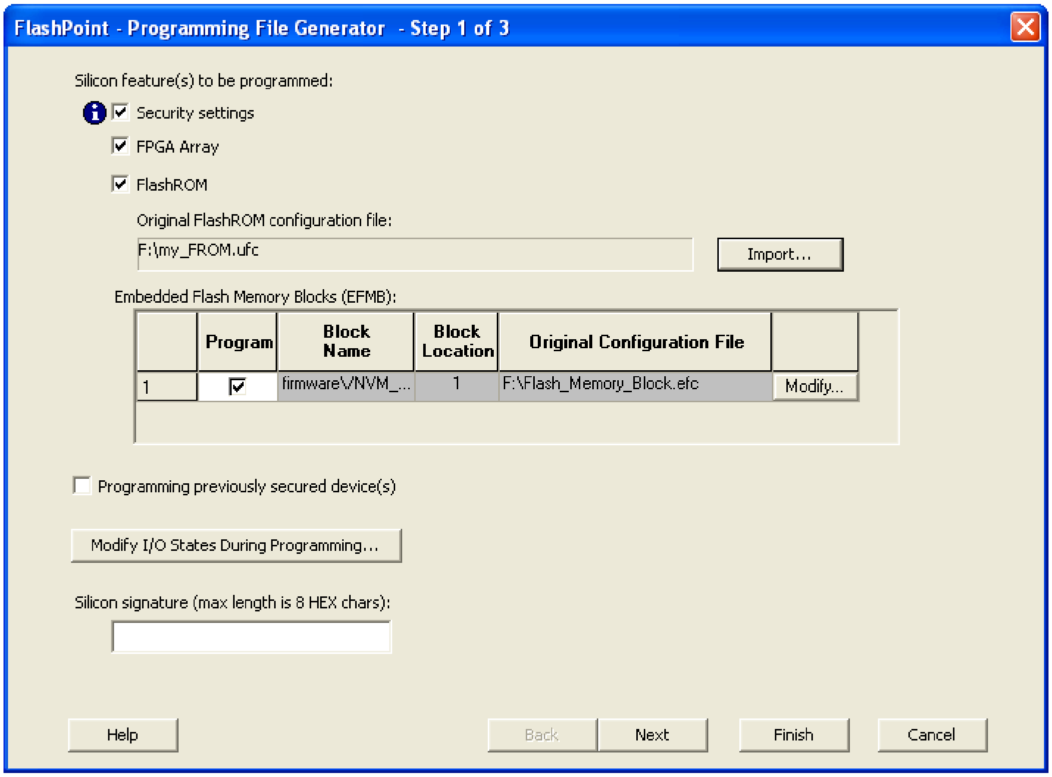
Note: Note: Generated SVF files will only work with IHP.

Creating a Programming Database (PDB) File in Designer

The programming database (PDB) file supports SmartFusion, IGLOO, ProASIC3 and Fusion devices only. This allows reconfiguration of the security settings, FlashROM, FPGA Array, and Embedded Flash Memory Blocks. You create the file in Designer using FlashPoint and you modify the file in FlashPro.

You must create programming files for SmartFusion in FlashPro; see the Generate a Programming File for SmartFusion topic for more information.

1. From the Designer main window, click the **Programming File** button. This brings up FlashPoint (see figure below).



FlashPoint - Programming File Generator - Step 1 of 3

Silicon feature(s) to be programmed:

☑ Security settings

☑ FPGA Array

☑ FlashROM

Original FlashROM configuration file: F:\my_FROM.ufc

Embedded Flash Memory Blocks (EFMB):

	Program	Block Name	Block Location	Original Configuration File	
1	☑	firmware\NVM_...	1	F:\Flash_Memory_Block.efc	Modify...

□ Programming previously secured device(s)

Silicon signature (max length is 8 HEX chars):

Figure 66 · FlashPoint Programming File Generator - PDB File

2. Select the silicon feature(s) to be programmed: Security Settings, FPGA array, FlashROM, and Embedded Flash Memory Block. If you are programming a previously secured device, check the Programming previously secured device(s) and enter the silicon signature.
3. Click **Finish** to create the PDB file.

See Also

Configuring security and FlashROM settings in FlashPro

Configuring security settings in FlashPro

Configuring FPGA array settings

Configuring FlashROM settings in FlashPro

Configuring Embedded Flash Memory Block settings in FlashPro

Programming Embedded Flash Memory Block

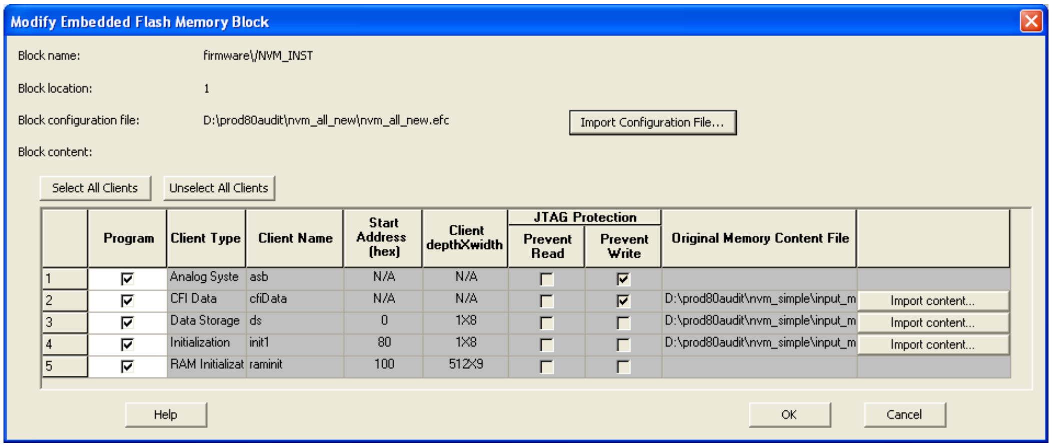
For more information about the Embedded Flash Memory Block, see the Flash Memory System Builder online help.

To program the Embedded Flash Memory Block:

1. Check the **Program** box to enable Embedded Flash Memory Block modification.

2. Click the **Modify** button to import Embedded Flash Memory Block configuration and memory content.

The **Modify Embedded Flash Memory Block** dialog box appears.



Modify Embedded Flash Memory Block

Block name: firmware\/NVM_INST

Block location: 1

Block configuration file: D:\prod80audit\nvm_all_new\nvm_all_new.efc

Block content:

	Program	Client Type	Client Name	Start Address (hex)	Client depthXwidth	JTAG Protection		Original Memory Content File	
						Prevent Read	Prevent Write		
1	☑	Analog Syste	asb	N/A	N/A	☐	☑		
2	☑	CFI Data	cfiData	N/A	N/A	☐	☑	D:\prod80audit\nvm_simple\input_m	Import content...
3	☑	Data Storage	ds	0	1X8	☐	☐	D:\prod80audit\nvm_simple\input_m	Import content...
4	☑	Initialization	init1	80	1X8	☐	☐	D:\prod80audit\nvm_simple\input_m	Import content...
5	☑	RAM Initializat	raminit	100	512X9	☐	☐		

Figure 67 · Modify Embedded Flash Memory Block Content Dialog Box

3. Click the **Import Configuration File** button (if available) to import the Embedded Flash Memory Block configuration and memory content from the EFC file. This will populate the client table below. All clients that belong to this block will be selected by default.
4. Click the **Import content** button if you want to change the client memory content.
5. Click **OK**.

Note: Note: FlashPoint audits original configuration and memory content files and warns you if the files cannot be located or if they have been updated.

Programming the FlashROM

You can program selected memory pages and specify the region values of the FlashROM.

- **Single STAPL file for all devices**: generates one programming file with all the generated increment values or with values in the custom serialization file.
- **One STAPL file per device**: generates one programming file for each generated increment value or for each value in the custom serialization file.

1. Select your target **Programmer type**.

   • Select Generic STAPL Player when generating STAPL files for generic STAPL players.

   • Select Silicon Sculptor II, BP Auto Programmer, or FlashPro4/3x/3 when generating programming files for those programmers.

   • Select Actel IHP (In House Programming) when generating STAPL or SVF files for Microsemi SoC (formerly Actel) IHP.

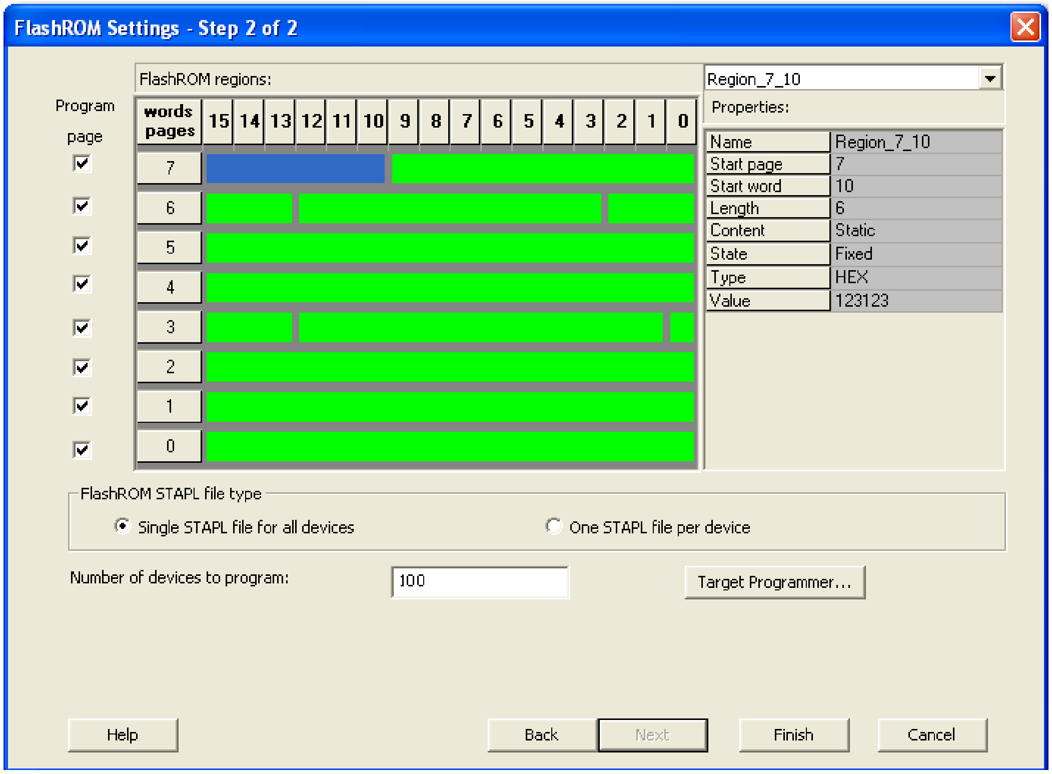
2. Click **OK**.
   FlashPoint generates your programming file.

Note: Note: You cannot change the FlashROM region configuration from FlashPoint. You can only change the configuration from the FlashROM core generator.

For more information, click the Help button in FlashROM.

To program FlashROM:

1. Select **FlashROM** from the **Generate Programming File** page.
2. Enter the location of the FlashROM configuration file. The **FlashROM Settings** page appears (see figure below).



FlashROM Settings - Step 2 of 2

FlashROM regions: Region_7_10

Properties:

Name	Region_7_10
Start page	7
Start word	10
Length	6
Content	Static
State	Fixed
Type	HEX
Value	123123

FlashROM STAPL file type

● Single STAPL file for all devices ○ One STAPL file per device

Number of devices to program: 100

Figure 68 · FlashROM Settings

3. Select the FlashROM memory page that you want to program.

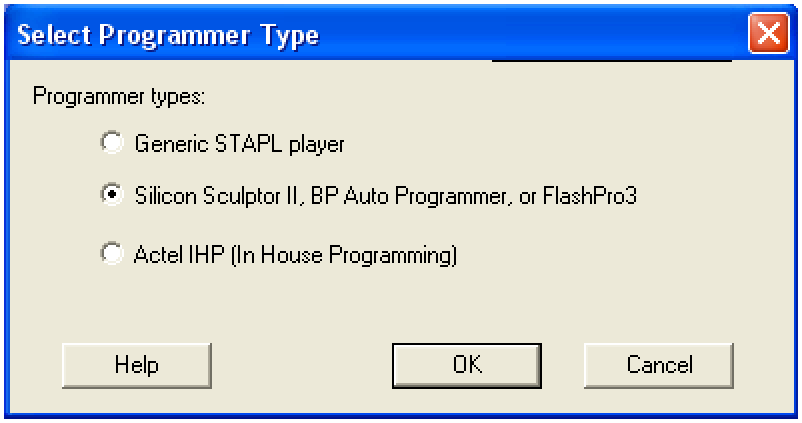
4. Enter the data value for the configured regions.

5. If you selected the region with a **Read From File**, specify the file location.

6. If you selected the **Auto Increment** region, specify the **Start** and **Max** values.

Enter the number of devices you want to program.

Select your target Programmer Type.



Select Programmer Type

Programmer types:

○ Generic STAPL player

● Silicon Sculptor II, BP Auto Programmer, or FlashPro3

○ Actel IHP (In House Programming)

Select Programmer

7. Click **Finish**.
FlashPoint generates your programming file.

Note: Note: You cannot change the FlashROM region configuration from FlashPoint. You can only change the configuration from the FlashROM core generator.

Silicon Signature

With Libero SoC tools, you can use the silicon signature to identify and track Microsemi designs and devices. When you generate a programming file, you can specify a unique silicon signature to program into the device. This signature is stored in the design database and in the programming file, and programmed into the device during programming.

The silicon signature is accessible through the USERCODE JTAG instruction.

Note: Note: If you set the security level to high, medium, or custom, you must program the silicon signature along with the Security Setting. If you have already programmed the Security Setting into the target device, you cannot reprogram the silicon signature without reprogramming the Security Setting.

Note: The previously programmed silicon signature will be erased if:

- You have already programmed the silicon signature and
- You are programming the security settings, but you do not have an entry in the silicon signature field

Programming Security Settings

FlashPoint allows you to set a security level of high, medium, or none (SmartFusion uses radio buttons and the option Clear Security instead of None).

To program Security Settings on the device:

1. If you choose to program Security Settings on the device from the **Generate Programming File** page, the wizard takes you to the **Security Settings** page.

   Your Security Settings page depends on your family.
2. Set the security level for FPGA and FlashROM (see the table below for a description of the security levels).

Table 7 · FPGA and FROM Security Settings

Security Level	Security Option	Description
High	Protect with a 128-bit Advanced Encryption Standard (AES) key and a Pass Key	Access to the device is protected by an AES Key and the Pass Key. The Write and Verify operations of the FPGA Array use a 128-bit AES encrypted bitstream. From the JTAG interface, the Write and Verify operations of the FlashROM use a 128-bit AES encrypted bitstream. Read back of the FlashROM content via the JTAG interface is protected by the Pass Key. Read back of the FlashROM content is allowed from the FPGA Array.
Medium	Protect with Pass Key	The Write and Verify operations of the FPGA Array require a Pass Key. From the JTAG interface, the Read and Write operations on the FlashROM content require a Pass Key. You can Verify the FlashROM content via the JTAG interface without a Pass Key. Read back of the FlashROM content is allowed from the FPGA Array.
None	No security	The Write and Verify operations of the FPGA Array do not require keys. The Read, Write, and Verify operations of the FlashROM content also do not require keys.

Security Level	Security Option	Description
		This option is available for SmartFusion; to choose it, de-select the Security Settings checkbox.

Note: Note: When a Device is programmed with a Pass key and AES key, only the Pass key is required for reprogramming since re-entering the correct Pass key unlocks the bits that restrict programming to require AES encryption and also unlocks the bits that prohibit reprogramming altogether (if locked); thus both plaintext and encrypted programming are [re-] enabled.

3. **Enable eNVM client JTAG protection** - Enables eNVM client JTAG protection in the event you have not set Medium or High security. Enables you to protect specific clients with a user pass key and then leave others unprotected. This can be advantageous if you want to protect your IP, but give another user access to the rest of the eNVM for storage. You can also set custom security levels for your eNVM.

4. Enter the **Pass Key** and/ or the **AES Key** as appropriate. You can generate a random key by clicking the **Generate random key** button.

The **Pass Key** protects all the Security Settings for the FPGA Array and/or FlashROM.

The **AES Key** decrypts FPGA Array and/or FlashROM programming file content. Use the AES Key if you intend to program the device at an unsecured site or if you plan to update the design at a remote site in the future.

You can also customize the security levels by clicking the **Custom Level** button. For more information, see the Custom Security Levels section.

To change or disable your security keys you must run the ERASE_SECURITY action code. This erases your security settings and enables you to generate the programming file with new keys and reprogram, or to generate a programming file that has no security key.

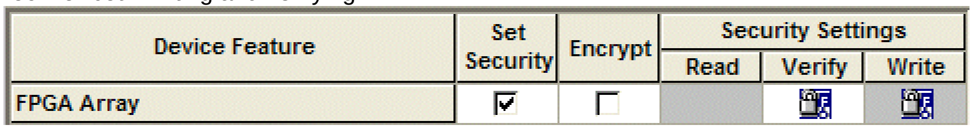
Custom Security Levels

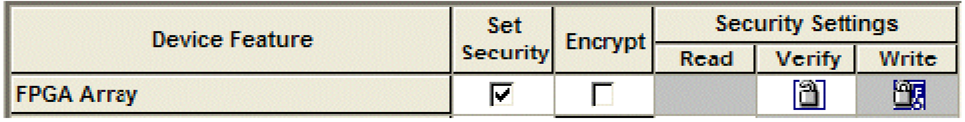
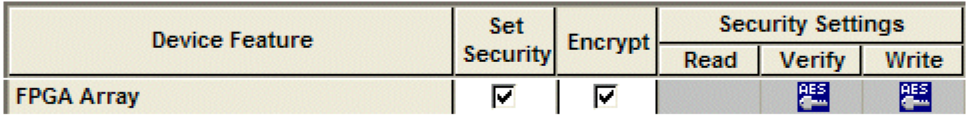
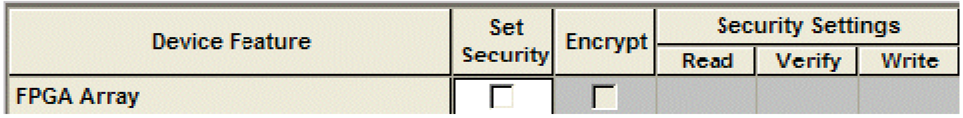
For advanced use, you can customize your security levels.

To set custom security levels:

1. Click the **Custom Level** button in the **Security Settings** page. The **Custom Security Level** dialog box appears.
2. Select the **FPGA Array Security** and the **FlashROM Security** levels. ForSmartFusion and Fusion devices, you can also choose the Embedded Flash Memory Block level of security. The FPGA Array and the FlashROM can have different Security Settings. See the tables below for a description of the custom security option levels for FPGA Array and FlashROM.

Table 8 · FPGA Array

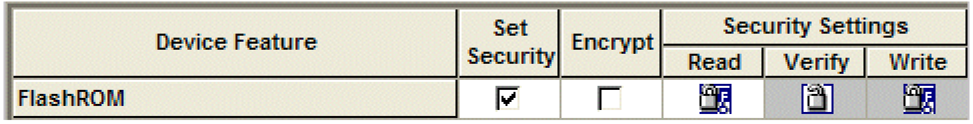
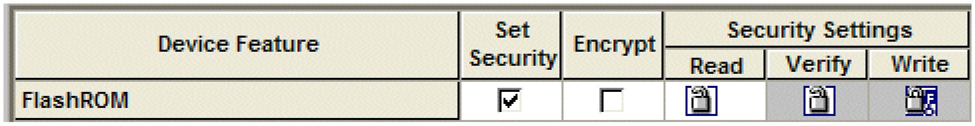
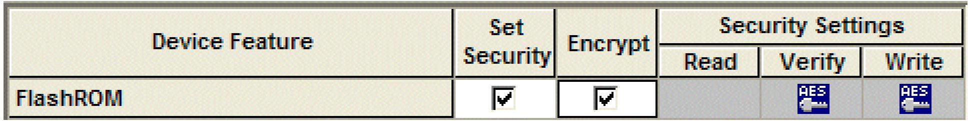
<table>
<tr><th>Security Option</th><th>Description</th></tr>
<tr><td>Lock for both writing and verifying

<table>
<tr><th rowspan="2">Device Feature</th><th rowspan="2">Set Security</th><th rowspan="2">Encrypt</th><th colspan="3">Security Settings</th></tr>
<tr><th>Read</th><th>Verify</th><th>Write</th></tr>
<tr><td>FPGA Array</td><td>☑</td><td>□</td><td></td><td>🔒</td><td>🔒</td></tr>
</table></td><td>Allows writing/erasing and verification of the FPGA Array via the JTAG interface</td></tr>
</table>

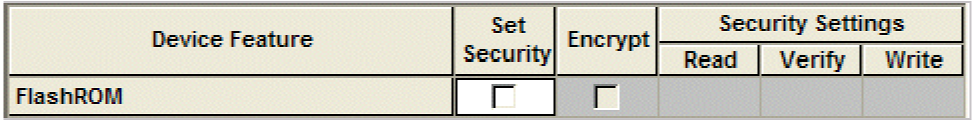
Security Option	Description
	only with a valid Pass Key.
Lock for writing  Device Feature | Set Security | Encrypt | Security Settings: Read | Verify | Write FPGA Array | ☑ | ☐ | | 🔒 | 🔒	Allows the writing/erasing of the FPGA Array only with a valid Pass Key. Verification is allowed without a valid Pass Key.
Use the AES Key for both writing and verifying  Device Feature | Set Security | Encrypt | Security Settings: Read | Verify | Write FPGA Array | ☑ | ☑ | | AES | AES	Allows the writing/erasing and verification of the FPGA Array only with a valid AES Key via the JTAG interface. This configures the device to accept an encrypted bitstream for reprogramming and verification of the FPGA Array. Use this option if you intend to complete final programming at an unsecured site or if you plan to update the design at a remote site in the future. Accessing the device security settings requires a valid Pass Key.
Allow write and verify  Device Feature | Set Security | Encrypt | Security Settings: Read | Verify | Write FPGA Array | ☐ | ☐ | | |	Allows writing/erasing and verification of the FPGA Array with plain text

Security Option	Description
	bitstream and without requiring a Pass Key or an AES Key. Use this option when you develop your product in-house.

Note: Note: The ProASIC3 family FPGA Array is always read protected regardless of the Pass Key or the AES Key protection.

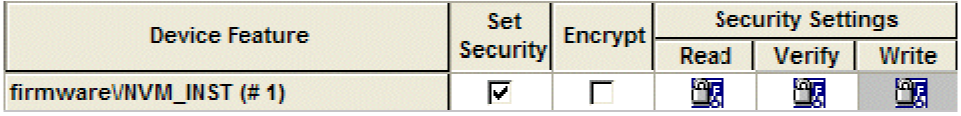
Table 9 · FlashROM

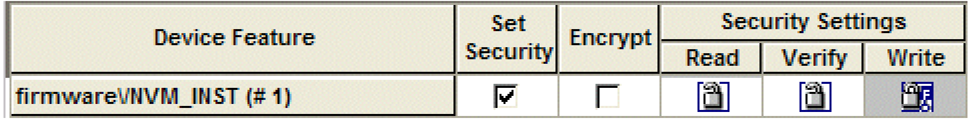
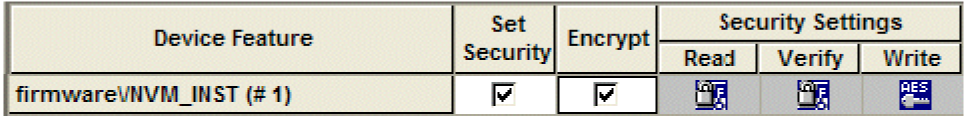
<table>
<tr><th>Security Option</th><th>Description</th></tr>
<tr><td>Lock for both reading and writing
<table><tr><th rowspan="2">Device Feature</th><th rowspan="2">Set Security</th><th rowspan="2">Encrypt</th><th colspan="3">Security Settings</th></tr><tr><th>Read</th><th>Verify</th><th>Write</th></tr><tr><td>FlashROM</td><td>☑</td><td>□</td><td>🔒</td><td>🔒</td><td>🔒</td></tr></table></td><td>Allows the writing/erasing and reading of the FlashROM via the JTAG interface only with a valid Pass Key. Verification is allowed without a valid Pass Key.</td></tr>
<tr><td>Lock for writing
<table><tr><th rowspan="2">Device Feature</th><th rowspan="2">Set Security</th><th rowspan="2">Encrypt</th><th colspan="3">Security Settings</th></tr><tr><th>Read</th><th>Verify</th><th>Write</th></tr><tr><td>FlashROM</td><td>☑</td><td>□</td><td>🔒</td><td>🔒</td><td>🔒</td></tr></table></td><td>Allows the writing/erasing of the FlashROM via the JTAG interface only with a valid Pass Key. Reading and verification is allowed without a valid Pass Key.</td></tr>
<tr><td>Use the AES Key for both writing and verifying
<table><tr><th rowspan="2">Device Feature</th><th rowspan="2">Set Security</th><th rowspan="2">Encrypt</th><th colspan="3">Security Settings</th></tr><tr><th>Read</th><th>Verify</th><th>Write</th></tr><tr><td>FlashROM</td><td>☑</td><td>☑</td><td></td><td>AES</td><td>AES</td></tr></table></td><td>Allows the writing/erasing and verification of the FlashROM via the JTAG interface only with a valid AES Key. This configures the device to</td></tr>
</table>

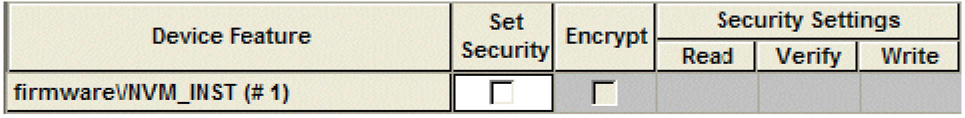
<table>
<tr><th>Security Option</th><th>Description</th></tr>
<tr><td></td><td>accept an encrypted bitstream for reprogramming and verification of the FlashROM. Use this option if you complete final programming at an unsecured site or if you plan to update the design at a remote site in the future. Note: The bitstream that is read back from the FlashROM is always unencrypted (plain text).</td></tr>
<tr><td>Allow reading, writing, and verifying

<table>
<tr><th>Device Feature</th><th>Set Security</th><th>Encrypt</th><th>Security Settings</th><th></th><th></th></tr>
<tr><td></td><td></td><td></td><td>Read</td><td>Verify</td><td>Write</td></tr>
<tr><td>FlashROM</td><td>☐</td><td>☐</td><td></td><td></td><td></td></tr>
</table></td><td>Allows writing/erasing, reading and verification of the FlashROM content with a plain text bitstream and without requiring a valid Pass Key or an AES Key.</td></tr>
</table>

Note: The FPGA Array can always read the FlashROM content regardless of these Security Settings.

Table 10 · Embedded Flash Memory Block

<table>
<tr><th>Security Option</th><th>Description</th></tr>
<tr><td>Lock for reading, verifying, and writing

<table>
<tr><th>Device Feature</th><th>Set Security</th><th>Encrypt</th><th>Security Settings</th><th></th><th></th></tr>
<tr><td></td><td></td><td></td><td>Read</td><td>Verify</td><td>Write</td></tr>
<tr><td>firmware\NVM_INST (# 1)</td><td>☑</td><td>☐</td><td>🔒</td><td>🔒</td><td>🔒</td></tr>
</table></td><td>Allows the writing and reading of the Embedded Flash Memory Block via the JTAG interface only with a valid Pass</td></tr>
</table>

<table>
<tr><th>Security Option</th><th>Description</th></tr>
<tr><td></td><td>Key. Verification accomplished by reading back and compare.</td></tr>
<tr><td>Lock for writing

<table>
<tr><th rowspan="2">Device Feature</th><th rowspan="2">Set Security</th><th rowspan="2">Encrypt</th><th colspan="3">Security Settings</th></tr>
<tr><th>Read</th><th>Verify</th><th>Write</th></tr>
<tr><td>firmware\NVM_INST (# 1)</td><td>☑</td><td>□</td><td>🔓</td><td>🔓</td><td>🔒F</td></tr>
</table></td><td>Allows the writing of the Embedded Flash Memory Block via the JTAG interface only with a valid Pass Key. Reading and verification is allowed without a valid Pass Key.</td></tr>
<tr><td>Use AES Key for writing

<table>
<tr><th rowspan="2">Device Feature</th><th rowspan="2">Set Security</th><th rowspan="2">Encrypt</th><th colspan="3">Security Settings</th></tr>
<tr><th>Read</th><th>Verify</th><th>Write</th></tr>
<tr><td>firmware\NVM_INST (# 1)</td><td>☑</td><td>☑</td><td>🔒F</td><td>🔒F</td><td>AES</td></tr>
</table></td><td>Allows the writing of the Embedded Flash Memory Block via the JTAG interface only with a valid AES Key. This configures the device to accept an encrypted bitstream for reprogramming of the Embedded Flash Block. Use this option if you complete final programming at an unsecured site or if you plan to update the design at a remote site in the future. The bitstream that is read back from the Embedded Flash Memory Block is always unencrypted</td></tr>
</table>

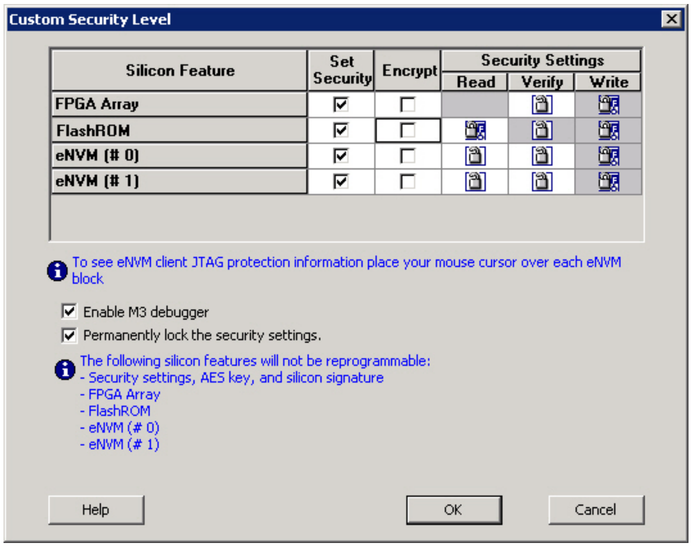
Security Option	Description
	(plain text), when a valid pass key is provided.
Allow reading, writing, and verifying  Device Feature · Set Security · Encrypt · Security Settings (Read, Verify, Write) firmwareVNVM_INST (# 1) · □ · □	Allows writing, reading and verification of the Embedded Flash Memory Block content with a plain text bitstream and without requiring a valid Pass Key or an AES Key.

3. To make the Security Settings permanent, select **Permanently lock the security settings** check box. This option prevents any future modifications of the Security Setting of the device. A Pass Key is not required if you use this option.

   Note: When you make the Security Settings permanent, you can never reprogram the Silicon Signature. If you Lock the write operation for the FPGA Array or the FlashROM, you can never reprogram the FPGA Array or the FlashROM, respectively. If you use an AES key, this key cannot be changed once you permanently lock the device.

4. (SmartFusion Only) Enable M3 Debugger option enables access to the M3 debugger even if security is enforced. Select the **Enable M3 debugger** checkbox if you want to access the M3 debugger after programming.

5. To use the Permanent FlashLock™ feature, select **Lock for both writing and verifying** for FPGA Array and **Lock for both reading and writing** for FlashROM and select the **Permanently lock the security settings** checkbox as shown in the figure below. This will make your device one-time-programmable.



Custom Security Level

Silicon Feature	Set Security	Encrypt	Security Settings		
			Read	Verify	Write
FPGA Array	☑	□		🔓	🔐
FlashROM	☑	□	🔐	🔓	🔐
eNVM (# 0)	☑	□	🔓	🔓	🔐
eNVM (# 1)	☑	□	🔓	🔓	🔐

To see eNVM client JTAG protection information place your mouse cursor over each eNVM block

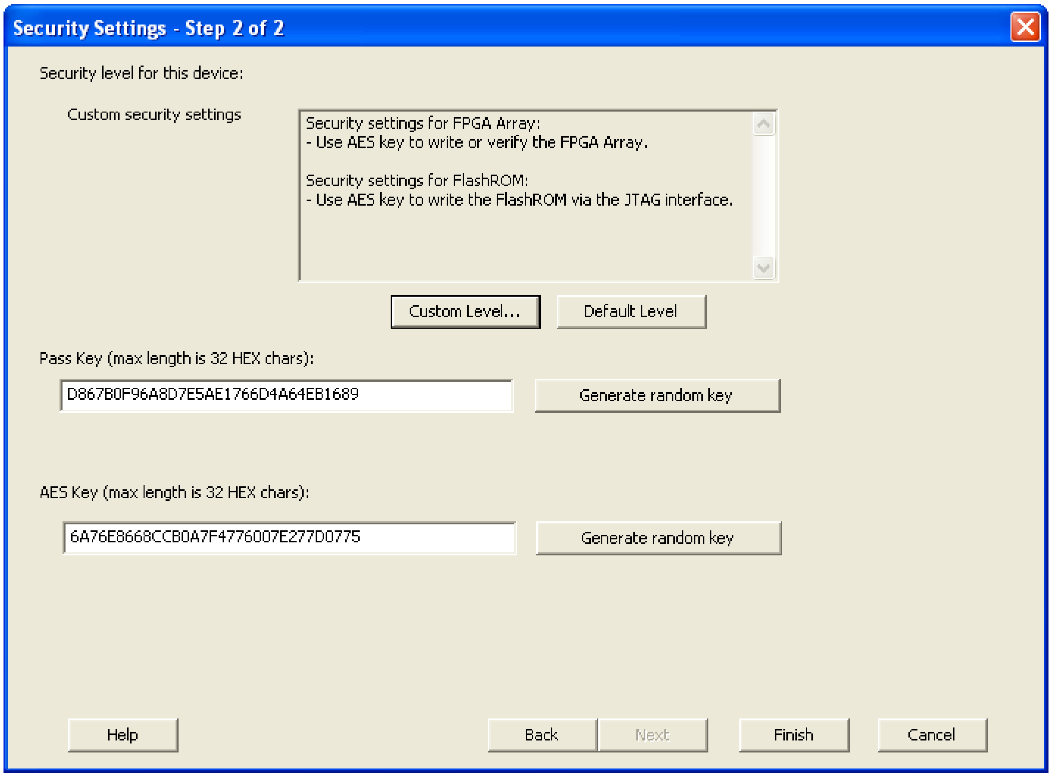
☑ Enable M3 debugger

☑ Permanently lock the security settings.

The following silicon features will not be reprogrammable:
- Security settings, AES key, and silicon signature
- FPGA Array
- FlashROM
- eNVM (# 0)
- eNVM (# 1)

Custom Security Level

6. Click the **OK** button. The **Security Settings** page appears with the **Custom security settings** information as shown in the figure below.



Security Settings - Step 2 of 2

Security level for this device:

Custom security settings

Security settings for FPGA Array:
- Use AES key to write or verify the FPGA Array.

Security settings for FlashROM:
- Use AES key to write the FlashROM via the JTAG interface.

Pass Key (max length is 32 HEX chars): D867B0F96A8D7E5AE1766D4A64EB1689

AES Key (max length is 32 HEX chars): 6A76E8668CCB0A7F4776007E277D0775

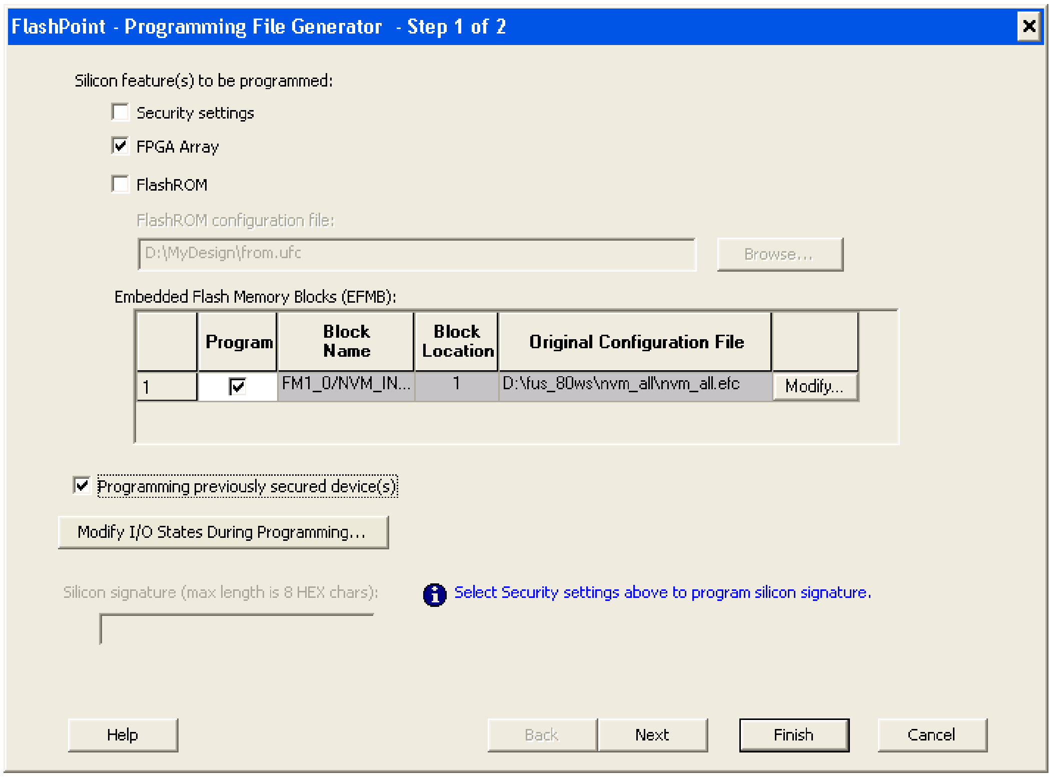
Figure 69 · Security Settings

Reprogramming a Secured Device

You must know the previous Security Settings of the device before you can reprogram a device with Security Settings.

To program a secured device:

1. In the Generate Programming File window, click the **Programming previously secured devices(s)** check box (see figure below).



FlashPoint - Programming File Generator - Step 1 of 2

Silicon feature(s) to be programmed:

□ Security settings

☑ FPGA Array

□ FlashROM

FlashROM configuration file: D:\MyDesign\from.ufc

Embedded Flash Memory Blocks (EFMB):

	Program	Block Name	Block Location	Original Configuration File	
1	☑	FM1_0/NVM_IN...	1	D:\fus_80ws\nvm_all\nvm_all.efc	Modify...

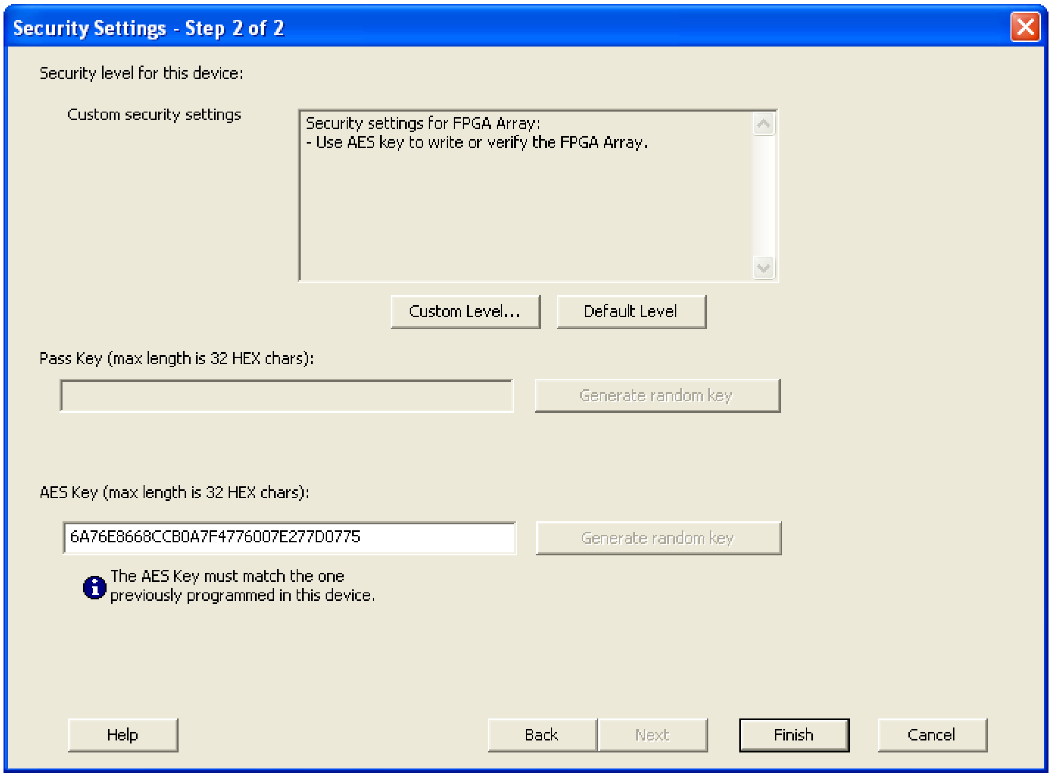
☑ Programming previously secured device(s)

Silicon signature (max length is 8 HEX chars):

Select Security settings above to program silicon signature.

Figure 70 · Generate Programming File

2. Specify the previously programmed security setting for the FlashROM and/or the FPGA Array. To generate a programming file for encrypted programming please ensure that the Security settings checkbox is unchecked.
3. If you programmed the device with a custom security level, click the **Custom Level** button to open the Custom security dialog box, and select the **Security Settings for the FPGA Array** or the FlashROM that you programmed (see figure below).



Security level for this device:

Custom security settings

Security settings for FPGA Array:
- Use AES key to write or verify the FPGA Array.

Pass Key (max length is 32 HEX chars):

AES Key (max length is 32 HEX chars): 6A76E8668CCB0A7F4776007E277D0775

The AES Key must match the one previously programmed in this device.

Figure 71 · Security Settings

4. Enter the previously programmed Pass Key and/or the AES Key.
5. Click **Finish**.

Note: Note: Enter the AES Key only if you want to perform encrypted programming.

Programming a Secured SmartFusion Device

After you create a PDB you may wish to export a programming file for a secured device. To do so:

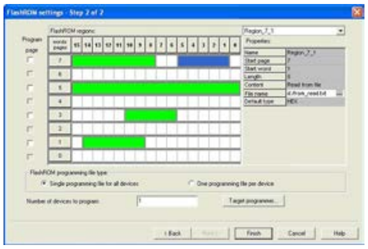
1. Create a PDB file (as explained above) with security set to **High** or **Medium**. Save the PDB file.
2. From the **File** menu, choose **Export Single Programming File**. The Export Programming Files dialog box appears.
3. Click the **Export programming file(s) for currently secured device** checkbox. This exports programming files for devices that already have security settings programmed.
4. Choose your outputs and enter your output file **Name** and **Location**.
5. Click **Export** to create the file(s). Your updated secured programming files are in the directory you specified.

Custom Serialization Data for FlashROM Region

FlashPoint enables you to specify a custom serialization file as a source to provide content for programming into a Read from file FlashROM region. You can use this feature for serializing the target device with a custom serialization scheme.

To specify a FlashROM region:

1. From the Properties section in the FlashROM Settings page, select the file name of the custom serialization file (see figure below). For more information on custom serialization files, see Custom Serialization Data File Format.



Name	Region_7_1
Start page	7
Start word	[illegible]
Length	[illegible]
Contents	Read from file
File name	d:/from_read.txt
Default type	HEX

FlashROM programming file type
● Single programming file for all devices ○ One programming file per device

Number of devices to program: 1 Target programmer...

Figure 72 · FlashROM Settings

2. Select the FlashROM programming file type you want to generate from the two options below:

- Single programming file for all devices option: generates one programming file with all the values in the custom serialization file.
- One programming file per device: generates one programming file for each value in the custom serialization file.

3. Enter the number of devices you want to program.
4. Click the **Target Programmer** button.
5. Select your target Programmer type.
6. Click **OK**.

Custom Serialization Data File Format

FlashPoint supports custom serialization data files that specify the data in binary, HEX, decimal, or ASCII text. The custom serialization data files may contain multiple data with the Line Feed (LF) character as the delimiter. You can create a file by entering serialization data into any type of text editor. Depending on the serialization data format (hex, ASCII, binary, decimal), input the serialization data according to the size of the region you specified in the FlashROM settings page.

Semantics

Each custom serialization file has only one type of data format (binary, decimal, Hex or ASCII text). For example, if a file contains two different data formats (i.e. binary and decimal) it is considered an invalid file.

The length of each data file must be shorter or equal to the selected region length. If the data is shorter then the selected region length, the most significant bits shall be padded with 0’s. If the specified region length is longer then the selected region length, it is considered an invalid file.

The digit / character length is as follows:

```
-Binary digit: 1 bit
-Decimal digit: 4 bits
-Hex digit: 4 bits
-ASCII Character: 8 bits
```

Note: Note the standard example below:

If you wanted to use, for example, device serialization for three devices with serialization data 123, 321, and 456, you would create file name from_read.txt. Each line in from_read.txt corresponds to the serialization data that will be programmed on each device. For example, the first line corresponds to the first device to be programmed, the second line corresponds to the second device to be programmed, and so on.

Hex serialization data file example

The following example is a Hex serialization data file for a 40-bit region. Enter the serialization data below into file created by any text editor:

```
123AEd210
AeB1
0001242E
```

Note: Note: If you enter an invalid Hex digit such as 235SedF1, an error occurs. An error will also occur if you enter data that is out of range, i.e. 4300124EFE.

The following is an example of programming "AeB1" into Region_7_1 located on page 7, from Word 5 to Word 1 in the FlashROM settings page. See Custom serialization data for FlashROM region for more information.

	Table 15	...		Word 5	Word 4	Word 3	Word 2	Word 1	Word 0
Page 7	...	...	...	00	00	00	AE	B1	...

Binary serialization data file example

The following example is a binary serialization data file for a 16-bit region:

1100110011010001

```
100110011010011
11001100110101111 (This is an error: data out of range)
1001100110110111
1001100110110112 (This is an error: invalid binary digit)
```

Decimal serialization data file example

The following example is a decimal serialization data file for a 16-bit region:

```
65534
65535
65536 (This is an error: data out of range)
6553A (This is an error: invalid decimal digit)
```

Text serialization data file example

The following example is a text serialization data file for a 32-bit region:

```
AESB
A )e
ASE3 23 (This is an error: data out of range)
65A~
1234
AEbF
```

Syntax

Indentations in the syntax below indicate a wrapped line. If a line wraps and is not indented, then it should appear on one line; you may need to expand your help window to view the syntax correctly.

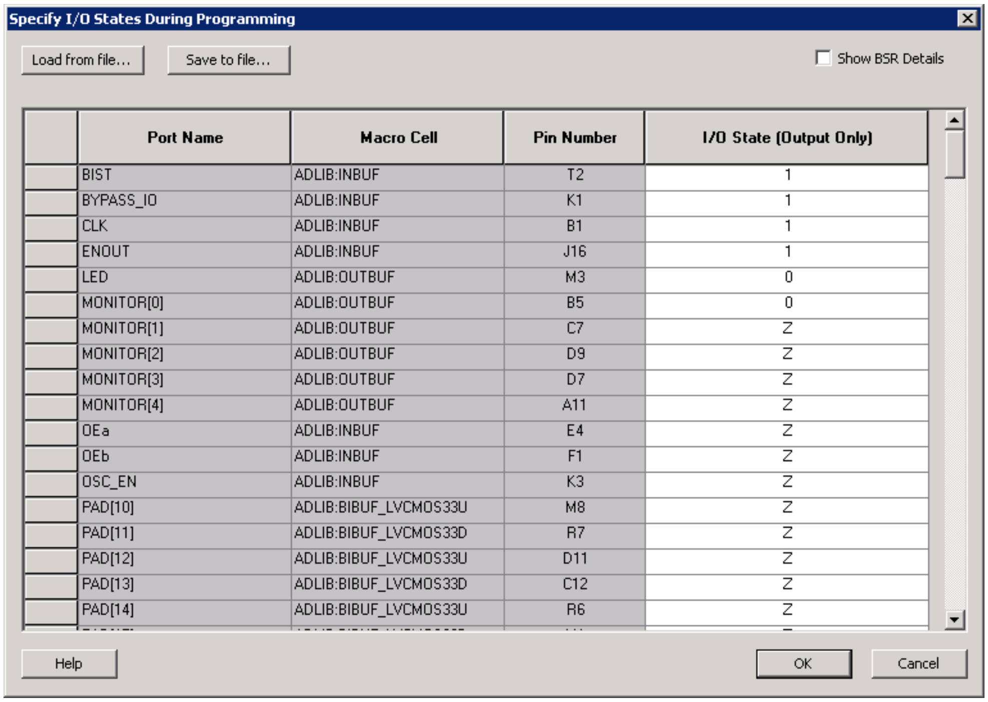
```
Custom serialization data file =
        <hex region data list> | <decimal region data list> |
        <binary region data list> | <ascii text data list>
Hex region data list = <hex data> <new line> { < hex data> <new line> }
Decimal region data list = <decimal data> <new line> {<decimal data><new line> }
Binary region data list = <binary data> <new line> { <binary data> <new line> }
ASCII text region data list = < ascii text data> <new line> { < ascii text data> <new
line> }
hex data = <hex digit> {<hex digit>}
decimal data = < decimal digit> {< decimal digit>}
binary data = < binary digit> {< binary digit>}
ASCII text data = <ascii character> {< ascii character >}
new line = LF
```

```
binary digit = ‘0’|‘1’
decimal digit = ‘0’|‘1’|‘2’|‘3’|‘4’|‘5’|‘6’|‘7’|‘8’| ‘9’
hex digit = ‘0’|‘1’|‘2’|‘3’|‘4’|‘5’|‘6’|‘7’|‘8’|‘9’|‘A’|‘B’|‘C’|‘D’ | ‘E’| ‘F’ |
        ‘a’| ‘b’ | ‘c’| ‘d’ | ‘e’| ‘f’
ascii character = characters from SP(0x20) to‘~’(0x7E).
```

Specifying I/O States During Programming

In Libero SoC, the I/O states can be set prior to programming, and held at the set values during programming. In Libero SoC, this feature is only available once layout is completed.

1. From the Designer GUI, click the **Modify I/O States During Programming** button. The Programming File Generator window appears.
2. Click the **Specify I/O States During Programming** button to display the Specify I/O States During Programming dialog box.
3. Sort the pins as desired by clicking any of the column headers to sort the entries by that header. Select the I/Os you wish to modify (as shown in the figure below).
4. Set the I/O Output state. You can set Basic I/O settings if you want to use the default I/O settings for your pins, or use Custom I/O settings to customize the settings for each pin. See the Specifying I/O States During Programming - I/O States and BSR Details help topic for more information on setting your I/O state and the corresponding pin values. Basic I/O state settings are:
   - 1 – I/O is set to drive out logic High
   - 0 – I/O is set to drive out logic Low
   - Last Known State: I/O is set to the last value that was driven out prior to entering the programming mode, and then held at that value during programming
   - Z - Tri-State: I/O is tristated



	Port Name	Macro Cell	Pin Number	I/O State (Output Only)
	BIST	ADLIB:INBUF	T2	1
	BYPASS_IO	ADLIB:INBUF	K1	1
	CLK	ADLIB:INBUF	B1	1
	ENOUT	ADLIB:INBUF	J16	1
	LED	ADLIB:OUTBUF	M3	0
	MONITOR[0]	ADLIB:OUTBUF	B5	0
	MONITOR[1]	ADLIB:OUTBUF	C7	Z
	MONITOR[2]	ADLIB:OUTBUF	D9	Z
	MONITOR[3]	ADLIB:OUTBUF	D7	Z
	MONITOR[4]	ADLIB:OUTBUF	A11	Z
	OEa	ADLIB:INBUF	E4	Z
	OEb	ADLIB:INBUF	F1	Z
	OSC_EN	ADLIB:INBUF	K3	Z
	PAD[10]	ADLIB:BIBUF_LVCMOS33U	M8	Z
	PAD[11]	ADLIB:BIBUF_LVCMOS33D	R7	Z
	PAD[12]	ADLIB:BIBUF_LVCMOS33U	D11	Z
	PAD[13]	ADLIB:BIBUF_LVCMOS33D	C12	Z
	PAD[14]	ADLIB:BIBUF_LVCMOS33U	R6	Z

Figure 73 · I/O States During Programming Window

6. Click **OK** to return to the FlashPoint – Programming File Generator window.

Note: NOTE: I/O States During programming are saved to the ADB and resulting programming files after completing programming file generation.

Custom I/O Settings and Boundary Scan Registers

Each I/O in your device is comprised of an Input, Output and Output Enable Boundary Scan Register (BSR) cell..

The BSR cells enable you to define I/O states during programming and control the individual states for each Input, Output, and Output Enable register.

The Specify I/O States During Programming dialog box enables access to each of these BSR cells for control over the individual states. You can use the I/O State (Output Only) settings to set a specific output state and ignore the other values for the individual BSR elements, or you can click the Show BSR Details checkbox for control over the settings for each Input, Output Enable, and Output as you exit programming.

Specifying I/O States During Programming - I/O States and BSR Details

The I/O States During Programming dialog box enables you to set custom I/O states prior to programming.

I/O State (Output Only)

Sets your I/O states during programming to one of the values shown in the list below.

- 1 – I/Os are set to drive out logic High
- 0 – I/Os are set to drive out logic Low
- Last Known State: I/Os are set to the last value that was driven out prior to entering the programming mode, and then held at that value during programming
- Z - Tri-State: I/Os are tristated

When you set your I/O state, the Boundary Scan Register cells are set according to the table below. Use the Show BSR Details option to set custom states for each cell.

Table 11 · Default I/O Output Settings

Output State	Settings		
	Input	Control (Output Enable)	Output
Z (Tri-State)	1	0	0
0 (Low)	1	1	0
1 (High)	0	1	1
Last_Known_State	Last_Known_State	Last_Known_State	Last_Known_State

Table Key:

- 1 – High: I/Os are set to drive out logic High
- 0 – Low: I/Os are set to drive out logic Low
- Last_Known_State - I/Os are set to the last value that was driven out prior to entering the programming mode, and then held at that value during programming

Boundary Scan Registers - Enabled with Show BSR Details

Sets your I/O state to a specific output value during programming AND enables you to customize the values for the Boundary Scan Register (Input, Output Enable, and Output). You can change any Don't Care value in Boundary Scan Register States without changing the Output State of the pin (as shown in the table below).

For example, if you want to Tri-State a pin during programming, set Output Enable to 0; the Don't Care indicates that the other two values are immaterial.

If you want a pin to drive a logic High and have a logic 1 stored in the Input Boundary scan cell during programming, you may set all the values to 1.

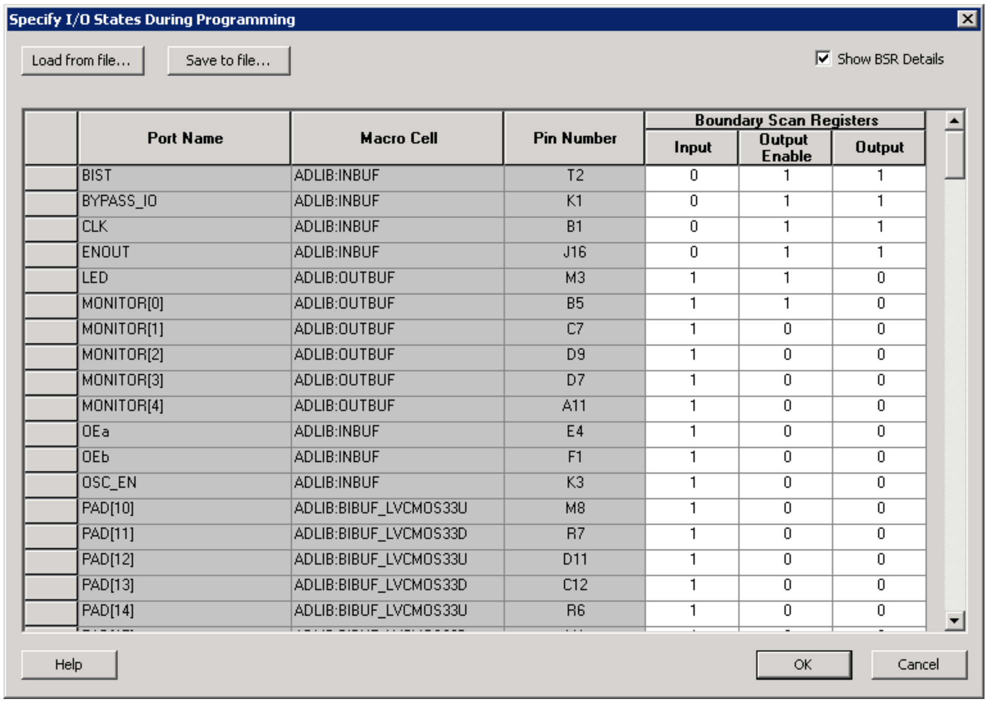
Table 12 · BSR Details I/O Output Settings

Output State	Settings		
	Input	**Output Enable**	**Output**
Z (Tri-State)	Don't Care	0	Don't Care
0 (Low)	Don't Care	1	0
1 (High)	Don't Care	1	1
Last Known State	Last State	Last State	Last State

Table Key:

- 1 – High: I/Os are set to drive out logic High
- 0 – Low: I/Os are set to drive out logic Low
- Don't Care – Don’t Care values have no impact on the other settings.
- Last_Known_State – Sampled value: I/Os are set to the last value that was driven out prior to entering the programming mode, and then held at that value during programming

The figure below shows an example of Boundary Scan Register settings.



Port Name	Macro Cell	Pin Number	Boundary Scan Registers		
			Input	Output Enable	Output
BIST	ADLIB:INBUF	T2	0	1	1
BYPASS_IO	ADLIB:INBUF	K1	0	1	1
CLK	ADLIB:INBUF	B1	0	1	1
ENOUT	ADLIB:INBUF	J16	0	1	1
LED	ADLIB:OUTBUF	M3	1	1	0
MONITOR[0]	ADLIB:OUTBUF	B5	1	1	0
MONITOR[1]	ADLIB:OUTBUF	C7	1	0	0
MONITOR[2]	ADLIB:OUTBUF	D9	1	0	0
MONITOR[3]	ADLIB:OUTBUF	D7	1	0	0
MONITOR[4]	ADLIB:OUTBUF	A11	1	0	0
OEa	ADLIB:INBUF	E4	1	0	0
OEb	ADLIB:INBUF	F1	1	0	0
OSC_EN	ADLIB:INBUF	K3	1	0	0
PAD[10]	ADLIB:BIBUF_LVCMOS33U	M8	1	0	0
PAD[11]	ADLIB:BIBUF_LVCMOS33D	R7	1	0	0
PAD[12]	ADLIB:BIBUF_LVCMOS33U	D11	1	0	0
PAD[13]	ADLIB:BIBUF_LVCMOS33D	C12	1	0	0
PAD[14]	ADLIB:BIBUF_LVCMOS33U	R6	1	0	0

Figure 74 · Boundary Scan Registers

Specify I/O States During Programming Dialog Box

The I/O States During Programming dialog box enables you to specify custom settings for I/Os in your programming file. This is useful if you want to set an I/O to drive out specific logic, or if you want to use a custom I/O state to manage settings for each Input, Output Enable, and Output associated with an I/O.

Load from file

Load from file enables you to load an I/O Settings (*.ios) file. You can use the IOS file to import saved custom settings for all your I/Os. The exported IOS file have the following format:

- Used I/Os have an entry in the IOS file with the following format:

```
set_prog_io_state -portName {<design_port_name>} -input <value> -outputEnable <value> -
output <value>
```

- Unused I/Os have an entry in the IOS file with the following format:

```
set_prog_io_state -pinNumber {<device_pinNumber>} -input <value> -outputEnable <value> -
output <value>
```

Where <value> is:

- 1 – I/O is set to drive out logic High
- 0 – I/O is set to drive out logic Low
- Last_Known_State: I/O is set to the last value that was driven out prior to entering the programming mode, and then held at that value during programming
- Z - Tri-State: I/O is tristated

Save to file

Saves your I/O Settings File (*.ios) for future use. This is useful if you set custom states for your I/Os and want to use them again later in conjunction with a PDC file.

Port Name

Lists the names of all the ports in your design.

Macro Cell

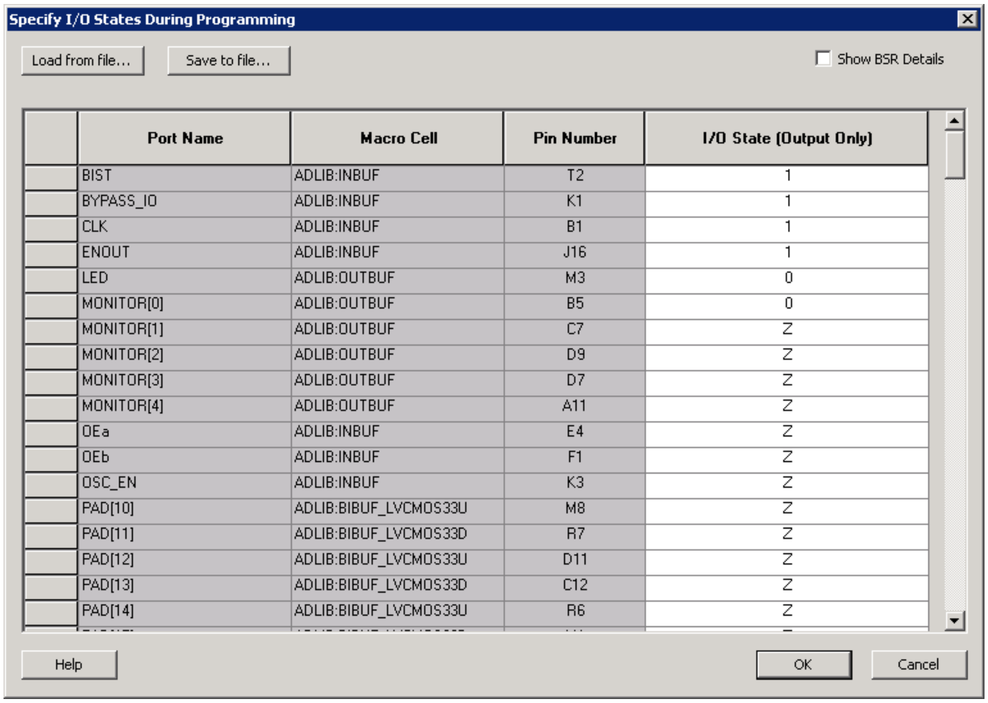
Lists the I/O type, such as INBUF, OUTBUF, PLLs, etc.

Pin Number

The package pin associate with the I/O.

I/O State (Output Only)

Your custom I/O State set during programming. This heading changes to Boundary Scan Register if you select the BSR Details checkbox; see the Specifying I/O States During Programming - I/O States and BSR Details help topic for more information on the BSR Details option.



	Port Name	Macro Cell	Pin Number	I/O State (Output Only)
	BIST	ADLIB:INBUF	T2	1
	BYPASS_IO	ADLIB:INBUF	K1	1
	CLK	ADLIB:INBUF	B1	1
	ENOUT	ADLIB:INBUF	J16	1
	LED	ADLIB:OUTBUF	M3	0
	MONITOR[0]	ADLIB:OUTBUF	B5	0
	MONITOR[1]	ADLIB:OUTBUF	C7	Z
	MONITOR[2]	ADLIB:OUTBUF	D9	Z
	MONITOR[3]	ADLIB:OUTBUF	D7	Z
	MONITOR[4]	ADLIB:OUTBUF	A11	Z
	OEa	ADLIB:INBUF	E4	Z
	OEb	ADLIB:INBUF	F1	Z
	OSC_EN	ADLIB:INBUF	K3	Z
	PAD[10]	ADLIB:BIBUF_LVCMOS33U	M8	Z
	PAD[11]	ADLIB:BIBUF_LVCMOS33D	R7	Z
	PAD[12]	ADLIB:BIBUF_LVCMOS33U	D11	Z
	PAD[13]	ADLIB:BIBUF_LVCMOS33D	C12	Z
	PAD[14]	ADLIB:BIBUF_LVCMOS33U	R6	Z

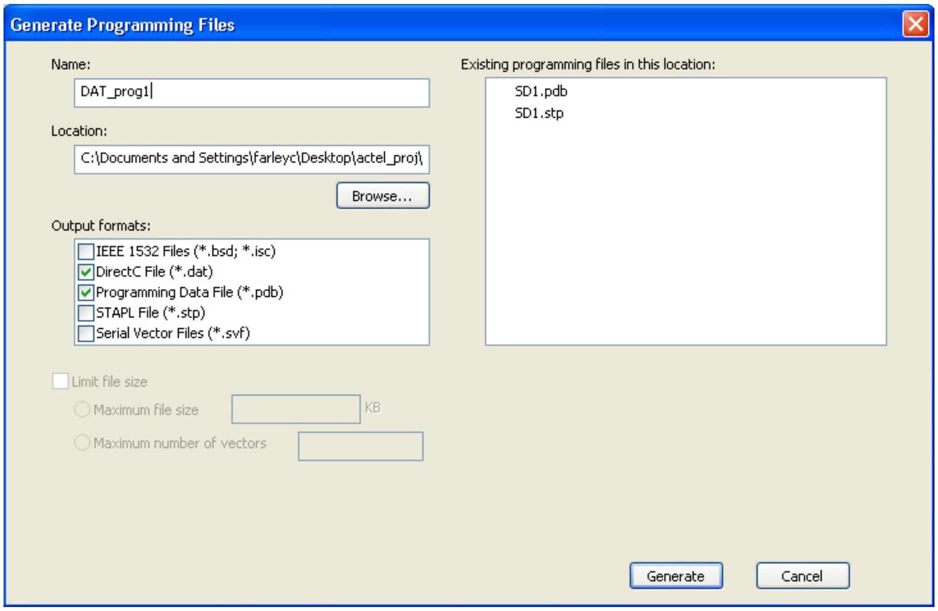
Figure 75 · I/O States During Programming Dialog Box

Generate a DAT file

DAT files are generated via the Generate Programming Files dialog box.

To access the Generate Programming Files dialog box from Libero SoC and generate a DAT file:

1. In the Design Flow window, expand **Implement Design**, right-click **Generate Programming Data** and choose **Open Interactively**. This opens Designer.
2. Click **Programming File** to start FlashPoint.
3. Set your feature and I/O options if necessary. Click **Finish**. This opens the Generate Programming File dialog box, as shown in the figure below.



Generate Programming Files

Name: DAT_prog1

Location: C:\Documents and Settings\farleyc\Desktop\actel_proj\

Output formats:

- □ IEEE 1532 Files (*.bsd; *.isc)
- ☑ DirectC File (*.dat)
- ☑ Programming Data File (*.pdb)
- □ STAPL File (*.stp)
- □ Serial Vector Files (*.svf)

□ Limit file size

○ Maximum file size KB

○ Maximum number of vectors

Existing programming files in this location:

SD1.pdb
SD1.stp

Figure 76 · Generate Programing Files Dialog Box - DirectC File (*.dat)

4. Set your output file **Name** and **Location**.
5. Set your Output Formats to **DirectC file (*.dat)** and **Programming Data File (*.pdb)**.
6. Click **Generate** to create your file.

FlashLock®

Microsemi's SmartFusion devices contain FlashLock circuitry to lock the device by disabling the programming and readback capabilities after programming. Care has been taken to make the locking circuitry very difficult to defeat through electronic or direct physical attack.

FlashLock has three security options: No Lock, Permanent Lock, and Keyed Lock.

No Lock

Creates a programming file which does not secure your device.

Permanent Lock

The permanent lock makes your device one time programmable. It cannot be unlocked by you or anyone else.

Keyed Lock

Within each device, there is a multi-bit security key user key. The number of bits depends on the size of the device. Once secured, read permission and write permission can only be enabled by providing the correct user key to first unlock the device. The maximum security key for the device is shown in the dialog box.

Generating Bitstream and STAPL files

Bitstream allows you to generate a STAPL file for IGLOO, ProASIC3, SmartFusion, Fusion devices. Please consult the Program Files table to find out which file type you should choose.

To generate a STAPL file:

1. From the **Tools** menu, choose **Programming File**.
2. Select **Bitstream** or **STAPL** from the **File Type** drop-down list box. Bitstream files are not available for SmartFusion, IGLOO, ProASIC3 and Fusion devices.
3. **FlashLock**. Select one of the following options:

- **No Locking:** Creates a programming file which does not secure your device.
- **Use Keyed Lock:** Creates a programming file which secures your device with a FlashLock key. The maximum security key for the device is shown in the dialog box. The maximum security key for the device is shown in the dialog box.
- **Use Permanent Lock:** Creates a one-time programmable device.

4. Click **OK**. Designer validates the security key and alerts you to any concerns.

Note: Note: The bitstream file header contains the security key.

Export Programming File

Double-click to export your programming file with default settings.

To modify your programming file settings before you export:

1. Right-click **Export Programming File** and choose **Open Interactively**. FlashPro opens.
2. From the **File** menu, choose **Export** and select your programming file type.
3. Set your options and click **OK**.

SmartFusion2 Only

To generate a STAPL file, in the Libero SoC Design Flow window expand Handoff Design for Production and double-click **Export Programming File**. See the Export Programming Files - SmartFusion2 topic for more information.

To modify your programming file settings before you export:

1. Right-click **Export Programming File** and choose **Open Interactively**. The Configure Programming File Options dialog box opens.
2. Choose your options, such as **DAT file** if you wish to include support for Embedded ISP, or **SPI file** if you need support for Auto Programming.
3. Set your options and click **OK**.

Export Pin Report

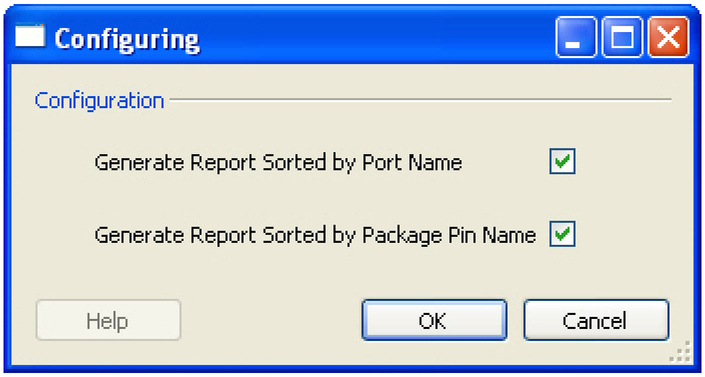
Double-click **Export Pin Report** to display the pin report in your Design Datasheet/Report.

The Pin Report lists the pins in your device. Right-click **Export Pin Report** and choose **Configure Options** to select your pin report type. You can generate a report sorted by port name and/or by package pin name, as shown in the figure below. The Pin Report generates two files:

- <design>_pinrpt_name.rpt - Pin report sorted by name.
- <design>_pinrpt_number.rpt - Pin report sorted by pin number.

You must select at least one report.

Export Pin Report also generates a Bank Report by default; the filename is <design>-bankrpt.rpt.



Configuring

Configuration

Generate Report Sorted by Port Name ☑

Generate Report Sorted by Package Pin Name ☑

Figure 77 · Export Pin Report Dialog Box

Export BSDL File

Double-click Export BSDL File (in the Libero SoC Design Flow window, **Handoff Design for Production > Export BSDL File**) to generate the BSDL File report to your Design Datasheet/Report.

The BSDL file provides a standard file format for electronics testing using JTAG. It describes the boundary scan device package, pin description and boundary scan cell of the input and output pins. BSDL models are available as downloads for many Microsemi SoC devices; see the Microsemi website for more information.

Export IBIS Model

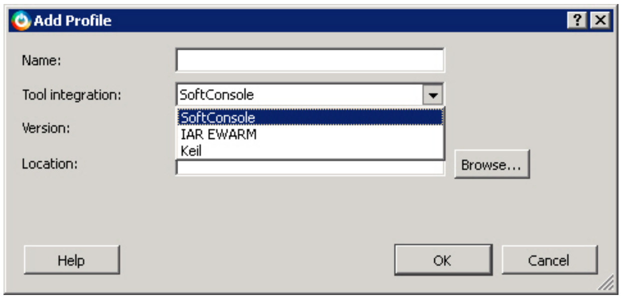
Double-click Export IBIS Model (in the Libero SoC Design Flow window, **Handoff Design for Production > Export IBIS Model**) to generate the IBIS Model report to your Design Datasheet/Report.

The IBIS model report provides a standard file format for recording parameters like driver output impedance, rise/fall time, and input loading, which may then be used by any software application.

See the IBIS model application note for more information on IBIS models.

Develop Firmware - Write Application Code

You must set your default third party software IDE profile and other options in the Tool Profiles dialog box (as shown in the figure below) in order to use Write Application Code.



Tool integration: SoftConsole

Figure 78 · Add Software IDE Tool in Tool Profiles Dialog Box

The generation of your root design automatically generates your firmware drivers and your software IDE workspace. Two projects are created based on the software toolchain selected in your Tools Profile. When you invoke the Write Application Code tool from your design flow the software IDE automatically opens the workspace with both projects.

The software projects are:

- hardware_platform - This project contains all the firmware and hardware abstraction layers that correspond to your hardware design. This project is configured as a library and is referenced by your application project. The contents of this folder get over-written every time you regenerate your root design in Libero SoC.
- application - This project produces a program and results in the binary file. It links with the hardware_platform project. From your application you can reference the header files of any hardware peripherals in the hardware_platform projct because all the include paths have been setup to work right out of the box. This folder does not get overwritten when you regenerate your root design in Libero SoC. This is where you can write your own main.c and other application code, as well as add other user drivers and files.

The benefit of separating your embedded projects into an _app and _hw_platform project enables you to better manage the files generated by Libero SoC vs your own firmware and application code.

Keep your user firmware files and application files in the _app project as this will be maintained upon each re-generation.

To build your workspace, make sure you have both the hw_platform and _app projects set to the same compile target (Release or Debug) and build both projects.

Run **Write Application Code** to open your projects in a third-party development tool, such as SoftConsole, Keil or IAR.

Command Line

If your tool does not support adding any external tools, then you can invoke Libreo SoC directly from the command line and pass these values as arguments, for example:

```
libero.exe "PROJECT_LOCATION:C:/Project" "DESIGN_NAME:MyMSS" "STARTED_BY:Keil"
```

Version Support

Libero SoC v10.0 supports the following versions of third-party development tools:

- SoftConsole v3.3
- IAR v5.4
- Keil v4.14

See Also

Libero SoC Frequently Asked Questions

Running Libero SoC from your Software Tool Chain

Software IDE Integration

View/Configure Firmware Cores

Running Libero SoC from your Software Tool Chain

When launched from your software toolchain, Libero SoC becomes solely an MSS configurator. This can be useful if you are responsible for the embedded code development for the SmartFusion device and are more comfortable in your existing software tool chain.

Any FPGA fabric development needs to be done using the regular Libero® SoC tool flow. Using the Libero SoC in the software toolchain mode only enables you to configure the SmartFusion Microcontroller Subsystem (MSS) and not the FPGA fabric.

The MSS Configurator can be integrated in any software development IDE that supports external tools. Configure your IDE to start the Libero SoC executable and use the parameters below to customize your interface. For SoftConsole, Keil and IAR the parameters are:

```
"PROJECT_LOCATION:<path>" //Project directory location, and the location of your
generated MSS files.
"DESIGN_NAME:<name>" //Name of your design.
"STARTED_BY:<tool>" //Identifies which tool invoked the MSS Configurator; can be
SoftConsole, Keil, or IAR EWARM
```

See Also

Develop Firmware - Write Application Code

Libero SoC Frequently Asked Questions

Software IDE Integration

View/Configure Firmware Cores

Project Manager Tcl Command Reference

A Tcl (Tool Command Language) file contains scripts for simple or complex tasks. You can run scripts from either the Windows or UNIX command line or store and run a series of Tcl commands in a *.tcl batch file. You can also run scripts from within the GUI in Project Manager.

Note: Note: Tcl commands are case sensitive. However, their arguments are not.

The Libero SoC Project Manager supports the following Tcl scripting commands:

Command	Action
add_file_to_library	Adds a file to a library in your project

Command	Action
add_library	Adds a VHDL library to your project
add_modelsim_path	Adds a ModelSim simulation library to your project
add_profile	Adds a profile; sets the same values as the Add or Edit Profile dialog box
associate_stimulus	Associates a stimulus file in your project
change_link_source	Changes the source of a linked file in your project
check_hdl	Checks the HDL in the specified file
check_schematic	Checks the schematic
close_project	Closes the current project in Libero SoC
create_links	Creates a link (or links) to a file/files in your project
create_symbol	Creates a symbol in a module
delete_files	Deletes files from your Libero SoC project
edit_profile	Edits a profile; sets the same values as the Add or Edit Profile dialog box
export_as_link	Exports a file to another directory and links to the file
export_io_constraints_from_adb	Exports the I/O constraints from your project ADB file to an output file
export_profiles	Exports your tool profiles; performs the same action as the Export Profiles dialog box
generate_ba_files	Generates the back-annotate files for your design
generate_hdl_from_schematic	Generates an HDL file from your schematic
generate_hdl_netlist	Generates the HDL netlist for your design and runs the design rule check
import_files (Libero SoC)	Imports files into your Libero SoC project
new_project	Creates a new project in the Libero SoC
open_project	Opens an existing Libero SoC project
organize_cdbs	Organizes the CDB files in your project
organize_constraints	Organizes the constraint files in your project
organize_sources	Organizes the source files in your project
project_settings	Modifies project flow settings for your Libero SoC project

Command	Action
remove_core	Removes a core from your project
remove_library	Removes a VHDL library from your project
remove_profile	Deletes a tool profile
rename_library	Renames a VHDL library in your project
rollback_constraints_from_adb	Opens the ADB file, exports the PDC file, and then replaces it with the specified PDC file
run_designer	Runs Designer with compile and layout options (if selected)
run_drc	Runs the design rule check on your netlist and generates an HDL file
run_simulation	Runs simulation on your project with your default simulation tool and creates a logfile
run_synthesis	Runs synthesis on your project and creates a logfile
save_log	Saves your Libero SoC log file
save_project	Saves your project
save_project_as	Saves your project with a different name
select_profile	Selects a profile to use in your project
set_actel_lib_options	Sets your simulation library to default, or to another library
set_device (Project Manager)	Sets your device family, die, and package in the Project Manager
set_modelsim_options	Sets your ModelSim simulation options
set_option	Sets your synthesis options on a module
set_userlib_options	Sets your user library options during simulation
set_root	Sets the module you specify as the root
synplify	Runs Synplify in batch mode and executes a Tcl script.
synplify_pro	Runs Synplify Pro in batch mode and executes a Tcl script.
unlink	Removes a link to a file in your project
use_file	Specifies which file in your project to use
use_source_file	Defines a module for your project

TCL Command Reference

Introduction to Tcl Scripting

Tcl, the Tool Command Language, pronounced *tickle*, is an easy-to-learn scripting language that is compatible with Libero SoC and Designer software. You can run scripts from either the Windows or UNIX command line or store and run a series of commands in a *.tcl batch file.

This section provides a quick overview of the main features of Tcl:

- Basic syntax
- Types of Tcl commands
- Variables
- Command substitution
- Quotes and braces
- Lists and arrays
- Control structures
- Handling exceptions
- Print statement and Return values
- Running Tcl scripts from the command line
- Running Tcl scripts from the GUI
- Exporting Tcl scripts
- Extended_run_gui
- Extended_run_shell
- Sample Tcl scripts
- Project Manager Tcl Commands
- Designer Tcl Commands

For complete information on Tcl scripting, refer to one of the books available on this subject. You can also find information about Tcl at web sites such as http://www.tcl.tk.

Basic Syntax

Tcl scripts contain one or more commands separated by either new lines or semicolons. A Tcl command consists of the name of the command followed by one or more arguments. The format of a Tcl command is:

```
command arg1 ... argN
```

The command in the following example computes the sum of 2 plus 2 and returns the result, 4.

```
expr 2 + 2
```

The **expr** command handles its arguments as an arithmetic expression, computing and returning the result as a string. All Tcl commands return results. If a command has no result to return, it returns an empty string.

To continue a command on another line, enter a backslash (\) character at the end of the line. For example, the following Tcl command appears on two lines:

```
import -format "edif" -netlist_naming "Generic" -edif_flavor "GENERIC" {prepi.edn}
```

Comments must be preceded by a hash character (#). The comment delimiter (#) must be the first character on a line or the first character following a semicolon, which also indicates the start of a new line. To create a multi-line comment, you must put a hash character (#) at the beginning of each line.

Note: Note: Be sure that the previous line does not end with a continuation character (\). Otherwise, the comment line following it will be ignored.

Special Characters

Square brackets ([ ]) are special characters in Tcl. To use square brackets in names such as port names, you must either enclose the entire port name in curly braces, for example, pin_assign -port {LFSR_OUT[15]} -iostd lvttl -slew High, or lead the square brackets with a slash (\) character as shown in the following example:
pin_assign -port LFSR_OUT\[15\] -iostd lvttl -slew High

Sample Tcl Script

```
#Set up a new design
new_design -name "multiclk" -family "Axcelerator" -path {.}

# Set device, package, speed grade, default I/O standard and
# operating conditions
set_device -die "AX1000" -package "BG729" -speed "-3" \
-voltage "1.5" -iostd "LVTTL" -temprange "COM" -voltrange "COM"

# Import the netlist
import -format "verilog" {multiclk.v}

# Compile the netlist
compile

# Import a PDC file
import_aux -format "pdc" {multiclk.pdc}

# Run standard layout
layout -incremental "OFF"

# Generate backannotated sdf and netlist file
backannotate -name {multiclk_ba} -format "sdf" -language "Verilog"

# Generate timing report
report -type "timing" -sortby "actual" -maxpaths "100" {report_timing.txt}

# Generate programming file
export -format "AFM" -signature "ffff" {multiclk.afm}
```

Types of Tcl commands

There are three types of Tcl commands:

- Built-in commands
- Procedures created with the proc command
- Commands built into the Designer software

Built-in commands

Built-in commands are provided by the Tcl interpreter. They are available in all Tcl applications. Here are some examples of built-in Tcl commands:

- Tcl provides several commands for manipulating file names, reading and writing file attributes, copying files, deleting files, creating directories, and so on.
- exec - run an external program. Its return value is the output (on stdout) from the program, for example:

```
set tmp [ exec myprog ]
puts stdout $tmp
```

- You can easily create collections of values (lists) and manipulate them in a variety of ways.
- You can create arrays - structured values consisting of name-value pairs with arbitrary string values for the names and values.
- You can manipulate the time and date variables.

- You can write scripts that can wait for certain events to occur, such as an elapsed time or the availability of input data on a network socket.

Procedures created with the proc command

You use the proc command to declare a procedure. You can then use the name of the procedure as a Tcl command.

The following sample script consists of a single command named **proc**. The proc command takes three arguments:

- The name of a procedure (myproc)
- A list of argument names (arg1 arg2)
- The body of the procedure, which is a Tcl script

```
proc myproc { arg1 arg2 } {
# procedure body
}
myproc a b
```

Commands built into the software

Many functions that you can perform through the software's GUI interface, you can also perform using an equivalent Tcl command. For example, the `backannotate` command is equivalent to executing the Back-Annotate command from Designer's Tools menu. For a list of Tcl commands supported in the Designer software, see "Tcl Commands."

Variables

With Tcl scripting, you can store a value in a variable for later use. You use the set command to assign variables. For example, the following set command creates a variable named x and sets its initial value to 10.

```
set x 10
```

A variable can be a letter, a digit, an underscore, or any combination of letters, digits, and underscore characters. All variable values are stored as strings.

In the Tcl language, you do not declare variables or their types. Any variable can hold any value. Use the dollar sign ($) to obtain the value of a variable, for example:

```
set a 1
set b $a
set cmd expr
set x 11
$cmd $x*$x
```

The dollar sign $ tells Tcl to handle the letters and digits following it as a variable name and to substitute the variable name with its value.

Global Variables

Variables can be declared global in scope using the Tcl global command. All procedures, including the declaration can access and modify global variables, for example:

```
global myvar
```

Command substitution

By using square brackets ([]), you can substitute the result of one command as an argument to a subsequent command, as shown in the following example:

```
set a 12
set b [expr $a*4]
```

Tcl handles everything between square brackets as a nested Tcl command. Tcl evaluates the nested command and substitutes its result in place of the bracketed text. In the example above, the argument that appears in square brackets in the second set command is equal to 48 (that is, 12* 4 = 48).

Conceptually,

```
  set b [expr $a * 4]
```

expands to

```
  set b [expr 12 * 4 ]
```

and then to

```
  set b 48
```

Quotes and braces

The distinction between braces ({ }) and quotes (" ") is significant when the list contains references to variables. When references are enclosed in quotes, they are substituted with values. However, when references are enclosed in braces, they are not substituted with values.

Example

With Braces	With Double Quotes
set b 2	set b 2
set t { 1 $b 3 }	set t " 1 $b 3 "
set s { [expr $b + $b] }	set s " [expr $b + $b] "
puts stdout $t	puts stdout $t
puts stdout $s	puts stdout $s

will output

```
1 $b 3                              vs.          1 2 3
[ expr $b + $b ]                                4
```

Filenames

In Tcl syntax, filenames should be enclosed in braces { } to avoid backslash substitution and white space separation. Backslashes are used to separate folder names in Windows-based filenames. The problem is that sequences of “\n” or “\t” are interpreted specially. Using the braces disables this special interpretation and specifies that the Tcl interpreter handle the enclosed string literally. Alternatively, double-backslash “\\n” and “\\t” would work as well as forward slash directory separators “/n” and “/t”.For example, to specify a file on your Windows PC at c:\newfiles\thisfile.adb, use one of the following:

```
  {C:\newfiles\thisfile.adb}
  C:\\newfiles\\thisfile.adb
  "C:\\newfiles\\thisfile.adb"
  C:/newfiles/thisfile.adb
  "C:/newfiles/thisfile.adb"
```

If there is white space in the filename path, you must use either the braces or double-quotes. For example:

```
  C:\program data\thisfile.adb
```

should be referenced in Tcl script as

```
  {C:\program data\thisfile.adb} or "C:\\program data\\thisfile.adb"
```

If you are using variables, you cannot use braces { } because, by default, the braces turn off all special interpretation, including the dollar sign character. Instead, use either double-backslashes or forward slashes with double quotes. For example:

```
   "$design_name.adb"
```

Note: Note: To use a name with special characters such as square brackets [ ], you must put the entire name between curly braces { } or put a slash character \ immediately before each square bracket.

The following example shows a port name enclosed with curly braces:

```
pin_assign -port {LFSR_OUT[15]} -iostd lvttl -slew High
```

The next example shows each square bracket preceded by a slash:

```
pin_assign -port LFSR_OUT\[15\] -iostd lvttl -slew High
```

Lists and arrays

A list is a way to group data and handle the group as a single entity. To define a list, use curly braces { } and double quotes " ". For example, the following set command {1 2 3 }, when followed by the list command, creates a list stored in the variable "a." This list will contain the items "1," "2," and "3."

```
set a { 1 2 3 }
```

Here's another example:

```
set e 2
set f 3
set a [ list b c d [ expr $e + $f ] ]
puts $a
```

displays (or outputs):

```
b c d 5
```

Tcl supports many other list-related commands such as lindex, linsert, llength, lrange, and lappend. For more information, refer to one of the books or web sites available on this subject.

Arrays

An array is another way to group data. Arrays are collections of items stored in variables. Each item has a unique address that you use to access it. You do not need to declare them nor specify their size.

Array elements are handled in the same way as other Tcl variables. You create them with the set command, and you can use the dollar sign ($) for their values.

```
set myarray(0) "Zero"
set myarray(1) "One"
set myarray(2) "Two"
for {set i 0} {$i < 3} {incr i 1} {
```

Output:

```
Zero
One
Two
```

In the example above, an array called "myarray" is created by the set statement that assigns a value to its first element. The for-loop statement prints out the value stored in each element of the array.

Special arguments (command-line parameters)

You can determine the name of the Tcl script file while executing the Tcl script by referring to the $argv0 variable.

```
puts "Executing file $argv0"
```

To access other arguments from the command line, you can use the `lindex` command and the *`argv`* variable:

To read the the Tcl file name:

```
lindex $argv 0
```

To read the first passed argument:

```
lindex $argv 1
```

Example

```
puts "Script name is $argv0" ; # accessing the scriptname
puts "first argument is [lindex $argv 0]"
puts "second argument is [lindex $argv 1]"
puts "third argument is [lindex $argv 2]"
puts "number of argument is [llength $argv]"
set des_name [lindex $argv 0]
puts "Design name is $des_name"
```

Control structures

Tcl control structures are commands that change the flow of execution through a script. These control structures include commands for conditional execution (if-then-elseif-else) and looping (while, for, catch).

An "if" statement only executes the body of the statement (enclosed between curly braces) if the Boolean condition is found to be true.

if/else statements

```
if { “$name” == “paul” } then {
…
# body if name is paul
} elseif { $code == 0 } then {
…
# body if name is not paul and if value of variable code is zero
} else {
…
# body if above conditions is not true
}
```

for loop statement

A "for" statement will repeatedly execute the body of the code as long as the index is within a specified limit.

```
for { set i 0 } { $i < 5 } { incr i } {
…
# body here
}
```

while loop statement

A "while" statement will repeatedly execute the body of the code (enclosed between the curly braces) as long as the Boolean condition is found to be true.

```
while { $p > 0 } {
…
}
```

catch statement

A "catch" statement suspends normal error handling on the enclosed Tcl command. If a variable name is also used, then the return value of the enclosed Tcl command is stored in the variable.

```
catch { open “$inputFile” r } myresult
```

Handling Exceptions (Tcl Scripting)

To control the flow of the Designer software based on certain conditions (for example, success or failure of certain commands), you can use the Tcl built-in catch command as follows:

```
if { [ catch {open_design $des_name.adb} ] } {
    puts "Cannot open $des_name.adb"
    export -format "log" -diagnostic $des_name.log"
   return 1
    } else {
   puts "Design $des_name.adb Successfully Opened"
}
## set layout mode to standard
layout -incremental "OFF"
if { [ catch {layout} ] } {
    puts "Layout Failed"
    export -format "log" -diagnostic $des_name.log"
    return 1
} else {
   puts "layout successful"
  export -format log "$des_name.log"
  save_design "$des_name.adb";
  close_design
}
```

Print statement and Return values

Print Statement

Use the puts command to write a string to an output channel. Predefined output channels are “stdout” and “stderr.” If you do not specify a channel, then puts display text to the stdout channel.

Note: Note: The STDIN Tcl command is not supported by Microsemi SoC tools.

Example:

```
set a [ myprog arg1 arg2 ]
puts "the answer from myprog was $a (this text is on stdout)"
puts stdout “this text also is on stdout”
```

Return Values

The return code of a Tcl command is a string. You can use a return value as an argument to another function by enclosing the command with square brackets [ ].

Example:

```
set a [ prog arg1 arg2 ]
exec $a
```

The Tcl command “exec” will run an external program. The return value of “exec” is the output (on stdout) from the program.

```
Example:
set tmp [ exec myprog ]
puts stdout $tmp
```

Running Tcl Scripts from the GUI

Instead of running scripts from the command line, you can use Execute Script dialog box to run a script in the software.

To run a Tcl script from the GUI:

1. In Libero SoC, from the **File** menu choose **Execute Script**.

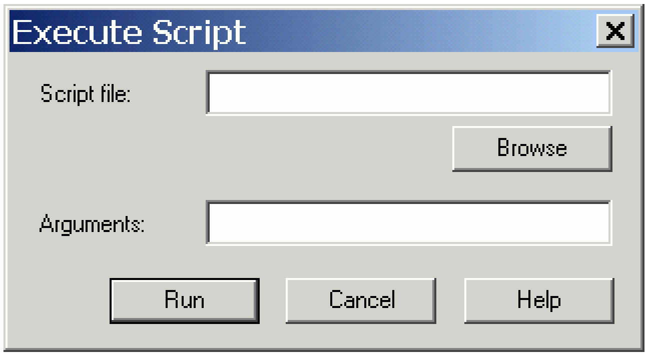
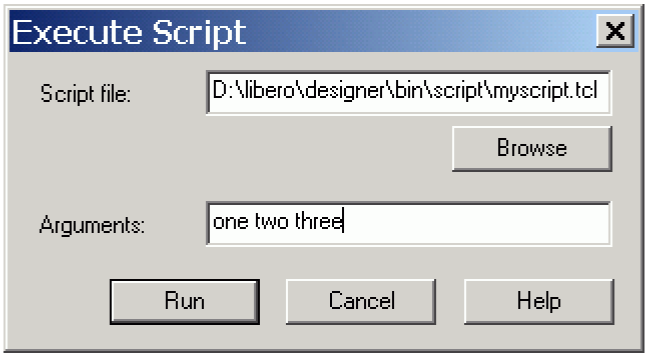


Figure 79 ·

Figure 80 · Execute Script Dialog Box

2. Click **Browse** to display the **Open** dialog box, in which you can navigate to the folder containing the script file to open. When you click **Open**, the software enters the full path and script filename into the Execute Script dialog box for you.
3. In the Arguments edit box, enter the arguments to pass to your Tcl script as shown in the following sample Execute Script dialog box. Separate each argument by a space character. For information about accessing arguments passed to a Tcl script, see "Running Scripts from the command line."



Script file: D:\libero\designer\bin\script\myscript.tcl

Arguments: one two three

Figure 81 · Execute Script Dialog Box Example

4. Click **Run**.

Specify your arguments in the Execute Script dialog box. To get those argument values from your Tcl script, use the following:

```
puts "Script name: $argv0"
puts "Number of arguments: $argc"
set i 0
foreach arg $argv {
  puts "Arg $i : $arg"
  incr i
}
```

Running Tcl scripts from the Command Line

You can run Tcl scripts from your Windows or Unix command line as well as pass arguments to scripts from the command line.

To execute a Tcl script file in the Libero SoC Project Manager software from a shell command line:

At the prompt, type the path to the Microsemi SoC software followed by the word "SCRIPT" and a colon, and then the name of the script file as follows:

```
  <location of Microsemi SoC software>\bin\libero SCRIPT:<filename>
```

where <location of Microsemi SoC software> is the root directory in which you installed the Microsemi SoC software, and <filename> is the name, including a relative or full path, of the Tcl script file to execute. For example, to run the Tcl script file "myscript.tcl", type:

```
C:\libero\designer\bin\libero SCRIPT:myscript.tcl
```

If `myscript.tcl` is in a particular folder named "mydesign", you can use SCRIPT_DIR to change the current working directory before calling the script, as in the following example:

```
C:\libero\designer\bin\libero SCRIPT:myscript.tcl "SCRIPT_DIR:C:\actelprj\mydesign"
```

To execute a Tcl script file in the Designer software from a shell command line:

At the prompt, type the path to the Microsemi SoC software followed by the word "SCRIPT" and a colon, and then the name of the script file as follows:

```
<location of Microsemi SoC software>\bin\designer SCRIPT:<filename>
```

where `<location of Microsemi SoC software>` is the root directory in which you installed the Microsemi SoC software, and**`<filename>`**is the name, including a relative or full path, of the Tcl script file to execute.

```
For example, to run the Tcl script file named "myscript.tcl" from the command line, you can type:
```

```
C:\libero\designer\bin\designer SCRIPT:myscript.tcl
```

If `myscript.tcl` is in a particular folder named "mydesign", you can use SCRIPT_DIR to change the current working directory before calling the script, as in the following example:

```
C:\libero\designer\bin\designer SCRIPT:myscript.tcl "SCRIPT_DIR:C:\actelprj\mydesign"
```

To pass arguments from the command line to your Tcl script file:

At the prompt, type the path to the Microsemi SoC software followed by the SCRIPT argument. Enclose the entire argument expression in double quotes:

```
<location of Microsemi SoC software>\bin\designer "SCRIPT:<filename arg1 arg2 ...>"
```

where `<location of Microsemi SoC software>` is the root directory in which you installed the Microsemi SoC software, and `<filename arg1 arg2 ...`**>**is the name, including a relative or full path, of the Tcl script file and arguments you are passing to the script file.

For example,

```
C:\libero\designer\bin\designer "SCRIPT:myscript.tcl  one two three"
```

To obtain the output from the log file:

At the prompt, type the path to the Microsemi SoC software followed by the SCRIPT and LOGFILE arguments.

```
<location of Microsemi SoC software> SCRIPT:<filename> SCRIPT_ARGS:"a b c"
LOGFILE:<output.log>
```

where

- `location of Microsemi SoC software` is the root directory in which you installed the Microsemi SoC software
- `filename` is the name, including a relative or full path, of the Tcl script file
- `SCRIPT_ARGS` are the arguments you are passing to the script file
- `output.log` is the name of the log file

For example,

```
C:\libero\designer\bin\designer SCRIPT:testTCLparam.tcl SCRIPT_ARGS:"a b c"
LOGFILE:testTCLparam.log
```

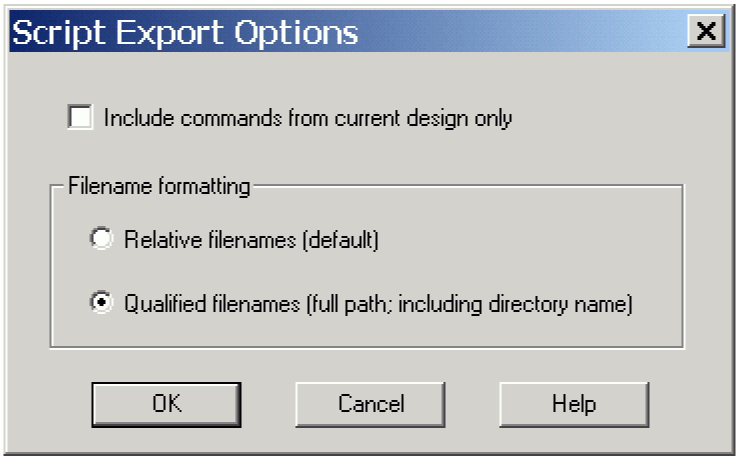
Exporting Tcl Scripts

You can write out a Tcl script file that contains the commands executed in the current session. You can then use this exported Tcl script to re-execute the same commands interactively or in batch. You can also use this exported script to become more familiar with Tcl syntax.

You can export Tcl scripts from the Project Manager or Designer; the actions are the same.

To export a Tcl session script from the Project Manager or Designer:

1. From the **File** menu, choose **Export Script File**. The **Export Script** dialog box appears.
2. Click **OK**. The **Script Export Options** dialog box appears



Script Export Options

☐ Include commands from current design only

Filename formatting

○ Relative filenames (default)

● Qualified filenames (full path; including directory name)

Figure 82 · Script Export Options

5. Check the **Include Commands from Current Design [Project] Only** checkbox. This option applies only if you opened more than one design or project in your current session. If so, and you do not check this box, Project Manager / Designer exports all commands from your current session.
6. Select the radio button for the appropriate filename formatting. To export filenames relative to the current working directory, select **Relative filenames (default)** formatting. To export filenames that include a fully specified path, select **Qualified filenames (full path; including directory name)** formatting.

   Choose **Relative filenames** if you do not intend to move the Tcl script from the saved location, or **Qualified filenames** if you plan to move the Tcl script to another directory or machine.
7. Click **OK**.

Project Manager / Designer saves the Tcl script with the specified filename.

Note: Notes:

- When exporting Tcl scripts, Project Manager and Designer always encloses filenames in curly braces to ensure portability.
- Libero SoC software does not write out any Tcl variables or flow-control statements to the exported Tcl file, even if you had executed the design commands using your own Tcl script. The exported Tcl file only contains the tool commands and their accompanying arguments.

extended_run_gui - Designer Only

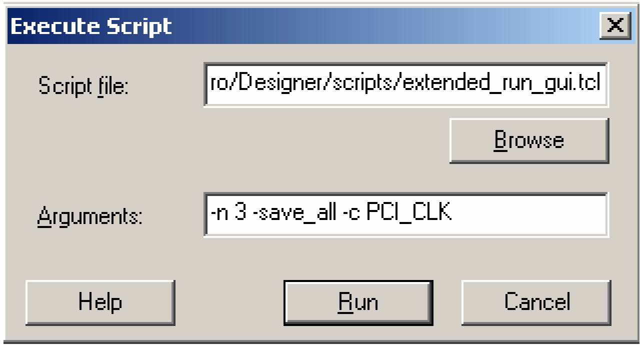
This script is used to reproduce the GUI behavior and is more suited for running through Designer or inside another Designer TCL script.

The only difference from the extended_run_shell Tcl script is that the extended_run_gui.tcl script does not need the `-adb` argument and assumes that the design is already saved and open.

```
extended_run_gui.tcl [-n numPasses] [-starting_seed_index numIndex] [-save_all] [-
compare_criteria value] [-c clockName] [-analysis value] [-slack_criteria value] [-
timing_driven|-standard] [-stop_on_success] [-run_placer value] [-place_incremental value]
[-route_incremental value] [-effort_level numLevel] [-timing_weight numWeight] [-
placer_high_effort value] [-mindel_repair value] [-power_driven value]
```

To invoke extended_run_gui from Designer:

1. Open an *.adb file in Designer.
2. From the **File** menu, select **Execute Script**. This opens the Execute Script dialog box.



Script file: ro/Designer/scripts/extended_run_gui.tcl

Arguments: -n 3 -save_all -c PCI_CLK

Figure 83 · Execute Script Dialog Box

3. Find the *extended_run-gui.tcl* script under *ACTEL_SW_DIR/scripts* and then copy all the parameters in to the arguments section.
4. Click **Run**.

To invoke extended_run_gui from within a TCL script:

1. Save the design in compiled state.

```
 …
compile
save_design “my.adb”
```

2. Override the original argument list in the caller script and then source the *extended_run_gui.tcl* script.

```
set save_argv0 $::argv0
set save_argv $::argv
set ACTEL_SW_DIR $env(ACTEL_SW_DIR)
set ::argv0 “$ACTEL_SW_DIR/scripts/extended_run_gui.tcl”
set ::argv [list -n 3 -save_all -c PCI_CLK]
set ::argc [llength $::argv]
source $::argv0
set ::argv0 $save_argv0
set ::argv $save_argv
set ::argc [llength $::argv]
```

See Also

Running Layout

Multiple Pass Layout

extended_run_shell

extended_run_shell - Designer Only

Note: Note: This is not a Tcl command; it is a shell script that can be run from the command line. To invoke multiple pass layout within another Designer Tcl script, refer to extended_run_gui.

The extended_run_shell Tcl script enables you to run the multiple pass layout in batch mode from a command line. Use this script from the tcl shell "acttclsh". **This is the script or command-line equivalent to using the multiple pass layout in the GUI.**

```
$ACTEL_SW_DIR/bin/acttclsh extended_run_shell.tcl -adb adbFileName.adb [-n numPasses] [-starting_seed_index numIndex] [-save_all] [-compare_criteria value] [-c clockName] [-analysis value] [-slack_criteria value] [-timing_driven|-standard] [-stop_on_success] [-run_placer value] [-place_incremental value] [-route_incremental value] [-placer_high_effort value] [-mindel_repair value] [-power_driven value]
```

Arguments

`-adb` *`adbFileName.adb`*

This is the design file to run multiple passes of layout.

`[-n` *`numPasses`*`]`

Sets the number of passes to run. The default number of passes is 5.

`[-starting_seed_index` *`numIndex`*`]`

Indicates the specific index into the array of random seeds which is to be the starting point for the passes. Its value should range from 1 to 101. If not specified, the default behavior is to continue from the last seed index which was used.

`[-save_all]`

Saves all intermediate designs in<adbFileName>_r<runNum>_s<seedIndex>.adb. The best result is also stored to the original *.adb file as well. The default behavior does not save all results.

`[-compare_criteria` *`value`*`]`

The following table shows the acceptable values for this argument:

Value	Description
frequency	Sets the criteria for comparing results between passes to be clock frequency based. This is the default. This option enables the -c option (described below).
violations	Sets the criteria for comparing results between passes to be timing violations (slack) based. This option enables the -analysis, -slack_criteria, and -stop_on_success options (described below).
power	Sets the criteria for comparing results between passes to be based on the lowest total power.

`[-c` *`clockName`*`]`

Applies only when the clock frequency comparison criteria is used. Specifies the particular clock that is to be examined. If no clock is specified, then the slowest clock frequency in the design in a given pass is used.

`[-analysis` *`value`*`]`

Applies only when the timing violations comparison criteria is used. The following table shows the acceptable values for this argument:

Value	Description
max	Examines timing violations (slacks) obtained from maximum delay analysis. This is the default.
min	Examines timing violations (slacks) obtained from minimum delay analysis.

`[-slack_criteria` *`value`*`]`

Applies only when the timing violations comparison criteria is used. The type of timing violations (slacks) is determined by the -analysis option. The following table shows the acceptable values for this argument:

Value	Description
worst	Sets the timing violations criteria to worst slack. For each pass obtains the most amount of negative slack (or least amount of positive slack if all constraints are met) from the timing violations report. The largest value out of all passes will determine the best pass. This is the default.

Value	Description
tns	Sets the timing violations criteria to total negative slack. For each pass obtains the sum of negative slacks from the first 100 paths from the timing violations report. The largest value out of all passes will determine the best pass. If no negative slacks exist for a pass, then the worst slack is used to evaluate that pass

`[-stop_on_success]`

Applies only when the timing violations comparison criteria is used. The type of timing violations (slacks) is determined by the -analysis option. Stops performing remaining passes if all timing constraints have been met (when there are no negative slacks reported in the timing violations report).

`[-timing_driven|-standard]`

Sets layout mode to be timing driven or standard (non-timing driven). The default is -timing_driven or the mode used in the previous layout command.

`[-run_placer` *`value`*`]`

The following table shows the acceptable values for this argument:

Value	Description
on	Invokes placer. This is the default.
off	Skips placer.

`[-place_incremental` *`value`*`]`

The following table shows the acceptable values for this argument:

Value	Description
off	Discards previous placement. This is the default.
on	Sets the previous placement as the initial starting point for each pass.
fix	Locks previous placement for each pass.

`[-route_incremental` *`value`*`]`

The following table shows the acceptable values for this argument:

Value	Description
off	Discards previous routing. This is the default.
on	Sets the previous routing as the initial starting point for each pass.

`[-placer_high_effort` *`value`*`]`

This is an advanced option that is available only for SmartFusion, IGLOO, ProASIC3 and Fusion families. The following table shows the acceptable values for this argument:

Value	Description
off	Runs layout in regular effort. This is the default.
on	Activates high effort layout mode.

`[-mindel_repair value]`

This is an advanced option that is available only for SmartFusion, IGLOO, ProASIC3 and Fusion families. The following table shows the acceptable values for this argument:

Value	Description
off	Does not run minimum delay violations repair. This is the default.
on	Enables repair of minimum delay violations during route.

`[-power_driven value]`

This option is available only for IGLOO, ProASIC3, SmartFusion2, SmartFusion, Fusionfamilies. The following table shows the acceptable values for this argument:

Value	Description
off	Does not run power-driven layout. This is the default.
on	Enables power-driven layout.

Return

A non-zero value will be returned on error.

Supported Families

IGLOO, ProASIC3, SmartFusion2, SmartFusion, Fusion

Exceptions

None

Example

1. On *my.adb*, run 5 (default) passes continuing from the last seed index using slowest clock frequency (default) comparison criteria.
   ```
   % acttclsh extended_run_shell.tcl -adb my.adb
   ```
2. On *my.adb*, run 3 passes starting with seed index 6, saving all results, using clock frequency comparison criteria for clock "PCI_CLK".
   ```
   % acttclsh extended_run_shell.tcl -adb my.adb -n 3 -starting_seed_index 6 -save_all -
   c PCI_CLK
   ```
3. On my.adb, run 5 (default) passes continuing from the last seed index, saving all results, using timing violations comparison criteria with maximum delay (default) analysis and worst slack (default) criteria; invoke high effort layout.
   ```
   % acttclsh extended_run_shell.tcl -adb my.adb -save_all -compare_criteria violations
   –placer_high_effort on
   ```
4. On my.adb, run 5 (default) passes continuing from the last seed index, saving all results, using timing violations comparison criteria with maximum delay (default) analysis and total negative slack criteria; invoke placement effort level 5.
   ```
   % acttclsh extended_run_shell.tcl -adb my.adb -save_all -compare_criteria violations
   -slack_criteria tns –effort_level 5
   ```
5. On my.adb, run 5 (default) passes continuing from the last seed index, saving all results, using timing violations comparison criteria with minimum delay analysis and worst slack (default) criteria; stop if there are no violations.
   ```
   % acttclsh extended_run_shell.tcl -adb my.adb -save_all -compare_criteria violations
   -analysis min –stop_on_success
   ```

6. On my.adb, run 5 (default) passes continuing from the last seed index, saving all results, using timing violations comparison criteria with minimum delay analysis and total negative slack criteria; invoke repair of minimum delay violations.

```
% acttclsh extended_run_shell.tcl -adb my.adb -save_all -compare_criteria violations
-analysis min -slack_criteria tns –mindel_repair on
```

See Also

Running Layout

Multiple Pass Layout

extended_run_gui

Sample Tcl Script - Project Manager

The following Tcl commands create a new project named proj1 and sets your project options.

```
#Create new project
new_project -name proj1 -location c:/actelprj -family fusion –die AFS090 -package "108
QFN" -hdl VHDL
#Import HDL source file named hdlsource1.vhd
import_files -hdl_source c:\hdlsource1.vhd
#Run synthesis and create a logfile named synth1.
run_synthesis -logfile synth.log
# he default ADB file, run Compile, run Layout
run_designer -logfile designer_log -adb new -compile TRUE -layout TRUE -export_ba TRUE
```

Tcl Flow in the Libero SoC

Use the following commands to manage and build your project in the Libero SoC.

Design Flow in the Project Manager

The Tcl commands below outline the entire design flow. Once you create a project in the Project Manager you can use the commands below to complete every operation from synthesis to generating an HDL netlist. Click any command to go to the command definition.

```
run_synthesis [-logfile name]
run_simulation [-logfile name]
check_hdl -file filename
check_schematic -file filename
create_symbol [-module module]
export_io_constraints_from_adb -adb filename -output outputfilename
generate_ba_files -adb filename
generate_hdl_from_schematic [-module modulename]
generate_hdl_netlist [-netlist filename] [-run_drc "TRUE | FALSE"]
rollback_constraints_from_adb -adb filename -output output_filename
run_designer [-logfile filename] [-script "script to append"] [-append_commands "commands
to execute"] [-adb "new | open | default"] [-compile "TRUE | FALSE"] [-layout "TRUE |
FALSE"] [-export_ba "TRUE | FALSE"]
run_drc [-netlist file] [-gen_hdl "TRUE | FALSE"]
```

Manage Profiles in the Project Manager

```
add_profile -name profilename -type "synthesis | simulation | stimulus | flashpro |
physynth | coreconfig" -tool profiletool -location tool_location [-args tool_parameters]
[-batch "TRUE | FALSE"]
```

```
edit_profile -name profilename -type "synthesis | simulation | stimulus | flashpro |
physynth | coreconfig" -tool profiletool -location tool_location [-args tool_parameters]
[-batch "TRUE | FALSE"] [-new_name name]
```

```
export_profiles -file name [-export "predefined | user | all"]
```

```
remove_profile -name profile_name
```

```
select_profile -name profile_name
```

Linking Files

```
change_link_source -file filename -path pathname
```

```
create_links [-hdl_source file]* [-stimulus file]* [-sdc file]* [-pin file]* [-dcf file]*
[-gcf file]* [-pdc file]* [-crt file]* [-vcd file]*
```

```
export_as_link -file filename -path link_path
```

```
unlink -file file [-local local_filename]
```

Set Simulation Options in the Project Manager

```
add_modelsim_path -lib library_name [-path library_path] [-remove " "]
```

Set Device in the Project Manager

```
set_device [-family family] [-die die] [-package package]
```

Miscellaneous Operations in the Project Manager

```
project_settings [-hdl "VHDL | VERILOG"] [-auto_update_modelsim_ini "TRUE | FALSE"] [-
auto_update_viewdraw_ini "TRUE | FALSE"] [-block_mode "TRUE | FALSE"] [-
auto_generate_synth_hdl "TRUE | FALSE"] [-auto_generate_physynth_hdl "TRUE | FALSE"] [-
auto_run_drc "TRUE | FALSE"] [-auto_generate_viewdraw_hdl "TRUE | FALSE"] [-
auto_file_detection "TRUE | FALSE"]
```

```
refresh
```

```
set_option [-synth "TRUE | FALSE"] [-physynth "TRUE | FALSE"] [-module "module"]
```

```
remove_core -name core_name
```

Project Manager Tcl Command Reference

A Tcl (Tool Command Language) file contains scripts for simple or complex tasks. You can run scripts from either the Windows or UNIX command line or store and run a series of Tcl commands in a *.tcl batch file. You can also run scripts from within the GUI in Project Manager.

Note: Note: Tcl commands are case sensitive. However, their arguments are not.

The Libero SoC Project Manager supports the following Tcl scripting commands:

Command	Action
add_file_to_library	Adds a file to a library in your project
add_library	Adds a VHDL library to your project
add_modelsim_path	Adds a ModelSim simulation library to your project
add_profile	Adds a profile; sets the same values as the Add or Edit Profile dialog box

Command	Action
associate_stimulus	Associates a stimulus file in your project
change_link_source	Changes the source of a linked file in your project
check_hdl	Checks the HDL in the specified file
check_schematic	Checks the schematic
close_project	Closes the current project in Libero SoC
create_links	Creates a link (or links) to a file/files in your project
create_symbol	Creates a symbol in a module
delete_files	Deletes files from your Libero SoC project
edit_profile	Edits a profile; sets the same values as the Add or Edit Profile dialog box
export_as_link	Exports a file to another directory and links to the file
export_io_constraints_from_adb	Exports the I/O constraints from your project ADB file to an output file
export_profiles	Exports your tool profiles; performs the same action as the Export Profiles dialog box
generate_ba_files	Generates the back-annotate files for your design
generate_hdl_from_schematic	Generates an HDL file from your schematic
generate_hdl_netlist	Generates the HDL netlist for your design and runs the design rule check
import_files (Libero SoC)	Imports files into your Libero SoC project
new_project	Creates a new project in the Libero SoC
open_project	Opens an existing Libero SoC project
organize_cdbs	Organizes the CDB files in your project
organize_constraints	Organizes the constraint files in your project
organize_sources	Organizes the source files in your project
project_settings	Modifies project flow settings for your Libero SoC project
remove_core	Removes a core from your project
remove_library	Removes a VHDL library from your project
remove_profile	Deletes a tool profile

Command	Action
rename_library	Renames a VHDL library in your project
rollback_constraints_from_adb	Opens the ADB file, exports the PDC file, and then replaces it with the specified PDC file
run_designer	Runs Designer with compile and layout options (if selected)
run_drc	Runs the design rule check on your netlist and generates an HDL file
run_simulation	Runs simulation on your project with your default simulation tool and creates a logfile
run_synthesis	Runs synthesis on your project and creates a logfile
save_log	Saves your Libero SoC log file
save_project	Saves your project
save_project_as	Saves your project with a different name
select_profile	Selects a profile to use in your project
set_actel_lib_options	Sets your simulation library to default, or to another library
set_device (Project Manager)	Sets your device family, die, and package in the Project Manager
set_modelsim_options	Sets your ModelSim simulation options
set_option	Sets your synthesis options on a module
set_userlib_options	Sets your user library options during simulation
set_root	Sets the module you specify as the root
synplify	Runs Synplify in batch mode and executes a Tcl script.
synplify_pro	Runs Synplify Pro in batch mode and executes a Tcl script.
unlink	Removes a link to a file in your project
use_file	Specifies which file in your project to use
use_source_file	Defines a module for your project

Tcl Command Documentation Conventions

The following table shows the typographical conventions used for the Tcl command syntax.

Syntax Notation	Description
`command - argument`	Commands and arguments appear in Courier New typeface.
variable	*Variables appear in blue, italic Courier New typeface. You must substitute an appropriate value for the variable.*
[-argument*value*] [*variable*]+	Optional arguments begin and end with a square bracket with one exception: if the square bracket is followed by a plus sign (+), then users must specify at least one argument. The plus sign (+) indicates that items within the square brackets can be repeated. Do not enter the plus sign character.

Note: Note: All Tcl commands are case sensitive. However, their arguments are not.

Examples

Syntax for the get_defvar command followed by a sample command:

```
get_defvar variable
```

```
get_defvar “DESIGN”
```

Syntax for the backannotate command followed by a sample command:

```
backannotate -name file_name -format format_type -language language -dir directory_name [-
netlist] [-pin]
```

```
backannotate -dir \
 {..\design} -name "fanouttest_ba.sdf" -format "SDF" -language "VERILOG" \
-netlist
```

Wildcard Characters

You can use the following wildcard characters in names used in Tcl commands:

Wildcard	What it Does
\	Interprets the next character literally
?	Matches any single character
*	Matches any string
[]	Matches any single character among those listed between brackets (that is, [A-Z] matches any single character in the A-to-Z range)

Note: Note: The matching function requires that you add a slash (\) before each slash in the port, instance, or net name when using wildcards in a PDC command and when using wildcards in the Find feature of the MultiView Navigator. For example, if you have an instance named “A/B12” in the netlist, and you enter that name as “A\/B*” in a PDC command, you will not be able to find it. In this case, you must specify the name as A\\\/B*.

Special Characters [ ], { }, and \

Sometimes square brackets ([ ]) are part of the command syntax. In these cases, you must either enclose the open and closed square brackets characters with curly brackets ({ }) or precede the open and closed square brackets ([ ]) characters with a backslash (\). If you do not, you will get an error message.

For example:

```
pin_assign -port {LFSR_OUT[0]} -pin 15
or
pin_assign -port LFSR_OUT\[0\] -pin 180
```

Note: Note: Tcl commands are case sensitive. However, their arguments are not.

Entering Arguments on Separate Lines

To enter an argument on a separate line, you must enter a backslash (\) character at the end of the preceding line of the command as shown in the following example:

```
backannotate -dir \
{..\design} -name "fanouttest_ba.sdf" -format "SDF" -language "VERILOG" \
-netlist
```

See Also

Introduction to Tcl scripting

Basic syntax

About Designer Tcl commands

Project Manager Tcl Commands

add_file_to_library

Tcl command; adds a file to a library in your project.

```
add_file_to_library
-library name
-file name
```

Arguments

`-library name`

Name of the library where you wish to add your file.

`-file name`

Specifies the new name of the file you wish to add (must be a full pathname).

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

- None

Example

Add a file named foo.vhd from the ./project/hdl directory to the library 'my_lib'

```
add_file_to_library -library my_lib -file ./project/hdl/foo.vhd
```

See Also

`add_library`

`remove_library`

`rename_library`

Project Manager Tcl Command Reference

add_library

Tcl command; adds a VHDL library to your project.

```
add_library
-library name
```

Arguments

`-library name`

Specifies the name of your new library.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

- None

Example

Create a new library called 'my_lib'.

```
add_library –library my_lib
```

See Also

remove_library

rename_library

Project Manager Tcl Command Reference

add_modelsim_path

Tcl command; adds a ModelSim simulation library to your project.

```
add_modelsim_path -lib library_name [-path library_path] [-remove " "]
```

Arguments

`-lib library_name`

Name of the library you want to add.

`-path library_path`

Path to library that you want to add.

`-remove " "`

Name of library you want to remove (if any).

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Example

Add the ModelSim library 'msim_update2' located in the c:\modelsim\libraries directory and remove the library 'msim_update1':

```
add_modelsim_path -lib msim_update2 [-path c:\modelsim\libraries] [-remove msim_update1]
```

See Also

Project Manager Tcl Command Reference

add_profile

Tcl command; sets the same values as the Add or Edit Profile dialog box.

```
add_profile -name profilename -type value -tool profiletool -location tool_location [-args
tool_parameters] [-batch value]
```

Arguments

`-name profilename`

Specifies the name of your new profile.

`-type` *`value`*

Specifies your profile type, where value is one of the following:

Value	Description
synthesis	New profile for a synthesis tool
simulation	New profile for a simulation tool
stimulus	New profile for a stimulus tool
flashpro	New FlashPro tool profile

`-tool` *`profiletool`*

Name of the tool you are adding to the profile.

`-location` *`tool_location`*

Full pathname to the location of the tool you are adding to the profile.

`-args` *`tool_parameters`*

Profile parameters (if any).

`-batch` *`value`*

Runs the tool in batch mode (if TRUE). Possible values are:

Value	Description
TRUE	Runs the profile in batch mode
FALSE	Does not run the profile in batch mode

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Example

Create a new FlashPro tool profile called 'myflashpro' linked to a FlashPro installation in my c:\programs\actel\flashpro\bin directory

```
new_profile -name myflashpro -type flashpro -tool flashpro.exe -location
c:\programs\actel\flashpro\bin\flashpro.exe -batch FALSE
```

See Also

Project Manager Tcl Command Reference

associate_stimulus

Tcl command; associates a stimulus file in your project.

```
-associate_stimulus
[-file name]*
```

```
[-mode value]
-module value
```

Arguments

`-file name`

Specifies the name of the file to which you want to associate your stimulus files.

`-mode value`

Specifies whether you are creating a new stimulus association, adding, or removing; possible values are:

Value	Description
new	Creates a new stimulus file association
add	Adds a stimulus file to an existing association
remove	Removes an stimulus file association

`-module value`

Sets the module, where value is the name of the module.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Example

The example associates a new stimulus file 'stim.vhd' for stimulus.

```
-associate_stimulus -file stim.vhd -mode new -module stimulus
```

See Also

Project Manager Tcl Command Reference

change_link_source

Tcl command; changes the source of a linked file in your project.

```
change_link_source -file filename -path new_source_path
```

Arguments

`-file filename`

Name of the linked file you want to change.

`-path new_source_path`

Location of the file you want to link to.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Example

Change the link to a file 'sim1.vhd' in your project and link it to the file in c:\actel\link_source\simulation_test.vhd

```
change_link_source -file sim1.vhd -path c:\actel\link_source\simulation_test.vhd
```

See Also

[Project Manager Tcl Command Reference]

check_hdl

Tcl command; checks the HDL in the specified file.

```
check_hdl -file filename
```

Arguments

```
-file filename
```

Name of the HDL file you want to check.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Example

Check HDL on the file hdl1.vhd.

```
check_hdl -file hdl1.vhd
```

See Also

[Project Manager Tcl Command Reference]

close_project

Tcl command; closes the current project in Libero SoC. Equivalent to clicking the File menu, and choosing Close Project.

```
close_project
```

Arguments

None

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

- None

Example

```
close_project
```

See Also

open_project

Project Manager Tcl Command Reference

check_schematic

Tcl command; checks the schematic.

```
check_schematic -file filename
```

Arguments

```
-file filename
```

Name of the schematic file you want to check.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Example

Check schematic on the file schem.2vd.

```
check_schematic -file schem.2vd
```

See Also

Project Manager Tcl Command Reference

create_links

Tcl command; creates a link (or links) to a file/files in your project.

```
create_links [-hdl_source file]* [-stimulus file]* [-sdc file]* [-pin file]* [-dcf file]* [-
gcf file]* [-pdc file]* [-crt file]* [-vcd file]*
```

Arguments

```
-hdl_source file
```

Name of the HDL file you want to link.

```
-stimulus file
```

Name of the stimulus file you want to link.

```
-sdc file
```

Name of the SDC file you want to link.

```
-pin file
```

Name of the PIN file you want to link.

```
-dcf file
```

Name of the DCF file you want to link.

```
-gcf file
```

Name of the GCF file you want to link.

```
-pdc file
```

Name of the PDC file you want to link.

```
-crt file
```

Name of the crt file you want to link.

```
-vcd file
```

Name of the VCD file you want to link.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Example

Create a link to the file hdl1.vhd.

```
create links [-hdl_source hdl1.vhd]
```

See Also

Project Manager Tcl Command Reference

create_symbol

Tcl command; creates a symbol in a module.

```
create_symbol [-module module]
```

Arguments

```
-module module
```

Name of the symbol module you want to create.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Example

Create a symbol named mod2.

```
create_symbol [-module mod2]
```

See Also

Project Manager Tcl Command Reference

defvar_get

Tcl command; provides access to the internal variables within Libero and returns its value. This command also prints the value of the variable on the Log window.

```
defvar_get -name variable
```

Arguments

variable

The internal variable.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Example

Example 1: Prints the design name on the log window.

```
   defvar_get -name “DESIGN”
set variableToGet "DESIGN"
set valueOfVariable [defvar_get $variableToGet]
puts "The value is $valueOfVariable"
```

See Also

Project Manager Tcl Command Reference

defvar_set

defvar_set

Tcl command; the defvar_set command sets an internal variable in the Libero system. You must specify at least one argument for this command.

```
defvar_set -name variable -value  value
```

Arguments

Variable must be a valid internal variable and could be accompanied by an optional value. If the *value* is provided, the *variable* is set to the value. If the *value* is null the *variable* is reset.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None.

Example

Example 1:

```
defvar_set -name “FORMAT” -value “VHDL”
```

Sets the FORMAT internal variable to VHDL.

Example 2:

```
set variableToSet "DESIGN"
set valueOfVariable “VHDL”
defvar_set $variableToSet $valueOfVariable
```

These commands set the FORMAT variable to VHDL, shows the use of variables for this command.

See Also

Project Manager Tcl Command Reference

`defvar_get`

delete_files

Tcl command; deletes files in your Libero SoC project.

```
delete_files
-file value
-from_disk
```

Arguments

`-file value`

Specifies the file you wish to delete from the project. This parameter is required for this Tcl command. It does not delete the file from the disk. Use the -from_disk flag to delete a file from the disk. Value is the name of the file you wish to delete (including the full pathname).

`-from_disk`

Deletes a file from the disk.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

- None

Example

Delete the files file1.vhd and file2.vhd from the project, and delete the file top_palace.sdc from the disk.

```
delete_files –file ./project/hdl/file1.vhd –file ./project/hdl/file2.vhd
delete_files –from_disk –file ./project/phy_synthesis/top_palace.sdc
```

The following command deletes the core 'add1' from your disk and project (it is the same as the command to delete an IP core from your disk and project).

```
delete_files -from_disk -file ./project/component/work/add1/add1.cxf
```

See Also

Project Manager Tcl Command Reference

`close_project`

`new_project`

edit_profile

Tcl command; sets the same values as the Add or Edit Profile dialog box.

```
edit_profile -name profilename -type value -tool profiletool -location profilelocation [-args
parameters] [-batch value] [-new_name name]
```

Arguments

`-name` *`profilename`*

Specifies the name of your new profile.

`-type` *`value`*

Specifies your profile type, where value is one of the following:

Value	Description
synthesis	New profile for a synthesis tool
simulation	New profile for a simulation tool
stimulus	New profile for a stimulus tool
flashpro	New FlashPro tool profile

`-tool` *`profiletool`*

Name of the tool you are adding to the profile.

`-location` *`profilelocation`*

Full pathname to the location of the tool you are adding to the profile.

`-args` *`parameters`*

Profile tool parameters (if any).

`-batch` *`value`*

Runs the tool in batch mode (if TRUE). Possible values are:

Value	Description
TRUE	Runs the profile in batch mode
FALSE	Does not run the profile in batch mode

`-new_name` *`name`*

Name of new profile.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Example

Edit a FlashPro tool profile called 'myflashpro' linked to a new FlashPro installation in my c:\programs\actel\flashpro\bin directory, change the name to updated_flashpro.

```
edit_profile -name myflashpro -type flashpro -tool flashpro.exe -location
c:\programs\actel\flashpro\bin\flashpro.exe -batch FALSE -new_name updated_flashpro
```

See Also

Project Manager Tcl Command Reference

export_as_link

Tcl command; exports a file to another directory and links to the file.

```
export_as_link -file filename -path link_path
```

Arguments

`-file` *`filename`*

Name of the file you want to export as a link.

`-path` *`link_path`*

Path of the link.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Example

Export the file hdl1.vhd as a link to c:\actel\link_source.

```
export_as_link -file hdl1.vhd -path c:\actel\link_source
```

See Also

Project Manager Tcl Command Reference

export_io_constraints_from_adb

Tcl command; exports the I/O constraints from your project ADB file to an output file.

```
export_io_constraints_from_adb -adb filename -output outputfilename
```

Arguments

`-adb` *`filename`*

Specifies name of the ADB file from which you want to export your I/O constraints.

`-output` *`filename`*

Specifies the output filename for your exported I/O constraints.

Supported Families

SmartFusion, IGLOO, ProASIC3 and Fusion

Exceptions

None

Example

The following example exports the I/O constraint file ios.pdc from the project file designer1.adb:

```
export_io_constraints_from_adb -adb designer1.adb -output ios.pdc
```

See Also

Project Manager Tcl Command Reference

export_profiles

Tcl command; exports your tool profiles. Performs the same action as the Export Profiles dialog box.

```
export_profile -file name [-export value]
```

Arguments

```
-file  name
```

Specifies the name of your exported profile.

```
-export  value
```

Specifies your profile export options. The following table shows the acceptable values for this argument:

Value	Description
predefined	Exports only predefined profiles
user	Exports only user profiles
all	Exports all profiles

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Example

The following command exports all profiles to the file 'all_profiles':

```
export_profiles -file all_profiles [-export all]
```

See Also

Project Manager Tcl Command Reference

generate_ba_files

Tcl command; generates the back-annotate files for your design.

```
generate_ba_files -adb filename
```

Arguments

```
-adb filename
```

Specifies name of the ADB file from which you wish to generate the backannotate files.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Example

The following example generates backa-nnotate files from the file designer1.adb:

```
generate_ba_files -adb designer1.adb
```

See Also

Project Manager Tcl Command Reference

generate_hdl_from_schematic

Tcl command; generates an HDL file from your schematic.

```
generate_hdl_from_schematic [-module modulename]
```

Arguments

`-module modulename`

Specifies the module name for your new HDL module

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Example

The following example generates a new HDL module module1.vhd:

```
generate_hdl_from_schematic [-module module1.vhd]
```

See Also

Project Manager Tcl Command Reference

generate_hdl_netlist

Tcl command; generates the HDL netlist for your design and runs the design rule check.

```
generate_hdl_netlist [-netlist filename] [-run_drc value]
```

Arguments

`-netlist  filename`

Specifies the filename of your netlist.

`-run_drc  value`

Runs the design rule check. The following table shows the acceptable values for this argument:

Value	Description
TRUE	Runs the design rule check
FALSE	Generates your netlist without running the design rule check

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Example

The following example generates your netlist netlist2 and runs the design rule check:

```
generate_hdl_netlist [-netlist netlist2][-run_drc TRUE]
```

See Also

Project Manager Tcl Command Reference

import_files (Libero SoC)

Tcl command; the import_files command imports all the files you need in your Libero SoC project.

```
import_files
-schematic {file}
-symbol {file}
-smartgen_core {file}
-ccp {file}
-stimulus {file}
-hdl_source {file}
-edif {file}
-sdc {file}
-pin {file}
-dcf {file}
-pdc {file}
-gcf {file}
-vcd {file}
-saif {file}
-crt {file}
-simulation {file}
-profiles {file}
-cxf {file}
-templates {file}
-ccz {file}
-wf_stimulus {file}
-modelsim_ini {file}
```

Arguments

```
-schematic {file}
```

Specifies the schematics you wish to import into your IDE project. Type parameter must be repeated for each file.

`-symbol` *{file}*

Specifies the symbols you wish to import into your IDE project. Type parameter must be repeated for each file.

`-smartgen_core` *{file}*

Specifies the cores you wish to import into your project. Type parameter must be repeated for each file.

`-ccp` *{file}*

Specifies the ARM or Cortex-M1 cores you wish to import into your project. Type parameter must be repeated for each file.

`-stimulus` *{file}*

Specifies HDL stimulus files you wish to import into your project. Type parameter must be repeated for each file.

`-hdl_source` *{file}*

Specifies the HDL source files you wish to import into your project. Type parameter must be repeated for each file.

`-edif` *{file}*

Specifies the EDIF files you wish to import into your project. Type parameter must be repeated for each file.

`-constraint_sdc` *{file}*

Specifies the SDC constraint files you wish to import into your project. Type parameter must be repeated for each file.

`-constraint_pin` *{file}*

Specifies the PIN constraint files you wish to import into your project. Type parameter must be repeated for each file.

`-constraint_dcf` *{file}*

Specifies the DCF constraint files you wish to import into your project. Type parameter must be repeated for each file.

`-constraint_pdc` *{file}*

Specifies the PDC constraint files you wish to import into your project. Type parameter must be repeated for each file.

`-constraint_gcf` *{file}*

Specifies the GCF constraint files you wish to import into your project. Type parameter must be repeated for each file.

`-constraint_vcd` *{file}*

Specifies the VCD constraint files you wish to import into your project. Type parameter must be repeated for each file.

`-constraint_saif` *{file}*

Specifies the SAIF constraint files you wish to import into your project. Type parameter must be repeated for each file.

`-constraint_crt` *{file}*

Specifies the CRT constraint files you wish to import into your project. Type parameter must be repeated for each file.

`-simulation` *{file}*

Specifies the simulation files you wish to import into your Libero SoC project. Type parameter must be repeated for each file.

`-profiles` *{file}*

Specifies the profile files you wish to import into your Libero SoC project. Type parameter must be repeated for each file.

`-cxf` *{file}*

Specifies the CXF file (such as SmartDesign components) you wish to import into your Libero SoC project. Type parameter must be repeated for each file.

`-templates  ` *{file}*

Specifies the template file you wish to import into your IDE project.

`-ccz  {file}`

Specifies the IP core file you wish to import into your project.

`-wf_stimulus {file}`

Specifies the WaveFormer Pro stimulus file you wish to import into your project.

`-modelsim_ini  {file}`

Specifies the ModelSIM INI file that you wish to import into your project.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Example

The command below imports the HDL source files file1.vhd and file2.vhd:

```
import_files -hdl_source file1.vhd –hdl_source file2.vhd
```

See Also

Project Manager Tcl Command Reference

new_project

Tcl command; creates a new project in the Libero SoC. If you do not specify a location, Libero SoC saves the new project in your current working directory.

```
new_project -name project_name -location project_location -family family_name –die device_die -package package_name -hdl HDL_type -speed speed_grade -die_voltage value -adv_options value
```

Arguments

`-name project_name`

The name of the project. This is used as the base name for most of the files generated from Libero SoC.

`-location project_location`

The location of the project. Must not be an existing directory.

`-family family_name`

The Microsemi SoC device family for your targeted design.

`-die device_die`

Die for your targeted design.

`-package package_name`

Package for your targeted design.

`-hdl HDL_type`

Sets the HDL type for your new project.

Value	Description
VHDL	Sets your new projects HDL type to VHDL
VERILOG	Sets your new projects to Verilog

`-speed` *`speed_grade`*

Sets the speed grade for your project. Possible values depend on your device, die and package. See your device datasheet for details.

`-die_voltage` *`value`*

Sets the die voltage for your project. Possible values depend on your device. See your device datasheet for details.

`-adv_options` *`value`*

Sets your advanced options, such as operating conditions.

Value	Description
IO_DEFT_STD:LVTTL	Sets your I/O default value to LVTTL
TEMPR:MIL	Sets your default temperature range; can be COM (Commercial), MIL (Military) or IND (industrial).
VCCI_1.5_VOLTR:COM	Sets VCCI to 1.5 and voltage range to Commercial
VCCI_1.8_VOLTR:COM	Sets VCCI to 1.8 and voltage range to Commercial
VCCI_2.5_VOLTR:COM	Sets VCCI to 2.5 and voltage range to Commercial
VCCI_3.3_VOLTR:COM	Sets VCCI to 3.3 and voltage range to Commercial
VOLTR:COM	Sets your voltage range; can be COM (Commercial), MIL (Military) or IND (industrial).

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Example

Creates a new project in the directory c:/netlists/test named "project", with the HDL type VHDL for the ProASIC3 family.

```
new_project -location C:/Netlists/Test -name Project -hdl VHDL -family PA3
```

See Also

Project Manager Tcl Command Reference

open_project

Tcl command; opens an existing Libero SoC project.

```
open_project project_name-do_backup_on_convert value-backup_file backup_filename
```

Arguments

`project_name`

Must include the complete path to the PRJ file. If you do not provide the full path, Libero SoC infers that you want to open the project from your current working directory.

`-do_backup_on_convert` *`value`*

Sets the option to backup your files if you open a project created in a previous version of Libero SoC.

Value	Description
TRUE	Creates a backup of your original project before opening
FALSE	Opens your project without creating a backup

`-backup_file` *`backup_filename`*

Sets the name of your backup file (if you choose to do_backup_on_convert).

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

- None

Example

Open project.prj from the c:/netlists/test directory.

```
open_project c:/netlists/test/project.prj
```

See Also

`close_project`

`new_project`

`save_project`

Project Manager Tcl Command Reference

organize_cdbs

Tcl command; enables you to organize the CDB files in your project.

```
organize_cdbs -file name -module name
```

Arguments

`-file` *`name`*

Specifies the name of the CDB file you intend to organize.

`-module` *`name`*

Identifies the name of the module to which you wish to add the CDB file.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Example

Adds the file design2.cdb to the module design_test.

```
organize_cdbs -file design2.cdb -module design_test
```

See Also

Project Manager Tcl Command Reference

organize_constraints

Tcl command; organizes the constraint files in your project.

```
-organize_constraints
[-file name]*
[-mode value]
-designer_view name
-module value
-tool value
```

Arguments

`-file name`

Specifies the name of the file to which you want to associate your stimulus files.

`-mode value`

Specifies whether you are creating a new stimulus association, adding, or removing; possible values are:

Value	Description
new	Creates a new stimulus file association
add	Adds a stimulus file to an existing association
remove	Removes an stimulus file association

`-designer_view name`

Sets the name of the Designer View in which you wish to add the constraint file, where name is the name of the view (such as impl1).

`-module value`

Sets the module, where value is the name of the module.

`-tool value`

Identifies the intended use for the file, possible values are:

Value	Description
synthesis	File to be used for synthesis
designer	File to be used in Designer
phsynth	File to be used in physical synthesis

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Example

The example adds the constraint file delta.vhd in the Designer View impl2 for the Designer tool.

```
-organize_constraints -file delta.vhd -mode new -designer_view impl2 -module constraint
-tool designer
```

See Also

Project Manager Tcl Command Reference

organize_sources

Tcl command; organizes the source files in your project.

Arguments

```
-organize_sources
[-file name]*
[-mode value]
-module value
-tool value
[-use_default value]
```

Arguments

`-file name`

Specifies the name of the file to which you want to associate your stimulus files.

`-mode value`

Specifies whether you are creating a new stimulus association, adding, or removing; possible values are:

Value	Description
new	Creates a new stimulus file association
add	Adds a stimulus file to an existing association
remove	Removes an stimulus file association

`-module value`

Sets the module, where value is the name of the module.

`-tool value`

Identifies the intended use for the file, possible values are:

Value	Description
synthesis	File to be used for synthesis
simulation	File to be used for simulation

`-use_default value`

Uses the default values for synthesis or simulation; possible values are:

Value	Description
TRUE	Uses default values for synthesis or simulation.

Value	Description
FALSE	Uses user-defined values for synthesis or simulation

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Example

The example organizes a new stimulus file 'stim.vhd' using default settings.

```
-organize_sources -file stim.vhd -mode new -module stimulus -tool synthesis -use_default
TRUE
```

See Also

Project Manager Tcl Command Reference

project_settings

Tcl command; modifies project flow settings for your Libero SoC project.

```
project_settings [-hdl "VHDL | VERILOG"] [-auto_update_modelsim_ini "TRUE | FALSE"] [-
auto_update_viewdraw_ini "TRUE | FALSE"] [-block_mode "TRUE | FALSE"] [-
auto_generate_synth_hdl "TRUE | FALSE"] [-auto_run_drc "TRUE | FALSE"] [-
auto_generate_viewdraw_hdl "TRUE | FALSE"] [-auto_file_detection "TRUE | FALSE"]
```

Arguments

`-hdl "VHDL | VERILOG"`

Sets your project HDL type.

`-auto_update_modelsim_ini "TRUE | FALSE"`

Sets your auto-update modelsim.ini file option. TRUE updates the file automatically.

`-auto_update_viewdraw_ini "TRUE | FALSE"`

Sets your auto-update viewdraw.ini file option. TRUE updates the file automatically.

`-block_mode "TRUE | FALSE"`

Puts the Project Manager in Block mode, enables you to create blocks in your project.

`-auto_generate_synth_hdl "TRUE | FALSE"`

Auto-generates your HDL file after synthesis (when set to TRUE).

`-auto_run_drc "TRUE | FALSE"`

Auto-runs the design rule check immediately after synthesis (when set to TRUE).

`-auto_generate_viewdraw_hdl "TRUE | FALSE"`

Auto-generates your HDL netlist after a Save & Check in ViewDraw (when set to TRUE).

`-auto_file_detection "TRUE | FALSE"`

Automatically detects when new files have been added to the Libero SoC project folder (when set to TRUE).

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Example

Set your project to VHDL, do not auto-update the ModelSim INI or ViewDraw INI files, auto-generate HDL after synthesis, and enable auto-detect for files.

```
project_settings [-hdl "VHDL"] [-auto_update_modelsim_ini "FALSE"] [-
auto_update_viewdraw_ini "FALSE"] [-block_mode "FALSE"] [-auto_generate_synth_hdl
"TRUE"] [-auto_file_detection "TRUE"]
```

See Also

Project Manager Tcl Command Reference

read_active_probe

Tcl command; reads active probe values from the device. The target probe points are selected by select_active_probe command.

```
read_active_probe [-deviceName device_name] [-file file_path]
```

Arguments

`-deviceName` *`device_name`*

Parameter is optional if only one device is available in the current configuration or set for debug (see SmartDebug user guide for details).

`-file` *`file_path`*

Optional; if specified, re-directs output with probe point values read from the device to the specified file.

Supported Families

SmartFusion2

Exceptions

- You must set a debug file
- Array must be programmed and active
- This command will affect any previously set Live probe channels
- Security locks may disable this function
- Probe points must be selected before calling this operation
- Probe point values are read asynchronously for all selected probe points

Example

Reads the active probe file in /sf2_proj/probe1 on the sf2 device.

```
read_active_probe [-deviceName sf2] [-file /sf2_proj/probe1]
```

See Also

Project Manager Tcl Command Reference

read_lsram

Tcl command; reads a specified block of large SRAM from the device.

```
read_lsram [-deviceName device_name] –block block_name [–file filename]
```

`-deviceName` *`device_name`*

Parameter is optional if only one device is available in the current configuration or set for debug (see SmartDebug help for details).

`-block` *`block_name`*

Specifies the name for the target block.

`-file` *`filename`*

Optional; specifies the output file name for the data read from the device.

Supported Families

SmartFusion2

Exceptions

- You must set a debug file
- Array must be programmed and active
- Security locks may disable this function

Example

Reads the SRAM Block sram_block1 from the sf2 device and writes it to the file sram_block_output.

```
read_lsram [-deviceName sf2] –block sram_block1 [–file sram_block_output]
```

See Also

Project Manager Tcl Command Reference

read_usram

Tcl command; reads a uSRAM block from the device.

```
read_lsram [-deviceName device_name] –block block_name [–file filename]
```

`-deviceName` *`device_name`*

Parameter is optional if only one device is available in the current configuration or set for debug (see SmartDebug help for details).

`-block` *`block_name`*

Specifies the name for the target block.

`-file` *`filename`*

Optional; specifies the output file name for the data read from the device.

Supported Families

SmartFusion2

Exceptions

- You must set a debug file
- Array must be programmed and active
- Security locks may disable this function

Example

Reads the uSRAM Block usram_block2 from the sf2 device and writes it to the file sram_block_output.

```
read_usram [-deviceName sf2] –block usram_block2 [–file sram_block_output]
```

See Also

[Project Manager Tcl Command Reference]()

refresh

Tcl command; refreshes your project, updates the view and checks for updated links and files.

```
refresh .
```

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Example

```
refresh
```

See Also

[Project Manager Tcl Command Reference]()

remove_core

Tcl command; removes a core from your project.

```
remove_core -name core_name
```

Arguments

`-name` *`core_name`*

Name of the core you want to remove.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Example

Remove the core ip-beta2:

```
remove_core -name ip-beta2.ccz
```

See Also

[Project Manager Tcl Command Reference]()

remove_library

Tcl command; removes a VHDL library from your project.

```
remove_library
-library name
```

Arguments

`-library name`

Specifies the name of the library you wish to remove.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Example

Remove (delete) a library called 'my_lib'.

```
remove_library –library my_lib
```

See Also

[Project Manager Tcl Command Reference]

[add_library]

[rename_library]

remove_profile

Tcl command; deletes a tool profile.

```
remove_profile -name  profilename
```

Arguments

`-name  profilename`

Specifies the name of the profile you wish to delete.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Example

The following command deletes the profile 'custom1':

```
remove_profile -name custom1
```

See Also

[Project Manager Tcl Command Reference]

rename_library

Tcl command; renames a VHDL library in your project.

```
rename_library
-library name
 -name name
```

Arguments

`-library name`

Identifies the current name of the library that you wish to rename.

`-name name`

Specifies the new name of the library.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Example

Rename a library from 'my_lib' to 'test_lib1'

```
rename_library –library my_lib -name test_lib1
```

See Also

Project Manager Tcl Command Reference

`add_library`

`remove_library`

rollback_constraints_from_adb

Tcl command; you can enter pin constraints from Project Manager by either using the text editor to add them to a PDC file or by using the I/O Attribute Editor.

Once you have imported I/O constraint files into Designer, you can modify the constraints with the MultiView Navigator. After modifying the constraints, you can import them back into Project Manager to replace the existing constraints.

When you use the Rollback Constraints feature, Project Manager opens the ADB file, exports the PDC file, and then replaces it with the specified PDC file.

```
rollback_constraints_from_adb -adb filename -output output_filename
```

Arguments

`-adb  filename`

Specifies the filename of the ADB file from which you want to rollback constraints.

`-output  output_filename`

Name of the output constraints file.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Example

The following example creates a rollback constraints PDC file called rollback1.pdc from the ADB file designer1.adb:

```
rollback_constraints_from_adb -file designer1.adb -output rollback1
```

See Also

Project Manager Tcl Command Reference

run_designer

Tcl command; runs Designer with compile and layout options (if selected).

```
run_designer [-logfile filename] [-script filename] [-append_commands commands] [-adb value]
[-compile value] [-layout value] [-export_ba value]
```

Arguments

`-logfile filename`

Specifies the filename of your logfile.

`-script filename`

Appends any scripts you wish to add to add to the flow, where filename is the name of the script.

`-append_commands commands`

Appends commands (if any), where commands is the list of appended commands.

`-adb value`

Creates or opens your ADB file. The following table shows the acceptable values for this argument:

Value	Description
new	Creates a new ADB file
open	Opens an existing ADB file
default	Uses the default ADB file in your Libero SoC project

`-compile value`

Compiles your design. The following table shows the acceptable values for this argument:

Value	Description
TRUE	Runs compile
FALSE	Does not run compile, proceeds to the next command

`-layout value`

Runs layout on your design. The following table shows the acceptable values for this argument:

Value	Description
TRUE	Runs layout

Value	Description
FALSE	Does not run layout, proceeds to the next command

`-export_ba value`

Exports back-annotate files for your design. The following table shows the acceptable values for this argument:

Value	Description
TRUE	Exports back-annotate files
FALSE	Does not export back-annotate files; proceeds to the next command

Supported Families

SmartFusion, IGLOO, ProASIC3 and Fusion

Exceptions

None

Example

The following example creates a logfile named designerlog2 and runs compile and layout on the default ADB file created in your Libero SoC project:

```
run_designer [-logfile designerlog2] [-adb default] [-compile TRUE] [-layout TRUE]
```

See Also

Project Manager Tcl Command Reference

run_drc

Tcl command; runs the design rule check on your netlist and generates an HDL file.

```
run_drc [-netlist file] [-gen_hdl value]
```

Arguments

`-netlist file`

Name of the netlist file you want the design rule check to evaluate.

`-gen_hdl value`

Generates an HDL file (if TRUE). The following table shows the acceptable values for this argument:

Value	Description
TRUE	Generates an HDL file for your design
FALSE	Does not generate an HDL file after the design rule check

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Example

Run the design rule check on the netlist named 'dsnr3' and generates the HDL file

```
run_drc [-netlist 'dsnr3'] [-gen_hdl TRUE]
```

See Also

Project Manager Tcl Command Reference

run_simulation

Tcl command; runs simulation on your project with your default simulation tool and creates a logfile.

```
run_simulation [-logfile "name"] [-wlf "name"] [-dofile "name"] [-refresh_lib "value"] [-state "value"]
```

Arguments

`-logfile "name"`

Name of your simulation logfile.

`-wlf "name"`

Name of WLF file you wish to use; this command and the -dofile command are exclusive.

`-dofile "name"`

Name of DO file you wish to use; this command and the -wlf command are exclusive.

`-refresh_lib "value"`

Sets your library refresh option using one of the following values:

Value	Description
TRUE	Refreshes your simulation library
FALSE	Does not refresh your simulation library

`-state "value"`

Identifies which simulation you want to perform.

Value	Description
Pre_Synthesis	Runs pre-synthesis simulation
Post_Synthesis	Runs post-synthesis simulation
Post_Phy_Synthesis	Runs post-synthesis physical simulation
Post_Layout	Runs post-layout simulation

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Example

The following command runs post-layout simulation on your project using the DO file 'myfile.do', does not refresh the simulation library, and creates the logfile 'mylog.log':

```
run_simulation -logfile “Mylog.log” -dofile “Myfile.do” -refresh_lib "TRUE" -state
“Post_Layout”
```

See Also

Project Manager Tcl Command Reference

run_synthesis

run_synthesis

Tcl command; runs synthesis on your project and creates a logfile.

```
run_synthesis [-logfile "name"] [-target "target file name"]
```

Arguments

`-logfile "name"`

Name of your synthesis logfile.

`-target "target file name"`

Name of your synthesis target file.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Example

Run synthesis on your project and create the logfile 'mysynlogfile', and creates the target file 'targfile'.

```
run_synthesis [-logfile "mysynlogfile"] [-target "targfile"]
```

See Also

Project Manager Tcl Command Reference

run_simulation

save_log

Tcl command; saves your Libero SoC log file.

```
save_log -file value
```

Arguments

`-file value`

Value is your name for the new log file.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Example

Save the log file file_log.

```
save_log -file file_log
```

See Also

`close_project`
`new_project`
Project Manager Tcl Command Reference

save_project

Tcl command; the save_project command saves the current project in Libero SoC.

```
save_project
```

Arguments

None

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Example

Saves the project in your current working directory:

```
save_project
```

See Also

`new_project`
`open_project`
Project Manager Tcl Command Reference

save_project_as

Tcl command; the save_project_as command saves the current project in Libero SoC with a different name and in a specified directory. You must specify a location with the -location parameter.

```
save_project_as
-name project_name
-location project_location
-files value
```

```
-designer_views value
-replace_links value
```

Arguments

`-name project_name`

Specifies the name of your new project.

`-location project_location`

Must include the complete path of the PRJ file. If you do not provide the full path, Libero SoC infers that you want to save the project to your current working directory. This is a required parameter.

`-files value`

Specifies the files you want to copy into your new project.

Value	Description
all	Copies all your files into your new project
project	Copies only your Libero SoC project files into your new project
source	Copies only the source files into your new project
none	Copies none of the files into your new project; useful if you wish to manually copy only specific project files

`-designer_views value`

Specifies the Designer views you wish to copy into your new project.

Value	Description
all	Copies all your Designer views into your new project
current	Copies only your current Designer fiew files into your new project
none	Copies none of your views into your new project

`-replace_links value`

Specifies whether or not you want to update your file links in your new project.

Value	Description
true	Replaces (updates) the file links in your project during your save
false	Saves your project without updating the file links

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Example

Saves your current Libero SoC project as mydesign.prj in the c:/netlists/testprj/mydesign directory:

```
save_project_as -location c:/netlists/testprj/mydesign -name mydesign.prj
```

See Also

`new_project`

`open_project`

`save_project`

Project Manager Tcl Command Reference

select_active_probe

Tcl command; manages the current selection of active probe points to be used by active probe READ operations. This command extends or replaces your current selection with the probe points found using the search pattern.

```
select_active_probe [-deviceName device_name] [–name probe_name_pattern] [-reset true|false]
```

Arguments

`-deviceName` *`device_name`*

Parameter is optional if only one device is available in the current configuration or set for debug (see SmartDebug user guide for details).

`-name` *`probe_name_pattern`*

Optional search pattern string that can specify one or multiple probe points. If this string contains an asterisk symbol (*) at the end of the string, the search return all probe points with names that begin with the symbols before the *.

-reset *`true | false`*

Optional parameter; resets all previously selected probe points. If name is not specified, empties out current selection.

Supported Families

SmartFusion2

Exceptions

None

Example

Selects the active probe *_a3 on the device sf2 with reset set to false.

```
select_active_probe [-deviceName sf2] [–name *_a3] [-reset false]
```

See Also

Project Manager Tcl Command Reference

select_profile

Tcl command; selects a profile to use in your project.

```
select_profile -name  profilename
```

Arguments

```
-name  profilename
```

Specifies the name of the profile you wish to use.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Example

The following command selects the profile 'custom1':

```
select_profile -name custom1
```

See Also

Project Manager Tcl Command Reference

set_actel_lib_options

Tcl command; the set_actel_lib_options command sets your simulation library to default, or to another library (when you specify a path).

```
set_actel_lib_options -use_default_sim_path value -sim_path {path}
```

Arguments

```
-use_default_sim_path value
```

Possible values are:

Value	Description
TRUE	Uses the default simulation library.
FALSE	Disables the default simulation library; enables you to specify a different simulation library with the -sim_path {path} option.

```
-sim_path {path}
```

Specifies the path to your simulation library.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Example

Uses a simulation library in the directory c:\sim_lib\test.

```
set_actel_lib_options -use_default_sim_path FALSE -sim_path {c:\sim_lib\test}
```

See Also

Project Manager Tcl Command Reference

set_debug_data_file

Tcl command; sets the debug file. The debug file contains information used by SmartDebug for mapping user design names to their respective physical addresses. It also contains other information used during the debug process. The debug file is generated by Libero during Place and Route and is stored in the /design folder.

The output file name follows the pattern: <design_name>_debug.txt

```
set_debug_data_file –file file_path
```

Arguments

`-file file_path`

Specifies the target debug filename and location. Left slashes in the file pathname are not allowed.

The debug file is only valid for a specific design/device type.

Supported Families

SmartFusion2

Exceptions

None

Example

Sets the debug data file to the sf2_proj/debug directory.

```
set_debug_data_file -file /sf2_proj/debug
```

See Also

Project Manager Tcl Command Reference

set_device (Project Manager)

Tcl command; sets your device family, die, and package in the Project Manager.

```
set_device [-family family] [-die die] [-package package].[-speed speed_grade] [-adv_options
value]
```

Arguments

`-family family`

Sets device family.

`-die die`

Sets device die.

`-package package`

Sets device package.

`-speed speed_grade`

Sets device speed grade.

`-adv_options value`

Sets your advanced options, such as temperature and voltage settings.

Value	Description
IO_DEFT_STD:LVTTL	Sets your I/O default value to LVTTL
TEMPR:COM	Sets your default temperature range; can be COM (Commercial), MIL (Military) or IND (industrial).
VCCI_1.5_VOLTR:COM	Sets VCCI to 1.5 and voltage range to Commercial
VCCI_1.8_VOLTR:COM	Sets VCCI to 1.8 and voltage range to Commercial
VCCI_2.5_VOLTR:COM	Sets VCCI to 2.5 and voltage range to Commercial
VCCI_3.3_VOLTR:COM	Sets VCCI to 3.3 and voltage range to Commercial
VOLTR:COM	Sets your voltage range; can be COM (Commercial), MIL (Military) or IND (industrial).

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Example

Set your device to Fusion, your die to AFS600, and your package to 484 FBGA

```
set_device [-family fusion] [-die afs600] [-package "484 FBGA"]
```

See Also

Project Manager Tcl Command Reference

set_live_probe

Tcl command; set_live_probe channels A and/or B to the specified probe point(s). At least one probe point must be specified. Only exact probe name is allowed (i.e. no search pattern that may return multiple points).

```
set_live_probe [-deviceName device_name] [–probeA probe_name] [–probeB probe_name]
```

Arguments

`-deviceName` *`device_name`*

Parameter is optional if only one device is available in the current configuration or set for debug (see SmartDebug user guide for details).

`-probeA` *`probe_ name`*

Specifies target probe point for the probe channel A.

-probeB *`probe_ name`*

Specifies target probe point for the probe channel B.

Supported Families

SmartFusion2

Exceptions

- You must set a debug file
- The array must be programmed and active
- Active probe read or write operation will affect current settings of Live probe since they use same probe circuitry inside the device
- Setting only one Live probe channel affects the other one, so if both channels need to be set, they must be set from the same call to set_live_probe
- Security locks may disable this function
- In order to be available for Live probe, ProbeA and ProbeB I/O's must be reserved for Live probe respectively

Example

Sets the Live probe channel A to the probe point A12 on device sf2.

```
set_live_probe [-deviceName sf2] [–probeA A12]
```

See Also

Project Manager Tcl Command Reference

set_modelsim_options

Tcl command; sets your Model*Sim* simulation options.

```
set_modelsim_options
[-use_automatic_do_file value]
[-user_do_file {path}]
[-sim_runtime {value}]
[-tb_module_name {value}]
[-tb_top_level_name {value}]
[-include_do_file value
[-included_do_file {value}]
[-type {value}]
[-resolution {value}]
[-add_vsim_options {value}]
[-display_dut_wave value]
[-log_all_signals value]
[-do_file_args value]
[-dump_vcd "TRUE | FALSE"]
[-vcd_file "VCD file name"]
```

Arguments

`-use_automatic_do_file` *`value`*

Uses an automatic.do file in your project. Possible values are:

Value	Description
TRUE	Uses the default automatic.do file in your project.
FALSE	Uses a different *.do file; use the other simulation options to specify it.

`-user_do_file {`*`path`*`}`

Specifies the location of your user-defined *.do file.

`-sim_runtime {`*`value`*`}`

Sets your simulation runtime. Value is the number and unit of time, such as {1000ns}.

`-tb_module_name {value}`

Specifies your testbench module name, where value is the name.

`-tb_top_level_name {value}`

Sets the top-level instance name in the testbench, where value is the name.

`-include_do_file value`

Includes a *.do file; possible values are:

Value	Description
TRUE	Includes the *.do file.
FALSE	Does not include the *.do file

`-included_do_file {value}`

Specifies the name of the included *.do file, where value is the name of the file.

`-type {value}`

Resolution type; possible values are:

Value	Description
min	Minimum
typ	Typical
max	Maximum

-resolution {*value*}

Sets your resolution value, such as {1ps}.

`-add_vsim_options {value}`

Adds more Vsim options, where value specifies the option(s).

-display_dut_wave *value*

Enables ModelSim to display signals for the tested design; possible values are:

Value	Description
0	Displays the signal for the top_level_testbench
1	Enables ModelSim to display the signals for the tested design

-log_all_signals *value*

Enables you to log all your signals during simulation; possible values are:

Value	Description
TRUE	Logs all signals
FALSE	Does not log all signals

`-do_file_args value`

Specifies *.do file command parameters.

`-dump_vcd value`

Dumps the VCD file when simulation is complete; possible values are:

Value	Description
TRUE	Dumps the VCD file
FALSE	Does not dump the VCD file

`-vcd_file {value}`

Specifies the name of the dumped VCD file, where value is the name of the file.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Example

Sets ModelSim options to use the automatic *.do file, sets simulation runtime to 1000ns, sets the testbench module name to "testbench", sets the testbench top level to <top>_0, sets simulation type to "max", resolution to 1ps, adds no vsim options, does not log signals, adds no additional DO file arguments, dumps the VCD file with a name power.vcd.

```
set_modelsim_options -use_automatic_do_file 1 -sim_runtime {1000ns} -tb_module_name
{testbench} -tb_top_level_name {<top>_0} -include_do_file 0 -type {max} -resolution
{1ps} -add_vsim_options {} -display_dut_wave 0 -log_all_signals 0 -do_file_args {} -
dump_vcd 0 -vcd_file {power.vcd}
```

See Also

Project Manager Tcl Command Reference

set_option

Tcl command; sets your synthesis options on a module.

```
set_option [-synth "TRUE | FALSE"] [-module module]
```

Arguments

`-synth "TRUE | FALSE"`

Runs synthesis (for a value of TRUE).

`-module module`

Identifies the module on which you will run synthesis.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Example

Run synthesis on the module test1.vhd:

```
set_option [-synth TRUE] [-module test1.vhd]
```

See Also

Project Manager Tcl Command Reference

set_user_lib_options

Tcl command; sets your user library options during simulation. If you do not use a custom library these options are not available.

```
set_user_lib_options
-name {value}
-path {path}
-option {value}
```

Arguments

```
-name {value}
```

Sets the name of your user library.

```
-path {path}
```

Sets the pathname of your user library.

```
-option {value}
```

Sets your default compile options on your user library; possible values are:

Value	Description
do_not_compile	User library is not compiled
refresh	User library is refreshed
compile	User library is compiled
recompile	User library is recompiled
refresh_and_compile	User library is refreshed and compiled

Supported Familes

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Example

The example below sets the name for the user library to "test1", the path to c:/actel_des_files/libraries/test1, and the compile option to "do not compile".

```
set_user_lib_options -name {test1} -path {c:/actel_des_files/libraries/test1} -option
{do_not_compile}
```

See Also

Project Manager Tcl Command Reference

set_root

Tcl command; sets the module you specify as the root.

```
set_root module_name
```

Arguments

`set_root` *`module_name`*

Specifies the name the module you want to set as root.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Example

Set the module mux8 as root:

```
set_root mux8
```

See Also

Project Manager Tcl Command Reference

synplify

Tcl command; runs Synplify in batch mode and executes a Tcl script.

```
synplify –batch -licensetype synplify_acteloem <Tcl_script>.tcl
```

Arguments

`-batch`

Runs Synplify in batch mode.

`-licensetype synplify_acteloem <Tcl_script>.tcl`

Runs Synplify and executes the Tcl script identified in the brackets; omit the brackets in the final script, as in the example below.

Exceptions

None

Example

The following example runs Synplify in batch mode and executes the Tcl script 'mytcl.tcl'.

```
synplify –batch -licensetype synplify_acteloem mytcl.tcl
```

See Also

Project Manager Tcl Command Reference

synplify_pro

Tcl command; runs Synplify Pro in batch mode and executes a Tcl script.

```
synplify_pro –batch -licensetype synplifypro_acteloem <Tcl_script>.tcl
```

Arguments

`-batch`

Runs Synplify Pro in batch mode.

`-licensetype synplifypro_acteloem <Tcl_script>.tcl`

Runs Synplify Pro and executes the Tcl script identified in the brackets; omit the brackets in the final script, as in the example below.

Exceptions

None

Example

The following example runs Synplify Pro in batch mode and executes the Tcl script 'mytcl.tcl':

```
synplify_pro –batch -licensetype synplifypro_acteloem mytcl.tcl
```

See Also

Project Manager Tcl Command Reference

unlink

Tcl command; removes a link to a file in your project.

```
unlink -file filename [-local local_filename]
```

Arguments

`-file` *`filename`*

Name of the linked (remote) file you want to unlink.

`-local` *`local_filename`*

Name of the local file that you want to unlink.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Example

Unlink the file hdl1.vhd from my local file test.vhd

```
unlink -file hdl1.vhd [-local test.vhd]
```

See Also

Project Manager Tcl Command Reference

use_file

Tcl command; specifies which file in your project to use.

```
use_file
-file value
 -module value
 -designer_view value
```

Arguments

`-filevalue`

Specifies the EDIF or ADB file you wish to use in the project. Value is the name of the file you wish use (including the full pathname).

`-module value`

Specifies the module in which you want to use the file.

`-designer_view value`

Specifies the Designer View in which you wish to use the file.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Example

Specify file1.edn in the ./project/synthesis directory, in the module named top, in the Designer View named impl1.

```
use_file –file “./project/synthesis/file1.edn” –module “top” –designer_view “Impl1”
```

See Also

`use_source_file`

Project Manager Tcl Command Reference

use_source_file

Tcl command; defines a module for your project.

```
use_source_file
-file value
 -module value
```

Arguments

`-file value`

Specifies the Verilog or VHDL file. Value is the name of the file you wish use (including the full pathname).

`-module value`

Specifies the module in which you want to use the file.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Example

Specify file1.vhd in the ./project/hdl directory, in the module named top.

```
use_source_file –file “./project/hdl/file1.vhd” –module “top"
```

See Also

use_file

Project Manager Tcl Command Reference

write_active_probe

Tcl command; sets the target probe point on the device to the specified value; the target probe point name must be specified.

```
write_active_probe [-deviceName device_name] –name probe_name -value true|false
```

Arguments

`-deviceName` *`device_name`*

Parameter is optional if only one device is available in the current configuration or set for debug (see SmartDebug user guide for details).

`-name` *`probe_name`*

Specifies the name for the target probe point.

`-value` *`true|false`*

Specifies values to be written.

Supported Families

SmartFusion2

Exceptions

- You must set a debug file
- Array must be programmed and active
- This command will affect any previously set Live probe channels
- Security locks may disable this function
- If the user clock is running, the target flip-flop keeps the state set by this command for one clock cycle only

Example

Sets the target probe point probe1 on device sf2 to true.

```
write_active_probe [-deviceName sf2] -name probe1 -value true
```

See Also

Project Manager Tcl Command Reference

write_lsram

Tcl command; writes a seven bit word into the specified large SRAM location.

```
write_lsram [-deviceName device_name] –block block_name] –offset offset_value –value value
```

`-deviceName` *`device_name`*

Parameter is optional if only one device is available in the current configuration or set for debug (see SmartDebug help for details).

`-block` *`block_name`*

Specifies the name for the target block.

`-offset` *`offset_value`*

Offset (address) of the target word within the memory block.

`-value` *`value`*

Value to be written to the target location.

Supported Families

SmartFusion2

Exceptions

- You must set a debug file
- Array must be programmed and active
- The maximum value that can be written is 0x1FF
- Security locks may disable this function

Example

Writes a value of 0x1A to the device sf2 in the block sram_block1 with an offset of 16.

```
write_lsram [-deviceName sf2] –block sram_block1 -offset 16 -value 0x1A
```

See Also

Project Manager Tcl Command Reference

write_usram

Tcl command; writes a seven bit word into the specified uSRAM location.

```
write_usram [-deviceName device_name] –block block_name] –offset offset_value –value value
```

`-deviceName` *`device_name`*

Parameter is optional if only one device is available in the current configuration or set for debug (see SmartDebug help for details).

`-block` *`block_name`*

Specifies the name for the target block.

`-offset` *`offset_value`*

Offset (address) of the target word within the memory block.

`-value` *`value`*

Seven bit value to be written.

Supported Families

SmartFusion2

Exceptions

- You must set a debug file

- Array must be programmed and active
- The maximum value that can be written is 0x1FF
- Security locks may disable this function

Example

Writes a value of 0x1A to the device sf2 in the block usram_block2 with an offset of 16.

```
write_usram [-deviceName sf2] –block usram_block2 -offset 16 -value 0x1A
```

See Also

Project Manager Tcl Command Reference

Designer Tcl Command Reference

A Tcl (Tool Command Language) file contains scripts for simple or complex tasks. You can run scripts from either the Windows or UNIX command line or store and run a series of Tcl commands in a ".tcl" batch file. You can also run scripts from within Designer.

Designer supports the following Tcl scripting commands:

Command	Action
add_probe	Adds a probe to an internal net in your design, using the original name from the optimized netlist in your design. Also, this command must be used in conjunction with the generate_probes command to generate a probed ADB file (see example below).
all_inputs	Returns an object representing all input and inout pins in the current design
all_outputs	Returns an object representing all output and inout pins in the current design
all_registers	Returns an object representing register pins or cells in the current scenario based on the given parameters
are_all_source_files_current	Audits all source files and determines whether or not they are out of date / imported into the workspace
backannotate	Extracts timing delays from your post layout data
check_timing_constraints	Checks all timing constraints in the current timing scenario for validity
clone_scenario	Creates a new timing scenario by duplicating an existing one

Command	Action
close_design	Closes the current design
compile	Performs design rule check and optimizes the input netlist before translating the source code into machine code
create_clock	Creates a clock constraint on the specified ports/pins, or a virtual clock if no source is specified
create_generated_clock	Creates an internally generated clock constraint on the ports/pins and defines its characteristics
create_scenario	Creates a new timing scenario with the specified name
delete_probe	Deletes a probe on nets in a probed ADB file
delete_scenario	Deletes the specified timing scenario
export	Converts a file from its current format into the specified file format, usually for use in another program
extended_run_shell	Runs multiple iterations of layout through Designer
generate_probes	Executes the probing and creates a new ADB file. This command is used in conjunction with the add_probe Tcl command (see example below).
get_cells	Returns an object representing the cells (instances) that match those specified in the pattern argument
get_clocks	Returns an object representing the clock(s) that match those specified in the pattern argument in the current timing scenario
get_current_scenario	Returns the name of the current timing scenario
get_defvar	Returns the value of the Designer internal variable you specify
get_design_filename	Returns the fully qualified path of the specified design file

Command	Action
get_design_info	Returns detailed information about your design, depending on which arguments you specify
get_nets	Returns an object representing the nets that match those specified in the pattern argument
get_out_of_date_files	Audits all files returns a list of filenames that are out of date
get_pins	Returns an object representing the pin(s) that match those specified in the pattern argument
get_ports	Returns an object representing the port(s) that match those specified in the pattern argument
import_aux	Imports the specified file as an auxiliary file, which are not audited and do not require you to re-compile the design
import_source	Imports the specified file as a source file, which include your netlist and design constraints
ioadvisor_apply_suggestion	Applies the suggestions for the selected attribute to the selected I/O(s)
ioadvisor_commit	Saves all changes in the I/O Advisor
ioadvisor_restore	Restores the I/O Advisor to the initial state
ioadvisor_restore_initial_value	Sets the current value for the selected attribute and I/Os to the initial value
ioadvisor_set_outdrive	Sets the outdrive for the selected I/Os
ioadvisor_set_outputload	Sets the output load for the selected I/Os
ioadvisor_set_slew	Sets the slew for the selected I/Os
is_design_loaded	Returns True if the design is loaded into Designer; otherwise, returns False
is_design_modified	Returns True if the design has been modified since it was last compiled;

Command	Action
	otherwise, returns False
is_design_state_complete	Returns True if the specified design state is complete (for example, you can inquire as to whether a die and package has been selected for the design); otherwise, returns False
is_source_file_current	Audits the source file and determines whether or not the file is out of date / imported into the workspace
layout	Place-and-route your design
list_clocks	Returns details about all of the clock constraints in the current timing constraint scenario
list_clock_latencies	Returns details about all of the clock latencies in the current timing constraint scenario
list_clock_uncertainties	Returns the list of clock-to-clock uncertainty constraints for the current scenario.
list_disable_timings	Returns the list of disable timing constraints for the current scenario
list_false_paths	Returns details about all of the false paths in the current timing constraint scenario
list_generated_clocks	Returns details about all of the generated clock constraints in the current timing constraint scenario
list_input_delays	Returns details about all of the input delay constraints in the current timing constraint scenario
list_max_delays	Returns details about all of the maximum delay constraints in the current timing constraint scenario
list_min_delays	Returns details about all of the minimum delay constraints in the current timing constraint scenario
list_multicycle_paths	Returns details about all of the multicycle paths in the current timing constraint scenario
list_objects	Returns a list of names of the objects in the specified list

Command	Action
list_output_delays	Returns details about all of the output delay constraints in the current timing constraint scenario
list_scenarios	Returns a list of names of all of the available timing scenarios
new_design	Creates a new design (.adb) file in a specific location for a particular design family such ProASIC3
open_design	Opens an existing design in the Designer software
pin_assign	Assigns the named pin to the specified port but does not lock its assignment.
pin_commit	Saves the pin assignments to the design (*.adb) file.
pin_fix	Locks the pin assignment for the specified port, so the pin cannot be moved during place-and-route.
pin_fix_all	Locks all the assigned pins on the device so they cannot be moved during place-and-route.
pin_unassign	Unassigns a specific pin from a specific port. The unassigned pin location is then available for other ports.
pin_unassign_all	Unassigns all pins from a specific port.
pin_unfix	Unlocks the specified pin from its port.
remove_clock	Removes the specified clock constraint from the current timing scenario
remove_clock_latency	Removes a clock source latency from the specified clock and from all edges of the clock
remove_clock_uncertainty	Removes a clock-to-clock uncertainty from the current timing scenario by specifying either its exact arguments or its ID
remove_disable_timing	Removes a disable timing constraint by specifying its arguments, or its ID

Command	Action
remove_false_path	Removes a false path from the current timing scenario by specifying either its exact arguments or its ID
remove_generated_clock	Removes the specified generated clock constraint from the current scenario
remove_input_delay	Removes an input delay a clock on a port by specifying both the clocks and port names or the ID of the input_delay constraint to remove
remove_max_delay	Removes a maximum delay constraint in the current timing scenario by specifying either its exact arguments or its ID.
remove_min_delay	Removes a minimum delay constraint in the current timing scenario by specifying either its exact arguments or its ID
remove_multicycle_path	Removes a multicycle path constraint in the current timing scenario by specifying either its exact arguments or its ID
remove_output_delay	Removes an ouput delay by specifying both the clocks and port names or the ID of the output_delay constraint to remove
rename_scenario	Renames the specified timing scenario with the new name provided
report	Generates the type of report you specify: Status, Timing, Timer Violations, Flip-flop, Power, Pin, or I/O Bank
report (Activity and Hazards Power Report)	Reads a VCD file and reports transitions and hazards for each clock cycle of the VCD file.
report (Bottleneck) using SmartTime	Creates a bottleneck report
report (Cycle Accurate Power Report)	Reports a power waveform with one power value per clock period or half-period instead of an average power for the whole simulation
Report (Data History)	Reports new features and

Command	Action
	enhancements, bug fixes and known issues for the current release that may impact the power consumption of the design
report (Datasheet) using SmartTime	Creates a datasheet report
Report (Power)	Creates a Power report, which enables you to determine if you have any power consumption problems in your design
Report (Power Scenario)	Creates a scenario power report, which enables you to enter a duration for a sequence of previously defined power modes and calculate the average power consumption and the excepted battery life for this sequence.
report (Timing) using SmartTime	Creates a timing report
report (Timing violations) using SmartTime	Creates a timing violations report
set_clock_latency	Defines the delay between an external clock source and the definition pin of a clock within SmartTime
set_clock_uncertainty	Specifies a clock-to-clock uncertainty and returns the ID of the created constraint if the command succeeded
set_current_scenario	Specifies the timing scenario for the Timing Analyzer to use
save_design	Writes the design to the specified filename
set_defvar	Sets the value of the Designer internal variable you specify >
set_design	Specifies the design name, family and path in which Designer will process the design
set_device	Specifies the type of device and its parameters
set_disable_timing	Disables timing arcs within a cell and returns the ID of the created constraint
set_false_path	Identifies paths that are considered false and excluded from the timing

Command	Action
	analysis in the current timing scenario
set_input_delay	Creates an input delay on a port list by defining the arrival time of an input relative to a clock in the current scenario
set_max_delay	Specifies the maximum delay for the timing paths in the current scenario
set_min_delay	Specifies the minimum delay for the timing paths in the current scenario
set_multicycle_path	Defines a path that takes multiple clock cycles in the current scenario
set_output_delay	Defines the output delay of an output relative to a clock in the current scenario
smartpower_add_new_custom_mode	Creates a new custom mode
smartpower_add_new_scenario	Creates a new scenario
smartpower_add_pin_in_domain	Adds a pin to either a Clock or Set domain
smartpower_change_clock_statistics	Changes the default frequencies and probabilities for a specific domain
smartpower_change_setofpin_statistics	Changes the default frequencies and probabilities for a specific set
smartpower_commit	Saves the changes made in SmartPower to the design file (.adb) in Designer
smartpower_create_domain	Creates a new clock or set domain
smartpower_edit_custom_mode	Edits a custom mode
smartpower_edit_scenario	Edits a scenario
smartpower_initialize_clock_with_constraints	Initializes the clock frequency and the data frequency of a single clock domain with a specified clock name and the initialization options
smartpower_init_do	Initializes the frequencies and probabilities for clocks, registers, set/reset nets, primary inputs, combinational outputs, enables and other sets of pins, and selects a

Command	Action
	mode for initialization
smartpower_init_set_clocks_options	Initializes the clock frequency of all clock domains
smartpower_init_set_combinational_options	Initializes the frequency and probability of all combinational outputs
smartpower_init_set_enables_options	Initializes the clock frequency of all enable clocks with the initialization options
smartpower_init_set_othersets_options	Initializes the frequency and probability of all other sets
smartpower_init_set_primaryinputs_options	Initializes the frequency and probability of all primary inputs
smartpower_init_set_registers_options	Initializes the frequency and probability of all register outputs
smartpower_init_set_set_reset_options	Initializes the frequency and probability of all set/reset nets
smartpower_init_setofpins_values	Initializes the frequency and probability of all sets of pins
smartpower_remove_all_annotations	Removes all initialization annotations for the specified mode
smartpower_remove_custom_mode	Removes a custom mode
smartpower_remove_domain	Removes an existing domain
smartpower_remove_file	Removes a VCD file from the specified mode
smartpower_remove_pin_enable_rate	This command is obsolete and it is replaced by smartpower_remove_pin_probability
smartpower_remove_pin_frequency	Removes the frequency of an existing pin
smartpower_remove_pin_of_domain	Removes a clock pin or a data pin from a Clock or Set domain, respectively.
smartpower_remove_pin_probability	Enables you to annotate the probability of a pin driving an enable pin
smartpower_remove_scenario	Removes a scenario from the current design

Command	Action
smartpower_remove_vcd	Removes an existing VCD file from a mode or entire design
smartpower_restore	Restores previously committed constraints
smartpower_set_battery_capacity	Sets the battery capacity
smartpower_set_cooling	Sets the cooling style to one of the predefined types, or a custom value
smartpower_set_mode_for_analysis	Sets the mode for cycle-accurate power analysis
smartpower_set_operating_condition	Sets the operating conditions used in SmartPower to best, typical, or worst case
smartpower_set_pin_enable_rate	This command is obsolete and it is now replaced by smartpower_set_pin_probability
smartpower_set_pin_frequency	Sets the frequency of an existing pin
smartpower_set_pin_probability	Enables you to annotate the probability of a pin driving an enable pin
smartpower_set_preferences	Sets SmartPower preferences such as power unit, frequency unit, operating mode, operating conditions, and toggle
smartpower_set_scenario_for_analysis	Sets the scenario for cycle-accurate power analysis
smartpower_set_temperature_opcond	Sets the temperature in the operating conditions used in SmartPower
smartpower_set_thermalmode	Sets the mode of computing junction temperature
smartpower_set_voltage_opcond	Sets the voltage in the operating conditions used in SmartPower
smartpower_temperature_opcond_set_design_wide	Sets the temperature for SmartPower design-wide operating conditions
smartpower_temperature_opcond_set_mode_specific	Sets the temperature for SmartPower mode-specific operating conditions
smartpower_voltage_opcond_set_design_wide	Sets the voltage settings for

Command	Action
	SmartPower design-wide operating conditions
smartpower_voltage_opcond_set_mode_specific	Sets the voltage settings for SmartPower mode-specific use operating conditions
st_create_set	Creates a set of paths to be analyzed
st_commit	Saves the changes made in SmartTime to the design (.adb) file
st_edit_set	Modifies the paths in a user set
st_expand_path	Displays expanded path information (path details) for paths
st_list_paths	Displays the list of paths in the same tabular format shown in SmartTime
st_remove_set	Deletes a user set from the design
st_restore	Restores constraints previously committed in SmartTime
st_set_options	Sets options for timing analysis
timer_get_path	Displays the Timer path information in the Log window
timer_get_clock_actuals	Displays the actual clock frequency in the Log window
timer_get_clock_constraints	Displays the clock constraints (period/frequency and dutycycle) in the Log window
timer_get_maxdelay	Displays the maximum delay constraint between two pins of a path in the Log window
timer_get_path_constraints	Displays the path constraints set for maxdelay in the Timer in the Log window
timer_remove_stop	Removes the path stop constraint on the specified pin
timer_restore	Restores previously committed constraints
timer_remove_all_constraints	Removes all the timing constraints previously entered in the Designer

Command	Action
	system

Note: Note: Tcl commands are case sensitive. However, their arguments are not.

See Also

Introduction to Tcl scripting

Basic syntax

Tcl Command Documentation Conventions

The following table shows the typographical conventions used for the Tcl command syntax.

Syntax Notation	Description
`command - argument`	Commands and arguments appear in Courier New typeface.
`variable`	*Variables appear in blue, italic Courier New typeface. You must substitute an appropriate value for the variable.*
[-argument*value*] [*variable*]+	Optional arguments begin and end with a square bracket with one exception: if the square bracket is followed by a plus sign (+), then users must specify at least one argument. The plus sign (+) indicates that items within the square brackets can be repeated. Do not enter the plus sign character.

Note: Note: All Tcl commands are case sensitive. However, their arguments are not.

Examples

Syntax for the get_defvar command followed by a sample command:

```
get_defvar variable
```

```
get_defvar “DESIGN”
```

Syntax for the backannotate command followed by a sample command:

```
backannotate -name file_name -format format_type -language language -dir directory_name [-
netlist] [-pin]
```

```
backannotate -dir \
 {..\design} -name "fanouttest_ba.sdf" -format "SDF" -language "VERILOG" \
-netlist
```

Wildcard Characters

You can use the following wildcard characters in names used in Tcl commands:

Wildcard	What it Does
\	Interprets the next character literally
?	Matches any single character

Wildcard	What it Does
*	Matches any string
[]	Matches any single character among those listed between brackets (that is, [A-Z] matches any single character in the A-to-Z range)

Note: Note: The matching function requires that you add a slash (\) before each slash in the port, instance, or net name when using wildcards in a PDC command and when using wildcards in the Find feature of the MultiView Navigator. For example, if you have an instance named “A/B12” in the netlist, and you enter that name as “A\\/B*” in a PDC command, you will not be able to find it. In this case, you must specify the name as A\\\\/B*.

Special Characters [ ], { }, and \

Sometimes square brackets ([ ]) are part of the command syntax. In these cases, you must either enclose the open and closed square brackets characters with curly brackets ({ }) or precede the open and closed square brackets ([ ]) characters with a backslash (\). If you do not, you will get an error message.

For example:

```
pin_assign -port {LFSR_OUT[0]} -pin 15
or
pin_assign -port LFSR_OUT\[0\] -pin 180
```

Note: Note: Tcl commands are case sensitive. However, their arguments are not.

Entering Arguments on Separate Lines

To enter an argument on a separate line, you must enter a backslash (\) character at the end of the preceding line of the command as shown in the following example:

```
backannotate -dir \
{..\design} -name "fanouttest_ba.sdf" -format "SDF" -language "VERILOG" \
-netlist
```

See Also

Introduction to Tcl scripting

Basic syntax

About Designer Tcl commands

add_probe

Tcl command; adds a probe to an internal net in your design, using the original name from the optimized netlist in your design. Also, this command must be used in conjunction with the generate_probes command to generate a probed ADB file (see example below).

You must complete layout before you use this command.

```
add_probe –net <net_name> [-pin <pin_name>] [-port <port_name>] [-assign_to_used_pin
<TRUE|FALSE>]
```

Arguments

```
-net <net_name>
```

Name of the net you want to probe. You cannot probe HARDWIRED, POWER, or INTRINSIC nets.

```
-pin <pin_name>
```

Name of the package pin at which you want to put the net to be probed. Argument is optional; if unspecified the net is routed to any free package pin.

`-port <port_name>`

Name of the port you are adding. Argument is optional; if unspecified the default value is PROBE_<n>.

`-assign_to_used_pin <TRUE|FALSE>`

Probes a net on an already used pin. The net on the existing pin will be disconnected. Argument is optional; if unspecified the net can be only routed on unused pin.

Supported Families

SmartFusion, IGLOO, ProASIC3 and Fusion

Exceptions

None

Example

The example below adds a probe to the net Count8_0/INV_0_Y on pin 7 and uses the port name PROBE_1, then generates the probe ADB file named test1.adb.

Note that generate_probes is a separate Tcl command.

```
add_probe -net Count8_0/INV_0_Y -assign_to_used_pin {FALSE} -pin {7} -port {PROBE_1}
generate_probes –save test1.adb
```

See Also

`delete_probe`

`generate_probes`

Generating a Probed Design

Generate Probed Design - Add Probe(s) Dialog Box

Designer Tcl Command Reference

all_inputs

Tcl command; returns an object representing all input and inout pins in the current design.

```
all_inputs
```

Arguments

None

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

You can only use this command as part of a –from, -to, or –through argument in the following Tcl commands: set_min_delay, set_max_delay, set_multicycle_path, and set_false_path.

Examples

```
set_max_delay -from [all_inputs] -to [get_clocks ck1]
```

See Also

Tcl documentation conventions

Designer Tcl Command Reference

all_outputs

Tcl command; returns an object representing all output and inout pins in the current design.

```
all_outputs
```

Arguments

None

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

You can only use this command as part of a –from, -to, or –through argument in the following Tcl commands: set_min_delay, set_max_delay, set_multicycle_path, and set_false_path.

Examples

```
set_max_delay -from [all_inputs] -to [all_outputs]
```

See Also

Tcl documentation conventions

Designer Tcl Command Reference

all_registers

Tcl command; returns an object representing register pins or cells in the current scenario based on the given parameters.

```
all_registers [-clock clock_name]
[-async_pins][-output_pins][-data_pins][-clock_pins]
```

Arguments

-clock `clock_name`

Specifies the name of the clock domain to which the registers belong. If no clock is specified, all registers in the design will be targeted.

-async_pins

Lists all register pins that are async pins for the specified clock (or all registers asynchronous pins in the design).

-output_pins

Lists all register pins that are output pins for the specified clock (or all registers output pins in the design).

-data_pins

Lists all register pins that are data pins for the specified clock (or all registers data pins in the design).

-clock_pins

Lists all register pins that are data pins for the specified clock (or all registers clock pins in the design).

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

You can only use this command as part of a –from, -to, or –through argument in the following Tcl commands: set_min_delay, set_max_delay, set_multicycle_path, and set_false_path.

Examples

```
set_max_delay 2.000 -from { ff_m:CLK ff_s2:CLK } -to [all_registers -clock_pins -clock {
ff_m:Q }]
```

See Also

Tcl documentation conventions

Designer Tcl Command Reference

are_all_source_files_current

Tcl command; audits all source files and determines whether or not they are out of date / imported into the workspace. Returns '1' if all source files are current Returns '0' if all source files are not current This command ignores the Audit settings in your ADB file.

```
are_all_source_files_current
```

Arguments

None

Supported Family

SmartFusion, IGLOO, ProASIC3 and Fusion

Exceptions

- The command will return an error if arguments are passed.

Example

The following code will determine if your source files are current.

```
are_all_source_files_current
```

See Also

`get_out_of_date_files`

`is_source_file_current`

Designer Tcl Command Reference

backannotate

Tcl command; equivalent to executing the Back-Annotate command from the Tools menu. You can export an SDF file, after layout, along with the corresponding netlist in the VHDL or Verilog format. These files are useful in backannotated timing simulation.

Best practice is to export both SDF and the corresponding VHDL/Verilog files. This will avoid name conflicts in the simulation tool.

Designer must have completed layout before this command can be invoked, otherwise the command will fail.

```
backannotate -name file_name -format format_type -language language-dir directory_name [-
netlist] [-pin] [-use_emd]
```

Arguments

`-name file_name`

Use a valid file name with this option. You can attach the file extension .sdf to the File_Name, otherwise the tool will append .sdf for you.

`-format format_type`

Only SDF format is available for back annotation

`-language language`

The supported Language options are:

Value	Description
VHDL93	For VHDL-93 style naming in SDF
VERILOG	For Verilog style naming in SDF

`-dir directory_name`

Specify the directory in which all the files will be extracted.

`-netlist`

Forces a netlist to be written. The netlist will be either in Verilog or VHDL.

`-pin`

Designer exports the pin file with this option. The .pin file extension is appended to the design name to create the pin file.

`-use_emd`

Enables Export Enhanced Min Delays for Best Case option in your backannotated file. .

Supported Families

SmartFusion, IGLOO, ProASIC3 and Fusion

Exceptions

None

Examples

Example 1 uses default arguments and exports SDF file for back annotation:

```
backannotate
```

Example 2 uses some of the options for VHDL:

```
backannotate -dir \
 {..\my_design_dir} -name "fanouttest_ba.sdf" -format "SDF" –language \ "VHDL93" –netlist
```

Example 3 uses some of the options for Verilog:

```
backannotate -dir \
 {..\design} -name "fanouttest_ba.sdf" -format "SDF" -language "VERILOG" \
-netlist
```

Example 4 enables you to catch exceptions and respond based on the success of backannotate operation:

```
If  { [catch { backannotate -name "fanouttest_ba" -format "SDF" } ]} {
              Puts “Back annotation failed”
              # Handle Failure
} else {
              Puts “Back annotation successful”
              # Proceed with other operations
}
```

Example 5 enables Export Enhanced Min Delays for Best Case:

```
backannotate -dir \ {..\my_design_dir} -name "fanouttest_ba.sdf" -format "SDF"
–language \ "VHDL93" –netlist -use_emd
```

See Also

Tcl command documentation conventions

Designer Tcl Command Reference

check_timing_constraints

Tcl command; checks all timing constraints in the current timing scenario for validity.

```
check_timing_constraints
```

Arguments

None

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Examples

```
check_timing_constraints
```

See Also

Tcl documentation conventions

Designer Tcl Command Reference

clone_scenario

Tcl command; creates a new timing scenario by duplicating an existing one. You must provide a unique name (that is, it cannot already be used by another timing scenario).

```
clone_scenario name -source origin
```

Arguments

`name`

Specifies the name of the new timing scenario to create.

`-source origin`

Specifies the source of the timing scenario to clone (copy). The source must be a valid, existing timing scenario.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Description

This command creates a timing scenario with the specified name, which includes a copy of all constraints in the original scenario (specified with the -source parameter). The new scenario is then added to the list of scenarios.

Example

```
clone_scenario scenario_A –source {Primary}
```

See Also

[create_scenario]

[delete_scenario]

Tcl documentation conventions

Designer Tcl Command Reference

close_design

Tcl command; closes the current design and brings Designer to a fresh state to work on a new design. This is equivalent to selecting the Close command from the File menu.

```
close_design
```

Arguments

None

Supported Families

SmartFusion, IGLOO, ProASIC3 and Fusion

Exceptions

None

Example

```
if  { [catch { close_design }] {
              Puts “Failed to close design”
              # Handle Failure
} else {
              puts “Design closed successfully”
              # Proceed with processing a new design
}
```

See Also

open_design

close_design

new_design

Designer Tcl Command Reference

compile

Tcl command; compile Tcl arguments available for SmartFusion, IGLOO, ProASIC3 and Fusion families.

```
compile
-pdc_abort_on_error value
-pdc_eco_display_unmatched_objects value
-pdc_eco_max_warnings value
-demote_globals value
-demote_globals_max_fanout value
-promote_globals value
```

```
-promote_globals_min_fanout value
-promote_globals_max_limit value
-localclock_max_shared_instances value
-localclock_buffer_tree_max_fanout value
-combine_register value
-delete_buffer_tree value
-delete_buffer_tree_max_fanout value
-report_high_fanout_nets_limit value
```

Block creation mode only:

```
-block_remove_ios value
-block_add_interface value
-block_add_interface_fanout value
```

Block instantiation mode only:

```
-block_placement_conflicts value
-block_routing_conflicts value
```

Arguments

`-pdc_abort_on_error value`

Changes the “Abort on PDC error” behavior. The following table shows the values for this argument:

Value	Description
ON	Stops the flow if any error is reported in reading your PDC file
OFF	Skips errors in reading your PDC file and just report them as warnings.

Default: ON

Note: Note: The flow always stops in the following two cases (even if this option is OFF):

- If there is a Tcl error (for example, the command does not exist or the syntax of the command is incorrect)
- The assign_local_clock command for assigning nets to LocalClocks fails. This may happen if any floor planning DRC check fails, such as, region resource check, fix macro check (one of the load on the net is outside the LocalClock region). If such an error occurs, then the Compile command fails. Correct your PDC file to proceed.

`-pdc_eco_display_unmatched_objects value`

Displays netlist objects in PDC that are not found in the imported netlist during Compile ECO mode.The following table shows the values for this argument:

Value	Description
ON	Reports netlist objects not found in the current netlist when reading the internal ECO PDC constraints
OFF	Specifies not to report netlist objects not found in the current netlist when reading the internal ECO PDC constraints

Default: OFF

`-pdc_eco_max_warnings value`

Defines the maximum number of errors/warnings in Compile ECO mode.

The value is the maximum number of error/warning messages to be displayed in the case of reading ECO constraints.

Default: 10000

`-demote_globals` *`value`*

Enables/disables global clock demotion of global nets to regular nets. The following table shows the values for this argument:

Value	Description
OFF	Disables global demotion of global nets to regular nets
ON	Enables global demotion of global nets to regular nets

Default: OFF

`-demote_globals_max_fanout` *`value`*

Defines the maximum fanout value of a demoted net;where value is the maximum value
Default: 12

Note: A global net is not automatically demoted (assuming the option is on) if the resulting fanout of the demoted net (if it was demoted) is greater than the max fanout value. Best practice is to set the automatic global demotion so that it only acts on small fanout nets. Drive high fanout nets with a clock network in the design to improve routability and timing.

`-promote_globals` *`value`*

Enables/disables global clock promotion. The following table shows the values for this argument:

Value	Description
ON	Enables global promotion of nets to global clock network
OFF	Disables global promotion of nets to global clock network

Default: OFF

`-promote_globals_min_fanout` *`value`*

Defines the minimum fanout of a promoted net; where *`value`* is the minimum fanout of a promoted net.
Default:200

`-promote_globals_max_limit` *`value`*

Defines the maximum number of nets to be automatically promoted to global The default value is 0. This is not the total number as nets need to satisfy the minimum fanout constraint to be promoted. The promote_globals_max_limit value does not include globals that may have come from either the netlist or PDC file (quadrant clock assignment or global promotion).

Note: Note: Demotion of globals through PDC or Compile is done before automatic global promotion is done.

Note: You may exceed the number of globals present in the device if you have nets already assigned to globals or quadrants from the netlist or by using a PDC file. The automatic global promotion adds globals on what already exists in the design.

`-localclock_max_shared_instances` *`value`*

Defines the maximum number of shared instances allowed to perform the legalization. This option is also available for quadrant clocks.

`value` is the maximum number of instances allowed to be shared by 2 LocalClock nets assigned to disjoint regions to perform the legalization (default is 12, range is 0-1000). If the number of shared instances is set to 0, no legalization is performed.

Note: Note: If you assign quadrant clocks to nets using MultiView Navigator, no legalization is performed.

`-localclock_buffer_tree_max_fanout` *`value`*

Defines the maximum fanout value used during buffer insertion for clock legalization. This option is also available for quadrant clocks.

Set *`value`* to 0 to disable this option and prevent legalization (default value is 12, range is 0-1000). If the value is set to 0, no buffer insertion is performed. If the value is set to 1, there will be one buffer inserted per pin.

`-combine_register` *`value`*

Combines registers at the I/O into I/O-Registers. The following table shows the values for this argument:

Value	Description
ON	Combines registers at the I/O into I/O-Registers
OFF	Does not optimize and combine registers at the I/O.

Default: OFF

`-delete_buffer_tree` *`value`*

Enables/disables buffer tree deletion on the global signals. The buffer and inverter are deleted. The following table shows the values for this argument:

Value	Description
ON	Enables buffer tree deletion from the netlist
OFF	Disables buffer tree deletion from the netlist

Default: OFF

`-delete_buffer_tree_max_fanout` *`value`*

Defines the maximum fanout of a net after buffer tree deletion;

`value` is the maximum value; the default value is 12.

Note: Note: A net does not automatically remove its buffer tree (assuming the option is on) if the resulting fanout of the net (if the buffer tree was removed) is greater than the max fanout value. Best practice is to set the automatic buffer tree deletion only so that acts on small fanout nets. Drive high fanout nets with a clock network in the design to improve routability and timing.

`-report_high_fanout_nets_limit` *`value`*

Enables flip-flop net sections in the compile report and defines the number of nets to be displayed in the high fanout.

Default: 10

Block creation mode only:

`-block_remove_ios` *`value`*

Removes I/Os, if any in the design. Possible values are shown in the table below:

Value	Description
ON	Removes I/Os from the block (if possible)
OFF	Leaves I/Os (if any) unchanged

`-block_add_interface` *`value`*

Adds buffers on ports in the block, no fanout limit. Values shown in the table below:

Value	Description
ON	Adds buffers on ports
OFF	Does not add any buffers to ports

`-block_add_interface_fanout` *value*

Adds buffers on ports in the block whose fanout is greater than <value>. This option is used in conjunction with the -block_add_interface option above.

Block instantiation mode only:

`-block_placement_conflicts` *value*

If there multiple blocks instantiated in your design, Designer uses the placement options to resolve the conflicts. Values shown in the table below:

Value	Description
ERROR	Compile errors out if any instance from a designer block is unplaced. This is the default option.
RESOLVE	If some instances get unplaced for any reason, the remaining non-conflicting elements are unplaced. In other words, if there are any conflicts, nothing from the block is kept.
KEEP	If some instances get unplaced for any reason, the non-conflicting elements remaining are preserved but not locked (you can move them).
LOCK	If some instances get unplaced for any reason, the remaining non-conflicting elements are preserved and locked.

`-block_routing_conflicts` *value*

If there multiple blocks instantiated in your design, Designer uses the routing options to resolve the conflicts. Values shown in the table below:

Value	Description
ERROR	Compile errors out if any preserved net routing in a designer block is deleted.
RESOLVE	If a nets' routing is removed for any reason, the routing for non-conflicting nets is also removed. In other words, if there are any conflicts, no routing from the block is kept
KEEP	If a nets routing is removed for any reason, the routing for the non-conflicting nets is preserved but not locked (so that they can be rerouted).
LOCK	If the routing is removed for any reason, the remaining non-conflicting nets are preserved and locked; they cannot be rerouted. This is the default option.

Exceptions

You cannot instantiate an ARM design and create a User Block.

Examples

```
compile \
    -pdc_abort_on_error "ON" \
    -pdc_eco_display_unmatched_objects "OFF" \
    -pdc_eco_max_warnings 10000 \
    -demote_globals "OFF" \
    -demote_globals_max_fanout 12 \
    -promote_globals "OFF" \
    -promote_globals_min_fanout 200 \
    -promote_globals_max_limit 0 \
    -localclock_max_shared_instances 12 \
    -localclock_buffer_tree_max_fanout 12 \
    -combine_register "OFF" \
    -delete_buffer_tree "OFF" \
    -delete_buffer_tree_max_fanout 12 \
    -report_high_fanout_nets_limit 10
```

See Also

Setting Compile Options

Designer Tcl Command Reference

create_clock

Tcl command; creates a clock constraint on the specified ports/pins, or a virtual clock if no source other than a name is specified.

```
create_clock -period period_value [-name clock_name]
[-waveform> edge_list][source_objects]
```

Arguments

`-period` *`period_value`*

Specifies the clock period in nanoseconds. The value you specify is the minimum time over which the clock waveform repeats. The period_value must be greater than zero.

`-name` *`clock_name`*

Specifies the name of the clock constraint. You must specify either a clock name or a source.

`-waveform` *`edge_list`*

Specifies the rise and fall times of the clock waveform in ns over a complete clock period. There must be exactly two transitions in the list, a rising transition followed by a falling transition. You can define a clock starting with a falling edge by providing an edge list where fall time is less than rise time. If you do not specify -waveform option, the tool creates a default waveform, with a rising edge at instant 0.0 ns and a falling edge at instant (period_value/2)ns.

`source_objects`

Specifies the source of the clock constraint. The source can be ports, pins, or nets in the design. If you specify a clock constraint on a pin that already has a clock, the new clock replaces the existing one. You must specify either a source or a clock name.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Description

Creates a clock in the current design at the declared source and defines its period and waveform. The static timing analysis tool uses this information to propagate the waveform across the clock network to the clock pins of all sequential elements driven by this clock source.

The clock information is also used to compute the slacks in the specified clock domain that drive optimization tools such as place-and-route.

Exceptions

- None

Examples

The following example creates two clocks on ports CK1 and CK2 with a period of 6, a rising edge at 0, and a falling edge at 3:

```
create_clock -name {my_user_clock} -period 6 CK1
create_clock -name {my_other_user_clock} –period 6 –waveform {0 3} {CK2}
```

The following example creates a clock on port CK3 with a period of 7, a rising edge at 2, and a falling edge at 4:

```
create_clock –period 7 –waveform {2 4} [get_ports {CK3}]
```

See Also

`create_generated_clock`

Tcl Command Documentation Conventions

Designer Tcl Command Reference

create_generated_clock

Tcl command; creates an internally generated clock constraint on the ports/pins and defines its characteristics.

```
create_generated_clock [-name name] -source reference_pin [-divide_by divide_factor] [-
multiply_by multiply_factor] [-invert] source
```

Arguments

`-name` *`name`*

Specifies the name of the clock constraint.

`-source` *`reference_pin`*

Specifies the reference pin in the design from which the clock waveform is to be derived.

`-divide_by` *`divide_factor`*

Specifies the frequency division factor. For instance if the *divide_factor* is equal to 2, the generated clock period is twice the reference clock period.

`-multiply_by` *`multiply_factor`*

Specifies the frequency multiplication factor. For instance if the *multiply_factor* is equal to 2, the generated clock period is half the reference clock period.

`-invert`

Specifies that the generated clock waveform is inverted with respect to the reference clock.

`source`

Specifies the source of the clock constraint on internal pins of the design. If you specify a clock constraint on a pin that already has a clock, the new clock replaces the existing clock. Only one source is accepted. Wildcards are accepted as long as the resolution shows one pin.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Description

Creates a generated clock in the current design at a declared source by defining its frequency with respect to the frequency at the reference pin. The static timing analysis tool uses this information to compute and propagate its waveform across the clock network to the clock pins of all sequential elements driven by this source.

The generated clock information is also used to compute the slacks in the specified clock domain that drive optimization tools such as place-and-route.

Examples

The following example creates a generated clock on pin U1/reg1:Q with a period twice as long as the period at the reference port CLK.

```
create_generated_clock -name {my_user_clock} –divide_by 2 –source [get_ports {CLK}]
 U1/reg1:Q
```

The following example creates a generated clock at the primary output of myPLL with a period ¾ of the period at the reference pin clk.

```
create_generated_clock –divide_by 3 –multiply_by 4  -source clk [get_pins {myPLL:CLK1}]
```

See Also

create_clock

Tcl Command Documentation Conventions

Designer Tcl Command Reference

create_scenario

Tcl command; creates a new timing scenario with the specified name. You must provide a unique name (that is, it cannot already be used by another timing scenario).

```
create_scenario name
```

Arguments

name

Specifies the name of the new timing scenario.

Supported Families

IGLOO, ProASIC3, SmartFusion2, SmartFusion, Fusion

Description

A timing scenario is a set of timing constraints used with a design. Scenarios enable you to easily refine the set of timing constraints used for Timing-Driven Place-and-Route, so as to achieve timing closure more rapidly.

This command creates an empty timing scenario with the specified name and adds it to the list of scenarios.

Exceptions

None

Example

```
create_scenario scenario_A
```

See Also

`clone_scenario`

Tcl Command Documentation Conventions

Designer Tcl Command Reference

delete_probe

Tcl command; deletes a probe on nets in a probed ADB file.

```
delete_probe –net <net_name>
```

Arguments

`-net <net_name>`

Name of the net you want to delete.

Supported Families

SmartFusion, IGLOO, ProASIC3 and Fusion

Exceptions

None

Example

The example below deletes the probe on the net Count8_0/INV_0_Y.

```
delete_probe -net Count8_0/INV_0_Y
```

See Also

`add_probe`

Generating a Probed Design

Generate Probed Design - Add Probe(s) Dialog Box

Designer Tcl Command Reference

delete_scenario

Tcl command; deletes the specified timing scenario.

```
delete_scenario name
```

Arguments

`name`

Specifies the name of the timing scenario to delete.

Supported Families

IGLOO, ProASIC3, SmartFusion2, SmartFusion, Fusion

Description

This command deletes the specified timing scenario and all the constraints it contains.

Exceptions

- At least one timing scenario must always be available. If the current scenario is the only one that exists, you cannot delete it.
- Scenarios that are linked to the timing analysis or layout cannot be deleted.

Example

```
delete_scenario scenario_A
```

See Also

create_scenario

Tcl Command Documentation Conventions

Designer Tcl Command Reference

export

Tcl command; saves your design to a file in the specified file format. The required and optional arguments this command takes depends on which file format you specify.

```
export
[-format value]
[-feature value]
[-secured_device value]
[-signature value]
[-pass_key value]
[-aes_key value]
[-from_config_file value]
[-number_of_devices value]
[-from_progfile_type value]
[-target_programmer value]
[-custom_security value]
[-fpga_security_level value]
[-from_security_level value]
[-security_permanent value]{filename}
[-from_program_pages value]
[-from_content value]
[-set_io_state value]
[-efm_block_security {location:X;security_level: value}]
[-efm_content {location:X;source:value}]
[-efm_block {location:X;config_file:{value}}]
[-efm_client {location:X;client: value;mem_file: value}}]
```

Arguments

`-format` *`value`*

Specifies the file format of the file to export. The exported files vary from one device family to another; see the Export help topic for a description of each file type and the list of supported families.

You can export the files listed in the table below using the value.

File Types	Value
Netlist Files	adl
	afl
	edn

File Types	Value
	v
	vhd
Constraint Files	crt
	dcf
	gcf
	sdc
	pdc
	pin
Programming Files	afm
	bit
	bts_stp
	dc (exports a *.dat programming file)
	fus
	isc
	pdb
	1532
	svf
FlashPro Data File	fdb
Debugging Files	bsd
	prb
Timing Files	mod
	sdf
	stf
	tcl
Script Files	tcl
Log Files	log
IBIS Files	ibs
Other Files	cob

File Types	Value
	loc
	seg
Block Files	cdb
	cxf
	v
	vhd

`-feature {value}`

Select the silicon feature(s) you want to program. Possible values for this option are listed in the table below, or the instance-specific program options available only for specific families (as shown in the table below). Best practice is to specify your program parameters for each Embedded Flash Memory Block (EFMB) instance, from 0-3. The instance specific program options replace [-feature {*value*}].

value	Family
{setup_security:on/off}	SmartFusion
{prog_fpga:on/off}	SmartFusion
{prog_from:on/off}	SmartFusion
{prog_nvm:on/off}	SmartFusion
{setup_security}	Fusion
{prog_from}	Fusion
{all}	IGLOO; ProASIC3

In Tcl mode for Fusion, programming all features are turned off by default. If there is -feature {setup_security} or -feature {prog_from} the programming for the corresponding feature is activated.

In Tcl mode for SmartFusion, the programming option is read from the loaded PDB and then updated from the command if the is parameter specified. If programming of specific features is disabled, other parameters related to the feature programming are ignored. For example, if -feature {prog_fpga:off}, then -fdb_file and -fdb_source are ignored.

`-secured_device value`

Specifies whether the device you are programming is secured. You can specify yes or no to enable or disable secured programming.

`-signature value`

Optional argument that identifies and tracks Microsemi SoC designs and devices.

`-pass_key value`

Protects all the security settings for FPGA Array, FlashROM, and Embedded Flash Memory Block. The maximum length of this value is 32 characters. You must use hexadecimal characters for the pass key value.

`-aes_key value`

Decrypts FPGA Array and/or FlashROM and Embedded Flash Memory Block programming file content.

Max length is 32 HEX characters.

`-from_config_file` *`value`*

Specifies the location of the FlashROM configuration file.

`-number_of_devices` *`value`*

Specifies the number of devices you want to program. Applicable only when FlashROM has serialization regions.

`-from_progfile_type` *`value`*

Applicable only when FlashROM has serialization regions and STAPL file generation. Possible values:

Value	Description
single	Generates one programming file with all the generated incremental value(s) in the external source file
multiple	Generates one individual programming file for each generated incremental value(s) in the external source file

`-target_programmer` *`value`*

Applicable only when FlashROM has serialization regions and STAPL file generation. Possible values:

Value	Description
specific	Silicon Sculptor, BP Auto Programmer, or FlashPro
generic	Generic STAPL player

`-custom_security` *`value`*

Possible values:

Value	Description
yes	Custom security level
no	Standard security level

`-fpga_security_level` *`value`*

Possible values:

Value	Description
write_verify_protect	The security level is medium (standard) and the FPGA Array cannot be written or verified without a Pass Key
write_protect	The security level is write protected. The FPGA Array cannot be written without a Pass Key, but it is open for verification (custom FPGA)
encrypted	The security level is high (standard) and uses a 128-bit AES encryption
none	The FPGA Array can be written and verified without a Pass Key

`-from_security_level` *`value`*

Possible values:

Value	Description
write_verify_protect	The security level is medium (standard) and the FlashROM cannot be read, written or verified without a Pass Key
write_protect	The security level is write protected. The FlashROM cannot be written without a Pass Key, but it is open for reading and verification (custom FlashROM)
encrypted	The security level is high (standard) and uses a 128-bit AES encryption
none	The FlashROM can be written and verified without a Pass Key

`-security_permanent` *`value`*

Specifies whether the security settings for this file are permanent or not. Possible values:

Value	Description
yes	Permanently disable future modification of security settings for FPGA Array and FlashROM
no	Enable future modifications for FPGA Array and FlashROM

`-from_program_pages` *`"value"`*

Specifies FROM program pages in FlashPoint. If you use FlashROM content from an ADB file and do not specify a value, FlashPoint uses the same pages that were selected for programming in the previous FlashPoint session. Value may be a sequence of page numbers ("123") without a delimiter, or you can use any character or space as a delimiter, as in -from_program_pages "1 2 3".

You must specify pages for programming if you want FlashROM content from the UFC file.

`-from_content "`*`value"`*

Identifies the source file for the FlashROM content- a UFC or ADB file.

If this Tcl parameter is missing, FlashPoint tries to use the ADB as a source of FROM configuration and content data.

Values are shown in table below:

Value	Description
adb	(default)FROM content is taken from your ADB. Configurations from your UFC and ADB files are not compared.
ufc	FlashPoint uses FROM configuration and FROM content from the specified UFC file

`-set_io_state` *`value`*

Sets the I/O state during programming by port name or pin number. You can also use this argument to save or load an IOS file.

To set the I/O by port name, use `-set_io_state {portName:<name>; state:<state>}`. To set the I/O port by pin number, use `-set_io_state {pinNumber:<number>; state:<state>}`. To set all I/Os to the specified state, use `-set_io_state {all; state:<state>}`.

To set BSR values for an I/O, use `-set_io_state { pinNumber:<pin>; input:<state>; output_enable:<state>;output:<state> }`. See the Boundary Scan Registers - Show BSR Details section of the FlashPoint help for more information on setting Boundary Scan Registers in your device.

The following table shows the possible values for this option if you have NOT set BSR values.

Value	Description
Z	Tri-State - Sets the I/O state to tristate
Last Known State	Sets the I/O to the last known state
1	High - Sets the I/O state to high
0	Low - Sets the I/O state to low

The following table shows the possible values for this option if you have set custom BSR values.

Value	Description
Last State	Sets the I/O to the last known state
1	High - Sets the I/O state to high
0	Low - Sets the I/O state to low

To save an IOS file use the argument `-set_io_state { save:<filepath> }`

To load an IOS file, use the argument `-set_io_state { load:<filepath> }`

`-efm_block_security{location:`*`X`*`;security_level: `*`value`*`}`

This option is available only for Fusion; this argument only applies when programming the security settings (setup_security) or programming previously secured devices.

'X' identifies an Embedded Flash Memory Block instance from 0-3.

Possible values for security_level:

Value	Description
clients_jtag_protect	Enables eNVM client JTAG protection; a pass key is required for this option
write_verify_protect	The security level is medium (standard) and the Embedded Flash Memory Block cannot be read, written or verified without a Pass Key
write_protect	The security level is write protected. The Embedded Flash Memory Block cannot be written without a Pass Key, but it is open for reading (custom FB)
encrypted	The security level is high (standard) and uses a 128-bit AES encryption
none	The Embedded Flash Memory Block can be written and read without a Pass Key

`-efm_content {location:X;source: value}`

This option is available only for Fusion; X identifies an Embedded Flash Memory Blockinstance from 0-3. Option identifies the source file for the Embedded Flash Memory Block content, either an EFC or ADB file.

If you wish to program the entire Embedded Flash Memory Block including all its clients that were programmed in previous sessions, and use ADB content for this client, this is the only parameter you must specify. If you wish to program the entire Embedded Flash Memory Block including all its clients and use the Embedded Flash Memory Block map file (EFC) you also have to specify the –efm_block parameter.

Possible values:

Value	Description
adb	(default) Embedded Flash Memory Block content is taken from your ADB
efc	FlashPoint uses the Embedded Flash Memory Block instance configuration and content from the EFC file specified in -efm_block_parameter

`-efm_block {location:X;source: value}`

This option is available only for Fusion; X identifies an Embedded Flash Memory Block (EFMB) instance from 0-3.

Config_file specifies the location of the EFMB instance configuration file (must be an EFC file with full pathname).

`-efm_client {location:X;client:value; mem_file: value}`

This option is available only for Fusion; X identifies an EFMB instance from 0-3.

You must specify the client name and its memory content file for each client of EFMB you wish to program.

Mem_file specifies the file with the memory content for the client. If a mem_file path is specified, the memory content from this file will overwrite the client content in ADB or EFC (as defined by the -efm_content argument). If the client memory file is not specified, the client memory content from the ADB or EFC file is used instead (as defined by the -efm_content argument).

`{filename}`

Specifies the path and name of the file you are exporting.

Supported Families

SmartFusion, IGLOO, ProASIC3 and Fusion

Notes

- None

Exceptions

- None

Examples

```
export -format "bts_stp"
-feature "all"
-secured_device “no”
-signature "123"
-pass_key "FB318707864EC889AE2ED8904B8EB30D"
-custom_security "no"
-fpga_security_level "write_verify_protect"
```

```
-from_security_level "write_verify_protect"
-from_config_file {.\g3_test\from.ufc}
-number_of_devices "1"
-from_progfile_type "single"
-target_programmer "specific" \
 {.\flp4.stp}
```

The following example uses the -set_io_state argument:

```
export \
-format "pdb " \
-feature "setup_security" \
-secured_device "no" \
-custom_security "no" \
-security_level "write_verify_protect" \
-security_permanent "no" \
-pass_key "012EB311B02E4C9A150B0F2BD8861CA0" \
-set_io_state { portName:AG9; state:Low} \
-set_io_state { pinNumber:AG10; state:High} \
-set_io_state { pinNumber:197; state:Tri-State} \
-set_io_state { pinNumber:198; state:Low} \
-set_io_state { pinNumber:199; state:Last Known State} \
{D:/designs/Fusion/DESIGN77}
```

The following example exports a DAT file for programming:

```
export -format "dc" -feature "prog_fpga" {./top.dat}
```

Fusion example 1:

Export soc.pdb file that includes programming data for three clients of EFM block 0. EFM block configuration file ./fus_new/nvm_simple/nvm_simple.efc and clients memory files are used for generating the programming file. Clients specified as TCL parameters must be included in EFC file.

```
export -format "pdb "
-efm_content {location:0; source:efc} \
-efm_block {location:0; config_file:{./fus_new/nvm_simple/nvm_simple.efc}} \
-efm_client {location:0; client:cfiData;
mem_file:{./fus_new/nvm_exmp/input_memfiles/ram1_block_0_ram1_R0C0.mem}} \
-efm_client {location:0; client:dataStorage;
mem_file:{./fus_new/nvm_exmp/input_memfiles/datast2_asb1_smtr_ram.hex}} \
-efm_client {location:0; client:init1;
mem_file:{./fus_new/nvm_exmp/input_memfiles/datast1_asb1_acm_rtc_ram.hex}} \
 {./soc}
```

Fusion example 2:

Export soc.stp and soc.pdb files that include programming data for EFM block 0. Information regarding block configuration, which clients to program, and their memory content is taken from ADB file.

```
export -format "pdb bts_stp"
-efm_content {location:0; source:adb} \
 {./soc}
```

Fusion example 3:

Export soc.stp and soc.pdb files that include programming data for client cfiData of EFM block 0. Other clients of block 0 are not selected to be programmed. ADB file is a source for block configuration and content; EFC is ignored.

```
export -format "pdb"
-efm_content {location:0; source:adb} \
-efm_block {location:0; config_file:{./fus_new/nvm_simple/nvm_simple.efc}} \
-efm_client {location:0; client:cfiData;} \
 {./soc}
```

See Also

Exporting files

Importing files

Tcl documentation conventions

Designer Tcl Command Reference

export (Designer Block support for IGLOO, Fusion and ProASIC3 Families)

Tcl command; exports (publishes) the Designer Block files to a specified directory, includes any added comments.

```
export -format "block"
-export_directory {value} \
-export_name "blockname" \
-placement "value"\
-routing "value"\
-comment "value" \
-export_language "value"\
-region "value"
```

Arguments

`-export_directory {value}`

Specifies the directory name for the exported *.v, *.vhd, *.cxf and *.cdb files. Value is the path and name of the directory

`-export_name "blockname"`

Specifies the prefix of the exported *.v, *.vhd, *.cxf, and *.cdb files, where blockname is the name of the prefix.

`-placement "value"`

Exports placement information. Possible values:

Value	Description
yes	Exports the placement information. Specify "yes" only if the placer state is valid and -placement is specified as "yes."
no	Do not export the placement information.

`-routing "value"`

Exports placement information. Possible values:

Value	Description
yes	Exports routing information. Specify "yes" only if the routing state is valid and -placement is specified as "yes."
no	Do not export the routing information.

`-comment "value"`

Adds comments to document the block.

```
-export_language "value"
```

Specifies the export format of the CXF file for Libero SoC. Possible values:

Value	Description
VERILOG	CXF file is Verilog.
VHDL	CXF file is VHDL.

```
-region "value"
```

Option to publish all the user regions and make them available when you instantiate the block. Possible values:

Value	Description
YES	Publishes all the user regions, makes them available when you instantiate your block.
NO	Disables region publishing

Supported Families

SmartFusion, IGLOO, ProASIC3 and Fusion

Exceptions

None

Example

```
export -format "block" -export_directory {.} -export_name "test_core" -placement "yes" -
routing "yes" -comment "toto" -export_language "VERILOG"
```

See Also

Exporting files

Importing files

Tcl documentation conventions

Designer Tcl Command Reference

generate_probes

Tcl command; executes the probing and creates a new ADB file. This command is used in conjunction with the add_probe Tcl command (see example below).

```
generate_probes –save <ADB_file_name>
```

Arguments

```
-save <ADB_file_name>
```

Name of the new ADB file with your probed nets.

Supported Families

SmartFusion, IGLOO, ProASIC3 and Fusion

Exceptions

None

Example

The example below adds a probe to the net net2 on pin 4 and port prb2 with the add_probe command, and generates the new ADB file test1.adb.

```
add_probe –net net2 –pin 4 –port prb2
generate_probes –save test1.adb
```

See Also

add_probe

Generating a Probed Design

Generate Probed Design - Add Probe(s) Dialog Box

Designer Tcl Command Reference

get_cells

Tcl command; returns an object representing the cells (instances) that match those specified in the pattern argument.

```
get_cells pattern
```

Arguments

```
pattern
```

Specifies the pattern to match the instances to return. For example, "get_cells U18*" returns all instances starting with the characters "U18", where “*” is a wildcard that represents any character string.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Description

This command returns a collection of instances matching the pattern you specify. You can only use this command as part of a –from, -to, or –through argument in the following Tcl commands: set_max delay, set_multicycle_path, and set_false_path.

Exceptions

None

Examples

```
set_max_delay 2 -from [get_cells {reg*}] -to [get_ports {out}]
set_false_path –through [get_cells {Rblock/muxA}]
```

See Also

get_clocks

get_nets

get_pins

get_ports

Tcl Command Documentation Conventions

Designer Tcl Command Reference

get_clocks

Tcl command; returns an object representing the clock(s) that match those specified in the pattern argument in the current timing scenario.

```
get_clocks pattern
```

Arguments

pattern

Specifies the pattern to use to match the clocks set in SmartTime or Timer.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Description

- If this command is used as a –from argument in either the set maximum (set_max_delay), or set minimum delay (set_min_delay), false path (set_false_path), and multicycle constraints (set_multicycle_path), the clock pins of all the registers related to this clock are used as path start points.
- If this command is used as a –to argument in either the set maximum (set_max_delay), or set minimum delay (set_min_delay), false path (set_false_path), and multicycle constraints (set_multicycle_path), the synchronous pins of all the registers related to this clock are used as path endpoints.

Exceptions

None

Example

```
set_max_delay -from [get_ports datal] -to \
[get_clocks ck1]
```

See Also

create_clock

create_generated_clock

Tcl Command Documentation Conventions

Designer Tcl Command Reference

get_current_scenario

Tcl command; returns the name of the current timing scenario.

```
get_current_scenario
```

Arguments

None

Supported Families

IGLOO, ProASIC3, SmartFusion2, SmartFusion, Fusion

Exceptions

None

Examples

```
get_current_scenario
```

See Also

set_current_scenario

Tcl documentation conventions

Designer Tcl Command Reference

get_defvar

Tcl command; provides access to the internal variables within Designer and returns its value. This command also prints the value of the Designer variable on the Log window.

```
get_defvar variable
```

Arguments

variable

The Designer internal variable.

Supported Families

SmartFusion, IGLOO, ProASIC3 and Fusion

Exceptions

None

Example

Example 1: Prints the design name on the log window.

```
   get_defvar “DESIGN”
set variableToGet "DESIGN"
set valueOfVariable [get_defvar $variableToGet]
puts "The value is $valueOfVariable"
```

See Also

set_defvar

Designer Tcl Command Reference

get_design_filename

Tcl command; retrieves the full qualified path of the design file. The result will be an empty string if the design has not been saved to disk. This command is equivalent to the command “get_design_info DESIGN_PATH.” This command predates get_design_info and is supported for backward-compatibility.

```
get_design_filename
```

Arguments

None

Supported Families

SmartFusion, IGLOO, ProASIC3 and Fusion

Exceptions

- The command will return an error if a design is not loaded.
- The command will return an error if arguments are passed.

Example

```
if { [ is_design_loaded ] } {
  set design_location [ get_design_filename ]
  if {$design_location != "" } {
    puts “Design is at $design_location.”
  } else {
    puts “Design has not been saved to a file on disk.”
  }
} else {
  puts "No design is loaded."
}
```

See Also

`get_design_info`
`is_design_loaded`
`is_design_modified`
`is_design_state_complete`
Designer Tcl Command Reference

get_design_info

Tcl command; retrieves some basic details of your design. The result value of the command will be a string value.

```
get_design_info value
```

Arguments

value

Must be one of the valid string values summarized in the table below:

Value	Description
name	Design name. The result is set to the design name string.
family	Silicon family. The result is set to the family name.
design_path	Fully qualified path of the design file. The result is set to the location of the .adb file. If a design has not been saved to disk, the result will be an empty string. This command replaces the command get_design_filename.
design_folder	Directory (folder) portion of the design_path.

Value	Description
design_file	Filename portion of the design_path.
cwdir	Current working directory. The result is set to the location of the current working directory
die	Die name. The result is set to the name of the selected die for the design. If no die is selected, this is an empty string.
Package	Package. The result is set to the name of the selected package for the design. If no package is selected, this is an empty string.
Speed	Speed grade. The result is set to the speed grade for the design. If no speed grade is selected, this is an empty string.

Supported Family

SmartFusion, IGLOO, ProASIC3 and Fusion

Exceptions

- Returns an error if a design is not loaded.
- Returns an error if more than one argument is passed.
- Returns an error if the argument is not one of the valid values.

Example

The following example uses get_design_info to display the various values to the screen.

```
if { [ is_design_loaded ] } {
  puts "Design is loaded."
  set bDesignLoaded 1
} else {
  puts "No design is loaded."
  set bDesignLoaded 0
}
if { $bDesignLoaded != 0 } {
  set var [ get_design_info NAME ]
  puts "  DESIGN NAME:\t$var"
  set var [ get_design_info FAMILY ]
  puts "  FAMILY:\t$var"
  set var [ get_design_info DESIGN_PATH ]
  puts "  DESIGN PATH:\t$var"
  set var [ get_design_info DESIGN_FILE ]
  puts "  DESIGN FILE:\t$var"
  set var [ get_design_info DESIGN_FOLDER ]
  puts "  DESIGN FOLDER:\t$var"
  set var [ get_design_info CWDIR ]
  puts "  WORKING DIRECTORY:  $var"
  set var [ get_design_info DIE ]
  puts "  DIE:\t$var"
  set var [ get_design_info PACKAGE ]
  puts "  PACKAGE:\t'$var'"
  set var [ get_design_info SPEED ]
  puts "  SPEED GRADE:\t$var"
  if { [ is_design_modified ] } {
    puts "The design is modified."
  } else {
    puts "The design is unchanged"
  }
}
puts "get_design.tcl done"
```

See Also

get_design_filename

is_design_loaded

is_design_modified

is_design_state_complete

Designer Tcl Command Reference

get_nets

Tcl command; returns an object representing the nets that match those specified in the pattern argument.

```
get_nets pattern
```

Arguments

pattern

Specifies the pattern to match the names of the nets to return. For example, "get_nets N_255*" returns all nets starting with the characters "N_255", where “*” is a wildcard that represents any character string.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Description

This command returns a collection of nets matching the pattern you specify. You can only use this command as source objects in create clock (create_clock) or create generated clock (create_generated_clock) constraints and as `-through` arguments in the set false path, set minimum delay, set maximum delay, and set multicycle path constraints.

Exceptions

None

Examples

```
set_max_delay 2 -from [get_ports RDATA1] -through [get_nets {net_chkp1 net_chkqi}]
set_false_path –through [get_nets {Tblk/rm/n*}]
create_clock -name mainCLK -period 2.5 [get_nets {cknet}]
```

See Also

create_clock

create_generated_clock

`set_false_path`
`set_min_delay`
`set_max_delay`
`set_multicycle_path`
Tcl documentation conventions
Designer Tcl Command Reference

get_out_of_date_files

Tcl command; audits all files returns a list of filenames that are out of date; each filename is separated by a space. The command returns a string of file names that are out of date separated by a space
i.e. file1 file2 ...

It returns empty string if all files are current.

This command ignores the Audit settings in your ADB file.

```
get_out_of_date_files
```

Arguments

None

Supported Family

SmartFusion, IGLOO, ProASIC3 and Fusion

Exceptions

None

Example

The following code returns a list of filenames that are out of date.

```
get_out_of_date_files
```

See Also

`are_all_source_files_curent`
`is_source_file_current`
Designer Tcl Command Reference

get_pins

Tcl command; returns an object representing the pin(s) that match those specified in the pattern argument.

```
get_pins pattern
```

Arguments

`pattern`

Specifies the pattern to match the pins to return. For example, "get_pins clock_gen*" returns all pins starting with the characters "clock_gen", where “*” is a wildcard that represents any character string.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Example

```
create_clock -period 10 [get_pins clock_gen/reg2:Q]
```

See Also

create_clock

create_generated_clock

set_clock_latency

set_false_path

set_min_delay

set_max_delay

set_multicycle_path

Tcl documentation conventions

Designer Tcl Command Reference

get_ports

Tcl command; returns an object representing the port(s) that match those specified in the pattern argument.

```
get_portspattern
```

Argument

pattern

Specifies the pattern to match the ports.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Example

```
create_clock -period 10 [get_ports CK1]
```

See Also

create_clock

set_clock_latency

set_input_delay

set_output_delay

set_min_delay

set_max_delay

set_false_path

set_multicycle_path

Tcl documentation conventions

Designer Tcl Command Reference

import_aux

Tcl command; imports the specified auxiliary file into the design. Equivalent to executing the Import Auxiliary Files command from the File menu.

```
import_aux
-format file_type-partial_parse value
-start_time value
-end_time value
-auto_detect_top_level_name value
-top_level_name value
-glitch_filtering value
-glitch_threshold value
filename
```

Arguments

`-format file_type`

Specifies the file format of the file to import. You can import one of the following types of files: pdc, sdc, pin, dcf, saif, vcd, or crt.

`-partial_parse {value}`

Specifies whether to partially parse the *.vcd file. The following table shows the acceptable values for this argument:

Value	Description
true	Partially parses the *.vcd file
false	Does not partially parse the *.vcd file

`-start_time {value}`

This option is available only if `-partially_parse` is set to `true`. Specifies the start time (in ns) to partially parse the *.vcd file.

`-end_time {value}`

This option is available only if `-partially_parse` is set to `true`. Specifies the end time (in ns) to partially parse the *.vcd file.

`-auto_detect_top_level_name {value}`

Specifies whether to automatically detect the top-level name. The following table shows the acceptable values for this argument:

Value	Description
true	Automatically detects the top-level name
false	Does not automatically detect the top-level name

`-top_level_name top_level_name`

Specifies the instance name of your design in the simulation testbench when you import a VCD or SAIF file.

When importing a VCD file, the automatic *top_level_name* detection is available. If the -top_level_name option is not specified, SmartPower will try to automatically detect the top level name.

When importing a SAIF file, the automatic *top_level_name* detection is not available and -top_level_name is a required argument.

To identify the top_level_name for SAIF and VCD files manually, refer to Importing a VCD file and Importing a SAIF file.

`-glitch_filtering {value}`

Specifies whether to use glitch filtering. The following table shows the acceptable values for this argument:

Value	Description
true	Glitch filtering is on
auto	Enables automatic glitch filtering. This option will ignore any value specified in `-glitch_threshold`
false	Glitch filtering is off

`-glitch_threshold {value}`

This option is only available when `-glitch_filtering` is set to *`true`*. Specifies the glitch filtering value in ps.

`filename`

Specifies the name of the auxiliary file to import.

Supported Families

SmartFusion, IGLOO, ProASIC3 and Fusion

Description

- Auxiliary files are not audited and are handled as one-time data-entry or data-change events, similar to entering data using one of the interactive editors (for example, PinEditor or Timer).
- If you import the SDC file as an auxiliary file, you do not have to re-compile your design. However, auditing is disabled when you import auxiliary files, and Designer cannot detect the changes to your SDC file(s) if you import them as auxiliary files.

Exceptions

None

Examples

```
import_aux -format sdc file.sdc
import_aux -format pdc file.pdc
import_aux -format vcd file.vcd // automatic detection of top level name
import_aux -format vcd -glitch_filter 10 // filter out glitches that are 10 ps or less
import_aux -format saif -top_level_name "top" file.saif
```

See Also

`import_source`

Importing auxiliary files

Importing source files

Importing files

Tcl documentation conventions

Designer Tcl Command Reference

import_source

Tcl command; imports the specified source file into the design. Equivalent to executing the Import Source Filecommand from the File menu in Designer.

All source files must be specified on one command line.

```
import_source [-merge_timing value][-merge_physical value][-merge_all value][-format
file_type][-abort_on_error value][-top_entity][-edif -edif_flavor value]filename
```

Arguments

`-merge_timing value`

Specifies whether to preserve all existing timing constraints when you import an SDC file. Same as selecting or unselecting the "Keep existing timing constraints" check box in the Import Files dialog box. The following table shows the acceptable values for this option:

Value	Description
yes	Designer merges the timing constraints from the imported SDC file with the existing constraints saved in the constraint database. If there is a conflict, the new constraint has priority over the existing constraint.
no	All existing timing constraints are replaced by the constraints in the newly imported SDC file.

`-merge_physical value`

Specifies whether to preserve all existing physical constraints when you import a GCF or PDC file. Same as selecting or unselecting the "Keep existing physical constraints" check box in the Import Files dialog box. The following table shows the acceptable values for this option:

Value	Description
yes	Designer preserves all existing physical constraints that you have entered either using one of the MVN tools (ChipPlanner, PinEditor, or the I/O Attribute Editor) or a previous GCF or PDC file. The software resolves any conflicts between new and existing physical constraints and displays the appropriate message.
no	All existing physical constraints are replaced by the constraints in the newly imported GCF or PDC file.

`-merge_all value`

Specifies whether to preserve all existing physical and timing constraints when you import an SDC and/or a PDC file. Same as selecting or unselecting the "Keep existing physical constraints" and "Keep existing timing constraints" check boxes in the Import Files dialog box. The following table shows the acceptable values for this option:

Value	Description
yes	Designer preserves all existing physical constraints that you have entered either using one of the MVN tools (ChipPlanner, PinEditor, or the I/O Attribute Editor) or a previous GCF or PDC file. The software resolves any conflicts between new and existing physical constraints and displays an appropriate message. Any existing timing constraints from your ADB are merged with the new information from your imported files. New constraints override any existing timing constraints whenever there is a conflict
no	All the physical constraints in the newly imported GCF or PDC files are

Value	Description
	used. All pre-existing physical constraints are lost. Existing timing constraints from the ADB are replaced by the new timing constraints from your imported file.

`-format` *`file_type`*

Specifies the file format of the file to import. You can import one of the following types of files: adl, edif, verilog, vhdl, gcf, pdc, sdc, or crt.

Note: Note: Refer to Importing source files to know the formats supported for each family.

`-abort_on_error` *`value`*

Aborts a PDC file if it encounters an error during import. Possible values are

Value	Description
yes	Designer aborts on error.
no	Designer ignores the error and continues.

`-top_entity`

Specifies the top entity to a VHDL file.

`-edif edif_flavor` *`value`*

Specifies the type of netlist. It can be edif, viewlogic, or mgc.

`filename`

Specifies the name of the source file to import.

Supported Families

SmartFusion, IGLOO, ProASIC3 and Fusion

Exceptions

Your script -merge options vary according to family as shown below:

- The -merge_timing, -merge_physical, and -merge_all arguments are available for IGLOO, Fusion and ProASIC3 families.
- For IGLOO, Fusion, ProASIC3:

```
import_source -merge_physical yes/no -merge_timing yes/no ...
import_source -merge_all yes/no  ...
import_source -merge yes/no  ...
```

The -merge_all and -merge options map to both -merge_physical and -merge_timing options for these families.

Examples

Consider the following sample scripts:

```
import_source  \
 -merge_physical "no" \
 -merge_timing "yes"
  -format "EDIF" -edif_flavor "GENERIC" \
 {.\designs\mydesign.edn} \
 -format "sdc" \
 {.\designs\mydesign.sdc} \
 -format "pdc" -abort_on_error "no" \
 {.\designs\mydesign.pdc}

import_source \
 -merge_physical "no" \
 -format "verilog" \
 {mydesign.v}

import_source \
  -merge_physical "no" \
  -merge_timing "no" \
  -format "vhdl" -top_entity "aclass" \
   {C:/mynetlist.vhd}

import_source \
  -merge_physical "no" \
  -merge_timing "no" \
  -format "adl" {mydesign.adl}
```

See Also

import_aux

Importing auxiliary files

Importing source files

Importing files

Tcl documentation conventions

Designer Tcl Command Reference

ioadvisor_apply_suggestion

Tcl command; applies the suggestions for the selected attribute to the selected I/O(s).

```
ioadvisor_apply_suggestion -attribute {value} -io {value}
```

Arguments

-attribute{*value*}

This specifies the attribute for which the values will be applied. The following table shows the acceptable values for this argument:

Value	Description
outdrive	Applies suggested outdrive values
slew	Applies suggested slew values

-io {*value*}

This selects the I/Os for which the suggestion will be applied. To select multiple I/Os, use `-io {value}` for each I/O.

Supported Families

SmartFusion, IGLOO, ProASIC3 and Fusion

Exceptions

None

Example

The following code applies the suggested outdrive values for two I/Os.

```
ioadvisor_apply_suggestion -attribute{outdrive} -io{nPWM_out_pad} -io{PWM_out_pad}
```

See Also

ioadvisor_commit

ioadvisor_restore

ioadvisor_restore_initial_value

ioadvisor_set_outdrive

ioadvisor_set_outputload

ioadvisor_set_slew

Designer Tcl Command Reference

ioadvisor_commit

Tcl command; saves all changes in the I/O Advisor.

```
ioadvisor_commit
```

Arguments

None

Supported Families

SmartFusion, IGLOO, ProASIC3 and Fusion

Exceptions

None

Example

The following code saves all changes in the I/O Advisor:

```
ioadvisor_commit
```

See Also

ioadvisor_apply_suggestion

ioadvisor_restore

ioadvisor_restore_initial_value

ioadvisor_set_outdrive

ioadvisor_set_outputload

ioadvisor_set_slew

Designer Tcl Command Reference

ioadvisor_restore

Tcl command; restores the I/O Advisor to the initial state. All changes not committed will be lost.

```
ioadvisor_restore
```

Arguments

None

Supported Families

SmartFusion, IGLOO, ProASIC3 and Fusion

Exceptions

None

Example

The following code restores the I/O Advisor to the initial state:

```
ioadvisor_restore
```

See Also

`ioadvisor_apply_suggestion`

`ioadvisor_commit`

`ioadvisor_restore_initial_value`

`ioadvisor_set_outdrive`

`ioadvisor_set_outputload`

`ioadvisor_set_slew`

Designer Tcl Command Reference

ioadvisor_restore_initial_value

Tcl command; sets the current value for the selected attribute and I/Os to the initial value.

```
ioadvisor_restore_initial_value -attribute {value} -io {value}
```

Arguments

`-attribute{`*`value`*`}`

This specifies the attribute for which the values will be restored. The following table shows the acceptable values for this argument:

Value	Description
outdrive	Restores initial outdrive values
output_load	Restores initial output load values
slew	Restores initial slew values

`-io {`*`value`*`}`

This selects the I/Os for which the initial values will be restored. To select multiple I/Os, use `-io {`*`value`*`}` for each I/O.

Supported Families

SmartFusion, IGLOO, ProASIC3 and Fusion

Exceptions

None

Example

The following code restores the initial outdrive values for two I/Os.

```
ioadvisor_restore_initial_value -attribute{outdrive} -io{nPWM_out_pad} -io{PWM_out_pad}
```

See Also

`ioadvisor_apply_suggestion`

`ioadvisor_commit`

`ioadvisor_restore`

`ioadvisor_set_outdrive`

`ioadvisor_set_outputload`

`ioadvisor_set_slew`

Designer Tcl Command Reference

ioadvisor_set_outdrive

Tcl command; sets the outdrive for the selected I/Os.

```
ioadvisor_set_outdrive -io {value} -outdrive {value}
```

Arguments

`-io {`*`value`*`}`

This selects the I/Os for which the outdrive will be set. To select multiple I/Os, use `-io {`*`value`*`}` for each I/O.

`-outdrive {`*`value`*`}`

This specifies the outdrive for the selected I/Os.The outdrive must be a positive integer value within the list of possible outdrives of the I/Os.

Supported Families

SmartFusion, IGLOO, ProASIC3 and Fusion

Exceptions

None

Example

The following code sets the outdrive for two I/Os.

```
ioadvisor_set_outdrive -io{nPWM_out_pad} -io{PWM_out_pad} -outdrive{5}
```

See Also

`ioadvisor_apply_suggestion`

`ioadvisor_commit`

`ioadvisor_restore`

`ioadvisor_restore_initial_value`

`ioadvisor_set_outputload`

`ioadvisor_set_slew`

Designer Tcl Command Reference

ioadvisor_set_outputload

Tcl command; sets the output load for the selected I/Os.

```
ioadvisor_set_outputload -io {value} -outload {value}
```

Arguments

`-io {value}`

This selects the I/Os for which the output load will be set. To select multiple I/Os, use `-io {value}` for each I/O.

`-outload {value}`

This specifies the output load for the selected I/Os.The output load must be a positive integer value.

Supported Families

SmartFusion, IGLOO, ProASIC3 and Fusion

Exceptions

None

Example

The following code sets the output load for two I/Os.

```
ioadvisor_set_outputload -io{nPWM_out_pad} -io{PWM_out_pad} -outload{5}
```

See Also

ioadvisor_apply_suggestion

ioadvisor_commit

ioadvisor_restore

ioadvisor_restore_initial_value

ioadvisor_set_outdrive

ioadvisor_set_slew

Designer Tcl Command Reference

ioadvisor_set_slew

Tcl command; sets the slew for the selected I/Os.

```
ioadvisor_set_slew -io {value} -slew {value}
```

Arguments

`-io {value}`

This selects the I/Os for which the slew will be set. To select multiple I/Os, use `-io {value}` for each I/O.

`-set_slew {value}`

This specifies the slew for the selected I/Os.The following table shows the acceptable values for this argument:

Value	Description
high	The slew is set to high.

Value	Description
low	The slew is set to low. This option is not available for all I/Os.

Supported Families

SmartFusion, IGLOO, ProASIC3 and Fusion

Exceptions

None

Example

The following code sets the slew for two I/Os.

```
ioadvisor_set_slew -io{nPWM_out_pad} -io{PWM_out_pad} -slew{high}
```

See Also

ioadvisor_apply_suggestion

ioadvisor_commit

ioadvisor_restore

ioadvisor_restore_initial_value

ioadvisor_set_outdrive

ioadvisor_set_outputload

Designer Tcl Command Reference

is_design_loaded

Tcl command; returns a Boolean value (0 for false, 1 for true) indicating if a design is loaded in the Designer software. True is returned if a design is currently loaded.

```
is_design_loaded
```

Arguments

None

Supported Family

SmartFusion, IGLOO, ProASIC3 and Fusion

Description

Some Tcl commands are valid only if a design is currently loaded in Designer. Use the ‘is_design_loaded’ command to prevent runtime errors by checking for this before invoking the commands.

Exceptions

The command will return an error if arguments are passed.

Example

The following code will determine if a design has been loaded.

```
set bDesignLoaded [ is_design_loaded ]
if { $bDesignLoaded == 0 } {
  puts “No design is loaded.”
}
```

See Also

get_design_filename

get_design_info

is_design_modified

is_design_state_complete

Designer Tcl Command Reference

is_design_modified

Tcl command; returns a Boolean value (0 for false, 1 for true) indicating if a design has been modified in the Designer software. True is returned if a design has been modified.

```
is_design_modified
```

Arguments

None

Supported Family

SmartFusion, IGLOO, ProASIC3 and Fusion

Description

Some Tcl commands are valid only if a design has been modified in Designer. Use the is_design_modified command to prevent runtime errors by checking for this before invoking the commands.

Exceptions

Returns an error if arguments are passed.

Example

The following code will determine if a design has been modified.

```
set bDesignModified [ is_design_modified ]
if { $bDesignModified == 0 } {
  puts “Design has not been modified.”
}
```

See Also

get_design_filename

get_design_info

is_design_loaded

is_design_state_complete

Designer Tcl Command Reference

is_design_state_complete

Tcl command; returns a Boolean value (0 for false, 1 for true) indicating if a specific design state is valid. True is returned if the specified design state is valid.

```
is_design_state_complete value
```

Arguments

value

Must be one of the valid string values summarized in the table below:

Value	Description
SETUP_DESIGN	The design is loaded and the family has been specified for the design
DEVICE_SELECTION	The design has completed device selection (die and package). This corresponds to having successfully called the set_device command to set the die and package
NETLIST_IMPORT	The design has imported a netlist
COMPILE	The design has completed the compile command
LAYOUT	The design has completed the layout command
BACKANNOTATE	The design has exported a post-layout timing file (e.g.SDF)
PROGRAMMING_FILES	The design has exported a programming file (e.g. AFM)

Supported Family

SmartFusion, IGLOO, ProASIC3 and Fusion

Description

Certain commands can only be used after Compile or Layout has been completed. The is_design_state_complete command allows a script to check the design state before calling one of these state-limited commands.

Exceptions

The command will return an error if a design is not loaded.

The command will return an error if more than one argument is passed.

The command will return an error if the argument is not one of the valid values.

Example

The following code runs layout, but checks that the design state for layout is complete before calling backannotate.

```
layout –timing_driven
set bLayoutDone [ is_design_state_complete LAYOUT ]
if { $bLayoutDone  != 0 } {
  backannotate -name {mydesign_ba}  -format "SDF" -language "verilog"
  }
}
```

See Also

compile

get_design_filename

get_design_info
is_design_loaded
is_design_modified
layout
set_design
set_device
Designer Tcl Command Reference

is_source_file_current

Tcl command; audits the source file and determines whether or not the file is out of date / imported into the workspace. Returns '0' if file_name is out of date or has not been imported into the workspace, and returns '1' if file_name is current.

This command ignores the Audit settings in your ADB file.

```
is_source_file_current(filename)
```

Arguments

filename is the path to the source file

Supported Families

SmartFusion, IGLOO, ProASIC3 and Fusion

Exceptions

None

Example

The following code determines whether or not the file has been imported into the workspace.

```
is_source_file_current (./hdl/adder.vhd)
```

See Also

are_all_source_files_curent
get_out_of_date_files
Designer Tcl Command Reference

layout - SmartFusion, IGLOO, ProASIC3 and Fusion

Tcl command is identical to the layout command in the Designer GUI. Refer to the Advanced Layout Options below for more information.

```
layout
[-timing_driven | -standard]
[–power_driven value]
[-run_placer value]
[–place_incremental value]
[–run_router value]
[–route_incremental value]
```

Arguments

```
-timing_driven|-standard
```

Sets layout mode to be timing driven or standard (non-timing driven). The default is -timing_driven or the mode used in the previous layout command.

`-power_driven` *`value`*

The following table shows the acceptable values for this argument:

Value	Description
off	Does not run power-driven layout. This is the default.
on	Enables power-driven layout

`-place_incremental` *`value`*

The following table shows the acceptable values for this argument:

Value	Description
off	Discards previous placement. This is the default.
on	Sets the previous placement as the initial starting point
fix	Sets the previously placed macros' locations as "fixed" and continues to place the remaining ones

`-route_incremental` *`value`*

The following table shows the acceptable values for this argument:

Value	Description
off	Skips incremental mode, discards previous information. This is the default.
on	Invokes incremental routing and sets the previous routing information as the initial starting point

`-run_placer` *`value`*

The following table shows the acceptable values for this argument:

Value	Description
on	Invokes placement. This is the default.
off	Skips placement

`-run_router` *`value`*

The following table shows the acceptable values for this argument:

Value	Description
on	Invokes routing if placement is successful. This is the default.
off	Skips routing

layout - Advanced Options for SmartFusion, IGLOO, ProASIC3 and Fusion

This is equivalent to executing commands within the Advanced Layout Options dialog box.

```
[-placer_high_effort value]
[–seq_opt  value]
[–mindel_repair value]
[–placer_seed value]
[–show_placer_seed]
```

Arguments

`-placer_high_effort value`

The following table shows the acceptable values for this argument:

Value	Description
off	Disables physical synthesis of combinational logic. This is the default.
on	Enables physical synthesis of combinational logic

`-seq_opt  value`

The following table shows the acceptable values for this argument:

Value	Description
off	Disables physical synthesis of sequential logic. This is the default.
on	Enables physical synthesis of sequential logic in high-effort mode

`-mindel_repair value`

The following table shows the acceptable values for this argument:

Value	Description
off	Does not run minimum delay violations repair. This is the default.
on	Enables repair of minimum delay violations during route

`-placer_seed value`

An integer value that you can set to change the initial random seed number for the placement.

`-show_placer_seed value`

Causes Layout to display the initial random seed number used for the placement.

Exceptions

- None

Example

```
layout
layout –place_incremental FIX –route_incremental ON
layout –placer_high_effort ON
layout –run_placer OFF –route_incremental ON –mindel_repair ON
layout –timing_driven –power_driven ON
layout –placer_seed 120
```

See Also

Place and Route (Layout)

SmartFusion, IGLOO, ProASIC3 and Fusion Advanced Place and Route (Layout) Options

Designer Tcl Command Reference

list_clocks

Tcl command; returns details about all of the clock constraints in the current timing constraint scenario.

```
list_clocks
```

Arguments

None

Supported Families

SmartFusion, IGLOO, ProASIC3 and Fusion

Exceptions

None

Examples

```
puts [list_clocks]
```

See Also

`create_clock`

`remove_clock`

Tcl documentation conventions

Designer Tcl Command Reference

list_clock_latencies

Tcl command; returns details about all of the clock latencies in the current timing constraint scenario.

```
list_clock_latencies
```

Arguments

None

Supported Families

SmartFusion, IGLOO, ProASIC3 and Fusion

Exceptions

None

Examples

```
puts [list_clock_latencies]
```

See Also

`set_clock_latency`

`remove_clock_latency`

Tcl documentation conventions

Designer Tcl Command Reference

list_clock_uncertainties

Tcl command; returns details about all of the clock uncertainties in the current timing constraint scenario.

```
list_clock_uncertainties
```

Arguments

None

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Examples

```
list_clock_uncertainties
```

See Also

`set_clock_uncertainty`

`remove_clock_uncertainty`

Designer Tcl Command Reference

list_disable_timings

Tcl command; returns the list of disable timing constraints for the current scenario.

```
list_disable_timings
```

Arguments

- None

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Example

```
list_disable_timings
```

See Also

Designer Tcl Command Reference

list_false_paths

Tcl command; returns details about all of the false paths in the current timing constraint scenario.

```
list_false_paths
```

Arguments

None

Supported Families

SmartFusion, IGLOO, ProASIC3 and Fusion

Exceptions

None

Examples

```
puts [list_false_paths]
```

See Also

`set_false_path`

`remove_false_path`

Tcl documentation conventions

Designer Tcl Command Reference

list_generated_clocks

Tcl command; returns details about all of the generated clock constraints in the current timing constraint scenario.

```
list_generated_clocks
```

Arguments

None

Supported Families

SmartFusion, IGLOO, ProASIC3 and Fusion

Exceptions

None

Examples

```
puts [list_generated_clocks]
```

See Also

`create_generated_clock`

`remove_generated_clock`

Tcl documentation conventions

Designer Tcl Command Reference

list_input_delays

Tcl command; returns details about all of the input delay constraints in the current timing constraint scenario.

```
list_input_delays
```

Arguments

None

Supported Families

SmartFusion, IGLOO, ProASIC3 and Fusion

Exceptions

None

Examples

```
puts [list_input_delays]
```

See Also

set_input_delay

remove_input_delay

Tcl documentation conventions

Designer Tcl Command Reference

list_max_delays

Tcl command; returns details about all of the maximum delay constraints in the current timing constraint scenario.

```
list_max_delays
```

Arguments

None

Supported Families

SmartFusion, IGLOO, ProASIC3 and Fusion

Exceptions

None

Examples

```
puts [list_max_delays]
```

See Also

set_max_delay

remove_max_delay

Tcl documentation conventions

Designer Tcl Command Reference

list_min_delays

Tcl command; returns details about all of the minimum delay constraints in the current timing constraint scenario.

```
list_min_delays
```

Arguments

None

Supported Families

SmartFusion, IGLOO, ProASIC3 and Fusion

Exceptions

None

Examples

```
puts [list_min_delays]
```

See Also

`set_min_delay`

`remove_min_delay`

Tcl documentation conventions

Designer Tcl Command Reference

list_multicycle_paths

Tcl command; returns details about all of the multicycle paths in the current timing constraint scenario.

```
list_multicycle_paths
```

Arguments

None

Supported Families

SmartFusion, IGLOO, ProASIC3 and Fusion

Exceptions

None

Examples

```
puts [list_multicycle_paths]
```

See Also

`set_multicycle_path`

`remove_multicycle_path`

Tcl documentation conventions

Designer Tcl Command Reference

list_objects

Tcl command; returns a list of object matching the parameter. Objects can be nets, pins, ports, clocks or instances.

```
list_objects <object>
```

Arguments

Any timing constraint parameter.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Example

The following example lists all the inputs in your design:

```
list_objects [all_inputs]
```

You can also use wildcards to filter your list, as in the following command:

```
list_objects [get_ports a*]
```

See Also

Tcl documentation conventions

Designer Tcl Command Reference

list_output_delays

Tcl command; returns details about all of the output delay constraints in the current timing constraint scenario.

```
list_output_delays
```

Arguments

None

Supported Families

SmartFusion, IGLOO, ProASIC3 and Fusion

Exceptions

None

Examples

```
puts [list_output_delays]
```

See Also

`set_output_delay`

`remove_output_delay`

Tcl documentation conventions

Designer Tcl Command Reference

list_scenarios

Tcl command; returns a list of names of all of the available timing scenarios.

```
list_scenarios
```

Arguments

None

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Examples

```
list_scenarios
```

See Also

get_current_scenario

Tcl documentation conventions

Designer Tcl Command Reference

LOGFILE

The LOGFILE command is not a Tcl script. It runs in conjunction with your Tcl script and enables you to specify a filename to which Designer will record/save a copy of the log messages generated in batch-mode (SCRIPT:...).

It is useful if you want to view the log after you run scripts in batch-mode on Windows.

For example, to run the script 'myscript.tcl' for Designer and save the log messages to a LOGFILE named 'mylog.txt', use the command:

```
designer.exe SCRIPT:myscript.tcl LOGFILE:mylog.txt
```

See Also

Introduction to Tcl scripting

new_design

Tcl command; creates a new design. You need all three arguments for this command. This command will set up the Designer software for importing design source files

```
new_design -name design_name -family family_name –path  pathname–block value
```

Arguments

```
-name design_name
```

The name of the design. This is used as the base name for most of the files generated from Designer.

```
-family family_name
```

The Microsemi SoC device family for which the design is being targeted.

```
-path path_name
```

The physical path of the directory in which the design files will be created.

```
block value
```

Enables or disables Block mode.The following table shows the acceptable values for this option:

Value	Description
on	Enables Block mode
off	Disables Block mode

Supported Families

SmartFusion, IGLOO, ProASIC3 and Fusion

Exceptions

None

Example

Example 1: Creates a new ACT3 design with the name “test” in the current folder.

```
new_design -name "test" -family "ACT3" -path {.}
```

Example 2: These set of commands create a new design through variable substitution.

```
set desName “test
set famName “ACT3”
set path {d:/examples/test}
new_design -name $desName -family $famName -path $path
```

Example 3: Design creation and catch failures

```
if  { [catch { new_design -name $desName -family $famName -path $path }] {
              Puts “Failed to create a new design”
              # Handle Failure
} else {
              puts “New design creation successful”
              # Proceed to Import source files
}
```

See Also

`close_design`

`open_design`

`save_design`

`set_design`

Designer Tcl Command Reference

open_design

Tcl command; opens an existing design into the Designer software.

```
open_design file_name
```

Note: Note: All previously open designs must be closed before opening a new design.

Arguments

file_name

The complete .adb file path. If the complete path is not provided, then the directory is assumed to be the current working directory.

Supported Families

SmartFusion, IGLOO, ProASIC3 and Fusion

Exceptions

None

Example

Example 1: Opens an existing design from the file “test.adb” in the current folder.

```
open_design {test.adb}
```

Example 2: Design creation and catch failures.

```
set designFile {d:/test/my_design.adb}
if  { [catch { open_design $designFile }] {
              Puts “Failed to open design”
              # Handle Failure
} else {
              puts “Design opened successfully”
              # Proceed to further processing
}
```

See Also

`close_design`

`new_design`

`save_design`

Designer Tcl Command Reference

pin_assign

Tcl command; use to either assign the named pin to the specified port or assign attributes to the specified port. This command has two syntax formats. The one you use depends on what you are trying to do. The first syntax format assigns the named pin to the specified port . The second one assigns attributes to the specified port.

```
pin_assign [-nofix] -port portname -pin pin_number
pin_assign -port portname [-iostd value][-iothresh value][-outload value][-slew value][-
res_pull value]
```

Arguments

`-nofix`

Unlocks the pin assignment (by default, assignments are locked).

`-port` *`portname`*

Specifies the name of the port to which the pin is assigned.

`-pin` *`pin_number`*

Specifies the alphanumeric number of the pin to assign.

`-iostd` *`value`*

Sets the I/O standard for this pin. Choosing a standard allows the software to set other attributes such as the slew rate and output loading. If the voltage standard used with the I/O is not compatible with other I/Os in the I/O bank, then assigning an I/O standard to a port will invalidate its location and automatically unassign the I/O. The following table shows the acceptable values for the supported devices:

I/O Standards table

Use the I/O Standards table to see which I/O standards can be applied to each family:

I/O Standard	IGLOO	Fusion	ProASIC3
CMOS			
CUSTOM			
GTLP25	IGLOOe only	X	ProASIC3E and ProASIC3L only
GTLP33	IGLOOe only	X	ProASIC3E and ProASIC3L only
GTL33	IGLOOe only	X	ProASIC3E and ProASIC3L only
GTL25	IGLOOe only	X	ProASIC3E and ProASIC3L only
HSTL1	IGLOOe only	X	ProASIC3E and ProASIC3L only
HSTLII	IGLOOe only	X	ProASIC3E and ProASIC3L only
LVCMOS33	X	X	X
LVCMOS25	IGLOOe only	X	X
LVCMOS25_50	X	X	X
LVCMOS18	X	X	X
LVCMOS15	X	X	X
LVCMOS12	X		ProASIC3L only
LVTTL	X	X	X
TTL	X	X	X
PCI	X	X	X
PCIX	X	X	X
SSTL2I	IGLOOe only	X	ProASIC3E and ProASIC3L only
SSTL2II	IGLOOe only	X	ProASIC3E and ProASIC3L only
SSTL3I	IGLOOe only	X	ProASIC3E and ProASIC3L only
SSTL3II	IGLOOe only	X	ProASIC3E and ProASIC3L only

See Also

I/O standard

Note: Note: The LVDS and LVPECL I/O standards cannot be set through a script.

`-iothresh value`

Sets the compatible threshold level for inputs and outputs. The default I/O threshold is based upon the I/O standard. You can set the I/O Threshold independently of the I/O specification in the PinEditor tool by selecting **CUSTOM** in the I/O Standard cell. The following table shows the acceptable values for the supported devices:

Value	Description
CMOS	RTSX-S devices only. An advanced integrated circuit (IC) manufacturing process technology for logic and memory, characterized by high integration, low cost, low power, and high performance. CMOS logic uses a combination of p-type and n-type metal-oxide-semiconductor field effect transistors (MOSFETs) to implement logic gates and other digital circuits found in computers, telecommunications, and signal processing equipment.
LVTTL	(Low-Voltage TTL) A general purpose standard (EIA/JESDSA) for 3.3V applications. It uses an LVTTL input buffer and a push-pull output buffer.
PCI	A computer bus for attaching peripheral devices to a computer motherboard in a local bus. This standard supports both 33 MHz and 66 MHz PCI bus applications. It uses an LVTTL input buffer and a push-pull output buffer. With the aid of an external resistor, this I/O standard can be 5V-compliant for most families, excluding SmartFusion, IGLOO, ProASIC3 and Fusion families.

Note: Note: The -iothresh attribute is also referred to as "Loading" in some families.

`-slew` *`value`*

Sets the output slew rate. Slew control affects only the falling edges. Rising edges are not affected. This attribute is only available for LVTTL, PCI, and PCI outputs. For LVTTL, it can either be high or low. For PCI and PCIX, it can only be set to high. The following table shows the acceptable values for the supported devices (IGLOO, ProASIC3, SmartFusion, Fusion):

Value	Description
high	Sets the I/O slew to high
low	Sets the I/O slew to low

`-res_pull` *`value`*

Allows you to include a weak resistor for either pull-up or pull-down of the input buffer. The following table shows the acceptable values for the supported devices (IGLOO, ProASIC3, SmartFusion, Fusion):

Value	Description
up	Includes a weak resistor for pull-up of the input buffer
down	Includes a weak resistor for pull-down of the input buffer
none	Does not include a weak resistor

`-out_load` *`value`*

Indicates the output-capacitance value based on the I/O standard selected. This option is not available in software. This attribute determines what Timer will use as the loading on the output pin and applies only to outputs. You can enter a capacitive load as an integral number of picofarads (*pF*). The default is 35*pF*. This attribute is available only for the following devices: SmartFusion, Fusion, ProASIC3.

Supported Families

SmartFusion, IGLOO, ProASIC3 and Fusion

Exceptions

None

Examples

You must use `pin_commit` after the pin_assign command to save the changes to your design:

```
pin_assign -port usw0 -pin A2
pin_commit

pin_assign -port usw0 -iostd LVTTL -slew low -res_pull down
pin_commit
```

Note: Note: To use a name with special characters such as square brackets [ ], you must put the entire name between curly braces { } or put a slash character \ immediately before each square bracket as shown in the following examples.

Note: The following example shows a port name enclosed with curly braces:

Note: The next example shows each square bracket preceded by a slash:

```
pin_assign -port LFSR_OUT\[15\] -iostd lvttl -slew High
```

See Also

`pin_commit`

`pin_fix`

`pin_unassign`

Tcl documentation conventions

Designer Tcl Command Reference

pin_commit

Tcl command; saves the pin assignments to the design (.adb) file.

```
pin_commit
```

Arguments

None

Supported Families

SmartFusion, IGLOO, ProASIC3 and Fusion

Exceptions

None

Examples

To save pin assignments in your design, you must add the pin_commit command to the end of the script:

```
pin_commit
```

See Also

`pin_fix`

`pin_unfix`

`pin_assign`

`pin_unassign`

Tcl documentation conventions

Designer Tcl Command Reference

pin_fix

Tcl command; locks the pin assignment for the specified port, so the pins cannot be moved during place-and-route.

```
pin_fix -port portname
```

Arguments

`-port portname`

Specifies the name of the port to which the pin must be locked at its assigned location.

Note: Note: You can assign only one pin to a port

Supported Families

SmartFusion, IGLOO, ProASIC3 and Fusion

Description

Fixed pins are locked pins. You cannot move locked pins during place-and-route.

Exceptions

None

Examples

You must use `pin_commit` after the pin_fix command to save the changes to your design:

```
pin_fix –port clk
pin_commit
```

See Also

`pin_commit`

`pin_unfix`

`pin_assign`

`pin_unassign`

Tcl documentation conventions

Designer Tcl Command Reference

pin_fix_all

Tcl command; locks all the assigned pins on the device so they cannot be moved during place-and-route.

```
pin_fix_all
```

Arguments

- None

Supported Families

SmartFusion, IGLOO, ProASIC3 and Fusion

Description

Fixed pins are locked pins. This command locks all the pins in your design. You cannot move locked pins during place-and-route.

Exceptions

None

Example

You must use `pin_commit` after the pin_fix_all command to save the changes to your design:

```
pin_fix_all
pin_commit
```

See Also

`pin_commit`

`pin_fix`

`pin_unfix`

`pin_assign`

`pin_unassign`

Tcl documentation conventions

Designer Tcl Command Reference

pin_unassign

Tcl command; unassigns the pin from the specified port. The unassigned pin location is then available for other ports. (Only one pin can be assigned to a port.)

```
pin_unassign -port portname
```

Arguments

`-port` *`portname`*

Specifies the name of the port for which the pin must be unassigned.

Supported Families

SmartFusion, IGLOO, ProASIC3 and Fusion

Exceptions

None

Examples

You must use `pin_commit` after the pin_assign command to save the changes to your design:

```
pin_unassign –port “clk”
pin_commit
```

See Also

`pin_commit`

`pin_fix`

`pin_fix_all`

`pin_unfix`

`pin_assign`

`pin_unassign`

[Tcl documentation conventions]

[Designer Tcl Command Reference]

pin_unassign_all

Tcl command; unassigns all the pins from all the ports so that all pin locations are available for assignment.

```
pin_unassign_all
```

Arguments

None

Supported Families

SmartFusion, IGLOO, ProASIC3 and Fusion

Exceptions

None

Examples

You must use `pin_commit` after the pin_assign_all command to save the changes to your design:

```
pin_unassign_all
pin_commit
```

See Also

`pin_commit`

`pin_fix`

`pin_unfix`

`pin_assign`

`pin_unassign`

Tcl documentation conventions

Designer Tcl Command Reference

pin_unfix

Tcl command; unlocks the pins assigned to the specified port, so the pins can be moved during place-and-route.

```
pin_unfix -port portname
```

Arguments

`-port` *`portname`*

Specifies the name of the port containing pins to unlock.

Supported Families

SmartFusion, IGLOO, ProASIC3 and Fusion

Exceptions

None

Examples

You must use `pin_commit command` after the pin_unfix command to save the changes to your design:

```
pin_unfix –port rst
pin_commit
```

See Also

pin_commit

pin_fix

pin_assign

pin_unassign

Tcl documentation conventions

Designer Tcl Command Reference

remove_clock

Tcl command; removes the specified clock constraint from the current timing scenario.

```
remove_clock {-name clock_name| -id constraint_ID
```

Arguments

`-name clock_name`

Specifies the name of the clock constraint to remove from the current scenario. You must specify either a clock name or an ID.

`-id constraint_ID`

Specifies the ID of the clock constraint to remove from the current scenario. You must specify either an ID or a clock name that exists in the current scenario.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Description

Removes the specified clock constraint from the current scenario. If the specified name does not match a clock constraint in the current scenario, or if the specified ID does not refer to a clock constraint, this command fails.

Do not specify both the name and the ID.

Exceptions

You cannot use wildcards when specifying a clock name.

Examples

The following example removes the clock constraint named "my_user_clock":

```
remove_clock -name my_user_clock
```

The following example removes the clock constraint using its ID:

```
set clockId [create_clock –name my_user_clock –period 2]
remove_clock –id $clockId
```

See Also

create_clock

Tcl Command Documentation Conventions

Designer Tcl Command Reference

remove_clock_latency

Tcl command; removes a clock source latency from the specified clock and from all edges of the clock.

```
remove_clock_latency {-source clock_name_or_source |-id constraint_ID}
```

Arguments

`-source clock_name_or_source`

Specifies either the clock name or source name of the clock constraint from which to remove the clock source latency. You must specify either a clock or source name or its constraint ID.

`-id constraint_ID`

Specifies the ID of the clock constraint to remove from the current scenario. You must specify either a clock or source name or its constraint ID.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Description

Removes a clock source latency from the specified clock in the current scenario. If the specified source does not match a clock with a latency constraint in the current scenario, or if the specified ID does not refer to a clock with a latency constraint, this command fails.

Do not specify both the source and the ID.

Exceptions

You cannot use wildcards when specifying a clock name.

Examples

The following example removes the clock source latency from the specified clock.

```
remove_clock_latency -source my_clock
```

See Also

`set_clock_latency`

Tcl Command Documentation Conventions

Designer Tcl Command Reference

remove_clock_uncertainty

Tcl command; removes a clock-to-clock uncertainty from the current timing scenario by specifying either its exact arguments or its ID.

```
remove_clock_uncertainty -from | -rise_from | -fall_from from_clock_list -to | -rise_to| -
fall_to to_clock_list -setup {value} -hold {value}
remove_clock_uncertainty -id constraint_ID
```

Arguments

`-from`

Specifies that the clock-to-clock uncertainty applies to both rising and falling edges of the source clock list. Only one of the `-from`, `-rise_from`, or `-fall_from` arguments can be specified for the constraint to be valid.

`-rise_from`

Specifies that the clock-to-clock uncertainty applies only to rising edges of the source clock list. Only one of the `-from`, `-rise_from`, or `-fall_from` arguments can be specified for the constraint to be valid.

`-fall_from`

Specifies that the clock-to-clock uncertainty applies only to falling edges of the source clock list. Only one of the `-from`, `-rise_from`, or `-fall_from` arguments can be specified for the constraint to be valid.

`from_clock_list`

Specifies the list of clock names as the uncertainty source.

`-to`

Specifies that the clock-to-clock uncertainty applies to both rising and falling edges of the destination clock list. Only one of the `-to`, `-rise_to` , or `-fall_to` arguments can be specified for the constraint to be valid.

`-rise_to`

Specifies that the clock-to-clock uncertainty applies only to rising edges of the destination clock list. Only one of the `-to`, `-rise_to` , or `-fall_to` arguments can be specified for the constraint to be valid.

`-fall_to`

Specifies that the clock-to-clock uncertainty applies only to falling edges of the destination clock list. Only one of the `-to`, `-rise_to` , or `-fall_to` arguments can be specified for the constraint to be valid.

`to_clock_list`

Specifies the list of clock names as the uncertainty destination.

`-setup`

Specifies that the uncertainty applies only to setup checks. If none or both `-setup` and `-hold` are present, the uncertainty applies to both setup and hold checks.

`-hold`

Specifies that the uncertainty applies only to hold checks. If none or both `-setup` and `-hold` are present, the uncertainty applies to both setup and hold checks.

`-id` *`constraint_ID`*

Specifies the ID of the clock constraint to remove from the current scenario. You must specify either the exact parameters to set the constraint or its constraint ID.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Description

Removes a clock-to-clock uncertainty from the specified clock in the current scenario. If the specified arguments do not match clocks with an uncertainty constraint in the current scenario, or if the specified ID does not refer to a clock-to-clock uncertainty constraint, this command fails.

Do not specify both the exact arguments and the ID.

Exceptions

None

Examples

```
remove_clock_uncertainty -from Clk1 -to Clk2
remove_clock_uncertainty -from Clk1 -fall_to { Clk2 Clk3 } -setup
remove_clock_uncertainty 4.3 -fall_from { Clk1 Clk2 } -rise_to *
remove_clock_uncertainty 0.1 -rise_from [ get_clocks { Clk1 Clk2 } ] -fall_to { Clk3
Clk4 } -setup
remove_clock_uncertainty 5 -rise_from Clk1 -to [ get_clocks {*} ]
remove_clock_uncertainty -id $clockId
```

See Also

remove_clock

`remove_generated_clock`

`set_clock_uncertainty`

Designer Tcl Command Reference

remove_disable_timing

Tcl command; removes a disable timing constraint by specifying its arguments, or its ID. If the arguments do not match a disable timing constraint, or if the ID does not refer to a disable timing constraint, the command fails.

```
remove_disable_timing -from value -to value name -id name
```

Arguments

`-from from_port`

Specifies the starting port. The –from and –to arguments must either both be present or both omitted for the constraint to be valid.

`-to to_port`

Specifies the ending port. The –from and –to arguments must either both be present or both omitted for the constraint to be valid.

`name`

Specifies the cell name where the disable timing constraint will be removed. It is an error to supply both a cell name and a constraint ID, as they are mutually exclusive. No wildcards are allowed when specifying a clock name, either alone or in an accessor command1.

`-id name`

Specifies the constraint name where the disable timing constraint will be removed. It is an error to supply both a cell name and a constraint ID, as they are mutually exclusive. No wildcards are allowed when specifying a clock name, either alone or in an accessor command1.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Example

```
remove_disable_timing -from port1 -to port2 -id new_constraint
```

Designer Tcl Command Reference

remove_false_path

Tcl command; removes a false path from the current timing scenario by specifying either its exact arguments or its ID.

```
remove_false_path [-from from_list] [-to to_list] [-through through_list] [-id constraint_ID]
remove_false_path -id constraint_ID
```

Arguments

`-from from_list`

Specifies a list of timing path starting points. A valid timing starting point is a clock, a primary input, an inout port, or a clock pin of a sequential cell.

`-through through_list`

Specifies a list of pins, ports, cells, or nets through which the disabled paths must pass.

-to *to_list*

Specifies a list of timing path ending points. A valid timing ending point is a clock, a primary output, an inout port, or a data pin of a sequential cell.

-id *constraint_ID*

Specifies the ID of the false path constraint to remove from the current scenario. You must specify either the exact false path to remove or the constraint ID that refers to the false path constraint to remove.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Description

Removes a false path from the specified clock in the current scenario. If the arguments do not match a false path constraint in the current scenario, or if the specified ID does not refer to a false path constraint, this command fails.

Do not specify both the false path arguments and the constraint ID.

Exceptions

- You cannot use wildcards when specifying a clock name, either alone or in an Accessor command such as get_pins or get_ports.

Examples

The following example specifies all false paths to remove:

```
remove_false_path -through U0/U1:Y
```

The following example removes the false path constraint using its id:

```
set fpId [set_false_path –from [get_clocks c*] –through {topx/reg/*} –to [get_ports
out15] ]
remove_false_path –id $fpId
```

See Also

set_false_path

Tcl Command Documentation Conventions

Designer Tcl Command Reference

remove_generated_clock

Tcl command; removes the specified generated clock constraint from the current scenario.

```
remove_generated_clock {-name clock_name | -id constraint_ID }
```

Arguments

-name *clock_name*

Specifies the name of the generated clock constraint to remove from the current scenario. You must specify either a clock name or an ID.

-id *constraint_ID*

Specifies the ID of the generated clock constraint to remove from the current scenario. You must specify either an ID or a clock name that exists in the current scenario.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Description

Removes the specified generated clock constraint from the current scenario. If the specified name does not match a generated clock constraint in the current scenario, or if the specified ID does not refer to a generated clock constraint, this command fails.

Do not specify both the name and the ID.

Exceptions

You cannot use wildcards when specifying a generated clock name.

Examples

The following example removes the generated clock constraint named "my_user_clock":

```
remove_generated_clock -name my_user_clock
```

See Also

create_generated_clock

Tcl Command Documentation Conventions

Designer Tcl Command Reference

remove_input_delay

Tcl command; removes an input delay a clock on a port by specifying both the clocks and port names or the ID of the input_delay constraint to remove.

```
remove_input_delay -clock clock_name port_pin_list
remove_input_delay -id constraint_ID
```

Arguments

`-clock clock_name`

Specifies the clock name to which the specified input delay value is assigned.

`port_pin_list`

Specifies the port names to which the specified input delay value is assigned.

`-id constraint_ID`

Specifies the ID of the clock with the input_delay value to remove from the current scenario. You must specify either both a clock name and list of port names or the input_delay constraint ID .

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Description

Removes an input delay from the specified clocks and port in the current scenario. If the clocks and port names do not match an input delay constraint in the current scenario, or if the specified ID does not refer to an input delay constraint, this command fails.

Do not specify both the clock and port names and the constraint ID.

Exceptions

You cannot use wildcards when specifying a clock name, either alone or in an accessor command.

Examples

The following example removes the input delay from CLK1 on port data1:

```
remove_input_delay -clock [get_clocks CLK1] [get_ports data1]
```

See Also

set_input_delay

Tcl Command Documentation Conventions

Designer Tcl Command Reference

remove_library

Tcl command; removes a VHDL library from your project.

```
remove_library
-library name
```

Arguments

`-library name`

Specifies the name of the library you wish to remove.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Example

Remove (delete) a library called 'my_lib'.

```
remove_library –library my_lib
```

See Also

Project Manager Tcl Command Reference

add_library

rename_library

remove_max_delay

Tcl command; removes a maximum delay constraint from the current timing scenario by specifying either its exact arguments or its ID.

```
remove_max_delay [-from from_list] [-to to_list] [-through through_list]
remove_max_delay -id constraint_ID
```

Arguments

`-from from_list`

Specifies a list of timing path starting points. A valid timing starting point is a clock, a primary input, an inout port, or a clock pin of a sequential cell.

`-through through_list`

Specifies a list of pins, ports, cells, or nets through which the disabled paths must pass.

`-to to_list`

Specifies a list of timing path ending points. A valid timing ending point is a clock, a primary output, an inout port, or a data pin of a sequential cell.

`-id` *`constraint_ID`*

Specifies the ID of the maximum delay constraint to remove from the current scenario. You must specify either the exact maximum delay arguments to remove or the constraint ID that refers to the maximum delay constraint to remove.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Description

Removes a maximum delay value from the specified clock in the current scenario. If the arguments do not match a maximum delay constraint in the current scenario, or if the specified ID does not refer to a maximum delay constraint, this command fails.

Do not specify both the maximum delay arguments and the constraint ID.

Exceptions

You cannot use wildcards when specifying a clock name, either alone or in an Accessor command.

Examples

The following example specifies a range of maximum delay constraints to remove:

```
remove_max_delay -through U0/U1:Y
```

See Also

`set_max_delay`

Tcl Command Documentation Conventions

Designer Tcl Command Reference

remove_min_delay

Tcl command; removes a minimum delay constraint in the current timing scenario by specifying either its exact arguments or its ID.

```
remove_min_delay [-from from_list] [-to to_list] [-through through_list]
remove_min_delay -id constraint_ID
```

Arguments

`-from` *`from_list`*

Specifies a list of timing path starting points. A valid timing starting point is a clock, a primary input, an inout port, or a clock pin of a sequential cell.

`-through` *`through_list`*

Specifies a list of pins, ports, cells, or nets through which the disabled paths must pass.

`-to`*`to_list`*

Specifies a list of timing path ending points. A valid timing ending point is a clock, a primary output, an inout port, or a data pin of a sequential cell.

`-id` *`constraint_ID`*

Specifies the ID of the minimum delay constraint to remove from the current scenario. You must specify either the exact minimum delay arguments to remove or the constraint ID that refers to the minimum delay constraint to remove.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Description

Removes a minimum delay value from the specified clock in the current scenario. If the arguments do not match a minimum delay constraint in the current scenario, or if the specified ID does not refer to a minimum delay constraint, this command fails.

Do not specify both the minimum delay arguments and the constraint ID.

Exceptions

You cannot use wildcards when specifying a clock name, either alone or in an accessor command.

Examples

The following example specifies a range of minimum delay constraints to remove:

```
remove_min_delay -through U0/U1:Y
```

See Also

set_min_delay

Tcl Command Documentation Conventions

Designer Tcl Command Reference

remove_multicycle_path

Tcl command; removes a multicycle path constraint in the current timing scenario by specifying either its exact arguments or its ID.

```
remove_multicycle_path [-from from_list] [-to to_list] [-through through_list]
remove multicycle_path -id constraint_ID
```

Arguments

`-from from_list`

Specifies a list of timing path starting points. A valid timing starting point is a clock, a primary input, an inout port, or a clock pin of a sequential cell.

`-through through_list`

Specifies a list of pins, ports, cells, or nets through which the disabled paths must pass.

`-toto_list`

Specifies a list of timing path ending points. A valid timing ending point is a clock, a primary output, an inout port, or a data pin of a sequential cell.

`-id constraint_ID`

Specifies the ID of the multicycle path constraint to remove from the current scenario. You must specify either the exact multicycle path arguments to remove or the constraint ID that refers to the multicycle path constraint to remove.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Description

Removes a multicycle path from the specified clock in the current scenario. If the arguments do not match a multicycle path constraint in the current scenario, or if the specified ID does not refer to a multicycle path constraint, this command fails.

Do not specify both the multicycle path arguments and the constraint ID.

Exceptions

You cannot use wildcards when specifying a clock name, either alone or in an accessor command.

Examples

The following example removes all paths between reg1 and reg2 to 3 cycles for setup check.

```
remove_multicycle_path -from [get_pins {reg1}] –to [get_pins {reg2}]
```

See Also

`set_multicycle_path`

Tcl Command Documentation Conventions

Designer Tcl Command Reference

remove_output_delay

Tcl command; removes an ouput delay by specifying both the clocks and port names or the ID of the output_delay constraint to remove.

```
remove_output_delay -clock clock_name port_pin_list
remove_output_delay -id constraint_ID
```

Arguments

`-clock` *`clock_name`*

Specifies the clock name to which the specified output delay value is assigned.

`port_pin_list`

Specifies the port names to which the specified output delay value is assigned.

`-id` *`constraint_ID`*

Specifies the ID of the clock with the output_delay value to remove from the current scenario. You must specify either both a clock name and list of port names or the output_delay constraint ID .

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Description

Removes an output delay from the specified clocks and port in the current scenario. If the clocks and port names do not match an output delay constraint in the current scenario, or if the specified ID does not refer to an output delay constraint, this command fails.

Do not specify both the clock and port names and the constraint ID.

Exceptions

You cannot use wildcards when specifying a clock name, either alone or in an accessor command.

Examples

The following example removes the output delay from CLK1 on port out1:

remove_output_delay -clock [get_clocks CLK1] [get_ports out1]

See Also

`set_output_delay`

Tcl Command Documentation Conventions

Designer Tcl Command Reference

rename_library

Tcl command; renames a VHDL library in your project.

```
rename_library
-library name
 -name name
```

Arguments

`-library name`

Identifies the current name of the library that you wish to rename.

`-name name`

Specifies the new name of the library.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Example

Rename a library from 'my_lib' to 'test_lib1'

```
rename_library –library my_lib -name test_lib1
```

See Also

Project Manager Tcl Command Reference

`add_library`

`remove_library`

rename_scenario

Tcl command; renames the specified timing scenario with the new name provided. You must provide a unique new name (that is, it cannot already be used by another timing scenario).

```
rename_scenario oldname -new newname
```

Arguments

`oldname`

Specifies the current name of the timing scenario.

`-new newname`

Specifies the new name to give to the timing scenario.

Supported Families

IGLOO, ProASIC3, SmartFusion2, SmartFusion, Fusion

Description

This command changes the name of the timing scenario in the list of scenarios.

Example

```
rename_scenario scenario_A -new scenario_B
```

See Also

`create_scenario`

`delete_scenario`

Tcl documentation conventions

Designer Tcl Command Reference

report

The report command provides you with frequently-used information in a convenient format.

You can generate several different types of reports using this command, including:

- report (Status)
- report (Timing) for IGLOO, ProASIC3, SmartFusion2, SmartFusion, Fusion families
- report (Timing violations) for IGLOO, ProASIC3, SmartFusion2, SmartFusion, Fusion families
- report (Pin)
- report (Flip-flop)
- report (I/O Bank)
- report (Global Usage)
- report (Power)

report (Bottleneck) using SmartTime

Tcl command; creates a bottleneck report.

```
report -type bottleneck
 [-cost_type {value} ]
 [-use_slack_threshold{value} ]
 [-slack_threshold {value} ]
 [-set_name {value} ]
 [-clock clock_id -set_type value ]
 [-source_clock clock_id -sink_clock clock_id]
 [-source {pin_list} ]
 [-sink {pin_list} ]
 [-max_instances {value} ]
 [-max_paths {value} ]
 [-max_parallel_paths {value} ]
 [-analysis_type {value} ]
 {filename} \
 [-format value]
```

Arguments

```
-cost_type value
```

Specifies the type of bottleneck cost. The default option is path_count.

Value	Description
path_count	Instances with the greatest number of path violations will have the highest bottleneck cost
path_cost	Instances with the largest combined timing violations will have the highest bottleneck cost

`-use_slack_threshold` *`value`*

Specifies whether to consider the slack threshold when computing the bottlenecks in the report.

Value	Description
yes	Includes slack threshold in the bottleneck report
no	Excludes slack threshold in the bottleneck report

`-slack_threshold` *`value`*

Specifies that paths whose slack is larger than this given threshold will be considered. Only instances that lie on these violating paths are reported. The default option is 0.

`-set_name` *`value`*

Displays the bottleneck information for the named set. You can either use this option or use both –clock and –type. This option allows pruning based on a given set. Only paths that lie within the named set will be considered towards bottleneck.

`-clock` *`value`*

This option allows pruning based on a given clock domain. Only instances that lie on these violating paths are reported.

`-set_type` *`value`*

This option can only be used in combination with the –clock option, and not by itself. The options allow to filter which type of paths should be considered towards the bottleneck.

Value	Description
reg_to_reg	Paths between registers in the design
async_to_reg	Paths from asynchronous pins to registers
reg_to_async	Paths from registers to asynchronous pins
external_recovery	The set of paths from inputs to asynchronous pins
external_removal	The set of paths from inputs to asynchronous pins
external_setup	Paths from input ports to registers
external_hold	Paths from input ports to registers
clock_to_out	Paths from registers to output ports

`-source_clock` *`clock_id`*

Reports only bottleneck instances that lie on violating timing paths of the inter-clock domain that starts at the source clock specified by this option. This option can only be used in combination with -sink_clock, and not by itself.

`-sink_clock` *`clock_id`*

Reports only bottleneck instances that lie on violating timing paths of the inter-clock domain that ends at the sink clock specified by this option. This option can only be used in combination with -source_clock, and not by itself.

`-source` *`value`*

Reports only instances that lie on violating paths that start at locations specified by this option.

`-sink` *`value`*

Reports only instances that lie on violating paths that end at locations specified by this option.

`-max_instances` *`value`*

Specifies the maximum number of instances to be reported. Defaults to 10.

`-max_paths` *`value`*

Specifies the maximum number of paths to be considered per path set type. Allowed values are 1 to 2000000. Defaults to 100.

`-max_parallel_paths` *`value`*

Specifies the maximum number of paths allowed per end point pair. Only instances that lie on these violating paths are reported. Defaults to 1 (No parallel paths).

`-analysis_type` *`value`*

Specifies the analysis types (max or min) under which the violations are reported. Defaults to max analysis.

Value	Description
max_delay	Sets the analysis type to maximum delay
min_delay	Sets the analysis type to minimum delay

`–format` *`value`*

Specifies the output format of the generated report.

Value	Description
text	Generates a text report; text is the default value
csv	Generates the report in a comma-separated value format that you can import into a spreadsheet

`filename`

Specifies the name and destination of the bottleneck report.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Examples

The following example generates a bottleneck report named bottleneck.txt.

```
report -type bottleneck -cost-type path_count -slack_threshold 0 -set_name set1 -
max_cells 10 -max_paths 10 -max_parallel_paths 10 -analysis_type max -format text
bottleneck.txt
```

See Also

Tcl documentation conventions

Designer Tcl Command Reference

report (Cycle Accurate Power Report)

Tcl command; creates a cycle accurate power report, which reports a power waveform with one power value per clock period or half-period instead of an average power for the whole simulation.

```
report -type power_peak_analyzer \
[-vcd_file {path}] \
[-style {value}] \
[-partial_parse {value}] \
[-start_time {value}] \
[-end_time {value}] \
[-auto_detect_top_level_name {value}] \
[-top_level_name {name}] \
[-glitch_filtering {value}] \
[-glitch_threshold {value}] \
[-auto_detect_sampling_period {value}] \
[-sampling_clock { }] \
[-sampling_rate_per_period {value}] \
[-sampling_offset {value}] \
[-sampling_period {value}] \
[-use_only_local_extrema {value}] \
[-use_power_threshold {value}] \
[-power_threshold {value}] \
[-opmode {value}] \
{filename}
```

Arguments

`-type power_peak_analyzer`

Specifies the type of report to generate is a cycle accurate power report.

`-vcd_file {path}`

Specifies the path to the *.vcd file that you want to import.

`-style {value}`

Specifies the format in which the report will be exported. The following table shows the acceptable values for this argument:

Value	Description
Text	The report will be exported as Text file
CSV	The report will be exported as CSV file

`-partial_parse {value}`

Specifies whether to partially parse the *.vcd file. The following table shows the acceptable values for this argument:

Value	Description
true	Partially parses the *.vcd file
false	Does not partially parse the *.vcd file

`-start_time {value}`

This option is available only if `-partially_parse` is set to *true*. Specifies the start time (in ns) to partially parse the *.vcd file.

`-end_time {value}`

This option is available only if `-partially_parse` is set to *true*. Specifies the end time (in ns) to partially parse the *.vcd file.

`-auto_detect_top_level_name {value}`

Specifies whether to automatically detect the top-level name. The following table shows the acceptable values for this argument:

Value	Description
true	Automatically detects the top-level name
false	Does not automatically detect the top-level name

`-top_level_name {name}`

Specifies the top-level name.

`-glitch_filtering {value}`

Specifies whether to use glitch filtering. The following table shows the acceptable values for this argument:

Value	Description
true	Glitch filtering is on
auto	Enables automatic glitch filtering. This option will ignore any value specified in `-glitch_threshold`
false	Glitch filtering is off

`-glitch_threshold {value}`

This option is only available when `-glitch_filtering` is set to *`true`*. Specifies the glitch filtering value (in ps).

`-power_summary {value}`

Specifies whether to include the power summary, which shows the static and dynamic values in the report. The following table shows the acceptable values for this argument:

Value	Description
true	Includes the power summary in the report
false	Does not include the power summary in the report

`-auto_detect_sampling_period {value}`

Specifies whether to automatically detect the sampling period. The following table shows the acceptable values for this argument:

Value	Description
true	Automatically detects the sampling period
false	Does not automatically detect the sampling period

`-sampling_clock {}`

Specifies the sampling clock.

`-sampling_rate_per_period {value}`

Specifies whether to set the sampling rate per period. The following table shows the acceptable values for this argument:

Value	Description
true	Specifies the sampling rate per period
false	Specifies the sampling rate per half period

`-sampling_offset {value}`

Specifies the offset used to calculate the sampling offset (in ps).

`-sampling_period {value}`

Specifies the offset used to calculate the sampling period (in ps).

`-use_only_local_extrema {value}`

Specifies whether to limit the history size by keeping only local extrema. The following table shows the acceptable values for this argument:

Value	Description
true	Limits the history size by keeping only local extrema
false	Does not limit the history size by keeping only local extrema

`-use_power_threshold {value}`

Specifies whether to limit the history size by setting a power threshold. The following table shows the acceptable values for this argument:

Value	Description
true	Limits the history size by setting a power threshold
false	Does not limit the history size by setting a power threshold

`-power_threshold {value}`

Sets the power threshold value.

`-opmode {value}`

Use this option to specify the mode from which the operating conditions are extracted to generate the report.

`{filename}`

Specifies the name of the report.

Supported Families

IGLOO, ProASIC3, SmartFusion, Fusion

Exceptions

None

Examples

This example generates a cycle accurate power report named report_power_cycle_based.txt.

```
report -type "power_cycle_based" -vcd_file "D:/FPU/mul.vcd" -style "Text" -partial_parse
"TRUE" -start_time "0.05" -end_time "1.00" -auto_detect_top_level_name "TRUE" -
glitch_filtering "FALSE" -glitch_threshold "100" -auto_detect_sampling_period "TRUE" -
sampling_clock "clk" -sampling_rate_per_period "TRUE" -sampling_offset "0.00" -
```

```
sampling_period "10000.00" -use_only_local_extrema "TRUE" -use_power_threshold "TRUE" -
power_threshold "0.00" -opmode "Active" \ {D:/FPU/report_power_cycle_based.txt}
```

See Also

Tcl documentation conventions

Designer Tcl Command Reference

report (Datasheet) using SmartTime

Tcl command; creates a datasheet report.

```
report -type datasheet filename \
[-format value]
```

Arguments

`filename`

Specifies the name and destination of the datasheet report.

`-format value`

Specifies the output format of the generated the report.

Value	Description
text	Generates a text report; text is the default value
csv	Generates the report in a comma-separated value format which you can import into a spreadsheet

Supported Families

IGLOO, ProASIC3, SmartFusion2, SmartFusion, Fusion

Exceptions

None

Examples

The following example generates a datasheet report named datasheet.txt.

```
report -type datasheet -format Text datasheet.txt
```

See Also

Tcl documentation conventions

report (Timing) using SmartTime

report (Timing violations) using SmartTime

Designer Tcl Command Reference

report (Power Scenario)

Tcl command; creates a scenario power report for a previously defined scenario. It includes information about the global device and SmartPower preferences selection, and the average power consumption and the excepted battery life for this sequence.

```
report -type power_scenario \
[-powerunit {value}] \
[-frequnit {value}] \
[-opcond {value}] \
[-toggle {value}] \
[-scenario {value}] \
[-style {value}] \
[-battery_life {value}] \
[-battery_capacity {value}] \
[-rail_breakdown {value}] \
[-type_breakdown {value}] \
[-mode_breakdown {value}] \
[-opcond_summary {value}] \
{filename}
```

Arguments

`-type power_scenario`

Specifies the type of report to generate is a scenario power report.

`-powerunit {value}`

Specifies the unit in which power is set. The following table shows the acceptable values for this argument:

Value	Description
W	The power unit is set to watts
mW	The power unit is set to milliwatts
uW	The power unit is set to microwatts

`-frequnit {value}`

Specifies the unit in which frequency is set. The following table shows the acceptable values for this argument:

Value	Description
Hz	The frequency unit is set to hertz
kHz	The frequency unit is set to kilohertz
MHz	The frequency unit is set to megahertz

`-toggle {value}`

Specifies the toggle. The following table shows the acceptable values for this argument:

Value	Description
true	The toggle is set to true
false	The toggle is set to false

`-scenario{value}`

Specifies a scenario that the report is generated from.

`-style {value}`

Specifies the format in which the report will be exported. The following table shows the acceptable values for this argument:

Value	Description
Text	The report will be exported as Text file
CSV	The report will be exported as CSV file

`-battery_life {value}`

Specifies whether to include the battery life summary in the report. The following table shows the acceptable values for this argument:

Value	Description
true	Includes the battery life summary in the report
false	Does not include the battery life summary in the report

`-battery_capacity {value}`

Specifies the battery capacity in A*H.

`-rail_breakdown {value}`

Specifies whether to include the breakdown by rail summary in the report. The following table shows the acceptable values for this argument:

Value	Description
true	Includes the breakdown by rail summary in the report
false	Does not include the breakdown by rail summary in the report. This is the default value.

`-type_breakdown {value}`

Specifies whether to include the breakdown by type summary in the report. The following table shows the acceptable values for this argument:

Value	Description
true	Includes the breakdown by type summary in the report
false	Does not include the breakdown by type summary in the report. This is the default value.

`-mode_breakdown {value}`

Specifies whether to include a breakdown by mode in the report. The following table shows the acceptable values for this argument:

Value	Description
true	Includes the breakdown by mode in the report
false	Does not include the breakdown by mode in the report. This is the default value.

`-opcond_summary {value}`

Specifies whether to include the operating conditions summary in the report. The following table shows the acceptable values for this argument:

Value	Description
true	Includes the operating conditions summary in the report
false	Does not include the operating conditions summary in the report

`{filename.rpt}`

Specifies the name of the report.

Supported Families

IGLOO, ProASIC3, SmartFusion, Fusion

Notes

- Flash*Freeze, Sleep, and Shutdown are available only for certain families and devices.
- Worst and Best are available only for certain families and devices.

Exceptions

None

Examples

This example generates a scenario power report named report.txt for my_scenario

```
report -type power_scenario -scenario my_scenario -rail_breakdown true -type_breakdown
true -mode_breakdown true -style text -battery_capacity 10 report.txt
```

See Also

Scenario Power Report

report (Timing) using SmartTime

Tcl command; creates a timing report.

```
report -type timing \
[-print_summary value]\
[-analysis value]\
[-use_slack_threshold value]\
[-slack_threshold value]\
[-print_paths value]\
[-max_paths value]\
[-max_expanded_paths value]\
[-include_user_sets value]\
[-include_pin_to_pin value]\
[-include_clock_domains value]\
[-select_clock_domains value]\
[-clock_domain clock_domain_list]
[-format value]
filename
```

Arguments

`-type timing`

Specifies the type of report to generate.

`-print_summary` ***value***

Specifies whether to print the summary section in the timing report.

Value	Description
yes	Includes summary section in the timing report (the default value).
no	Excludes summary section in the timing report

`-analysis` ***value***

Specifies whether the report will consider minimum analysis or maximum analysis.

Value	Description
min	Timing report considers minimum analysis
max	Timing report considers maximum analysis (the default value)

`-use_slack_threshold` ***value***

Specifies whether the report will consider slack threshold.

Value	Description
yes	Includes slack threshold in the timing report.
no	Excludes slack threshold in the timing report (the default value)

`-slack_threshold` ***value***

Specifies the threshold to consider when reporting path slacks. This is a floating-point number in nanoseconds (ns). By default, there is no threshold (all slacks are reported).

`-print_paths` ***value***

Specifies whether the path section (clock domains and in-to-out paths) will be printed in the timing report.

Value	Description
yes	Includes path section in the timing report (the default value)
no	Excludes path sections from the timing report

`-max_paths` ***value***

Defines the maximum number of paths to display for each set. This is a positive integer value greater than zero. The default is 5.

`-max_expanded_paths` ***value***

Defines the number of paths to expand per set. This is a positive integer value greater than zero. The default is 1.

`-include_user_sets` ***value***

Defines whether to include the user defined sets in the timing report.

Value	Description
yes	Includes user defined sets in the timing report (the default value)
no	Excludes user defined sets from the timing report

`-include_pin_to_pin` *`value`*

Specifies whether to show pin-to-pin paths in the timing report.

Value	Description
yes	Includes pin-to-pin paths in the timing report (the default value).
no	Excludes pin-to-pin paths from the timing report

`-include_clock_domains` *`value`*

Defines whether to include clock domains in the timing report.

Value	Description
yes	Includes clock domains
no	Excludes clock domains from the timing report

`-select_clock_domains` *`value`*

Specifies whether to show the clock domain list in the timing report.

Value	Description
yes	Includes the clock domain list in the timing report
no	Excludes the clock domain list from the timing report (the default value)

`-clock_domain clock_domain_list`

Defines the clock domain to be considered in the clock domain section. The domain list is a series of strings with domain names separated by spaces. Both the summary and the path sections in the timing report display only the listed clock domains.

`-format` *`value`*

Specifies the output format of the generated the report.

Value	Description
text	Generates a text report; text is the default value
csv	Generates the report in a comma-separated value format which you can import into a spreadsheet

`filename`

Specifies the name and destination of the timing report.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Examples

The following example generates a timing report named timing_report.txt. The report does not print the summary section. It includes a max-delay analysis and only reports paths with a slack value less than 0.50 ns. It reports a maximum of 3 paths per section and does not report any expanded paths. It only reports timing information for the clock domains count8_clock and count2_clk.

```
report -type timing -print_summary no \
-analysis max \
-use_slack_threshold yes \
-slack_threshold 0.50 \
-print_paths yes -max_paths 3 \
-max_expanded_paths 0 \
-include_user_sets yes \
-include_pin_to_pin yes \
-select_clock_domains yes \
-clock_domain {count8_clock count2_clk} \
timing_report.txt
```

See Also

Tcl documentation conventions

`report (Timing violations) using SmartTime`

`report (Datasheet) using SmartTime`

Designer Tcl Command Reference

report (Timing violations) using SmartTime

Tcl command; creates a timing violations report.

```
report -type timing_violations \
[-analysis value]\
[-use_slack_threshold value]\
[-slack_threshold value]\
[-limit_max_paths value]\
[-max_paths value]\
[-max_expanded_paths value] \
[-format value]
filename
```

Arguments

`-type timing_violations`

Specifies the type of report to generate.

`-analysis value`

Specifies whether to consider minimum analysis or maximum analysis in the timing violations report.

Value	Description

Value	Description
min	Timing report considers minimum analysis
max	Timing report considers maximum analysis (the default value)

`-use_slack_threshold` *`value`*

Specifies whether to consider the slack threshold in the timing violations report.

Value	Description
yes	Includes slack threshold in the timing violations report
no	Excludes slack threshold in the timing violations report (the default value)

`-slack_threshold` *`value`*

Specifies the threshold to consider when reporting path slacks. This value is a floating-point number in nanoseconds (ns). By default, there is no threshold (all slacks reported).

`-limit_max_paths` *`value`*

Specifies if the paths are limited by the number of paths.

Value	Description
yes	Limits the maximum number of paths to report
no	Specifies that there is no limit to the number of paths to report (the default value)

`-max_paths` *`value`*

Specifies the maximum number of paths to display for each set. This value is a positive integer value greater than zero. Default is 100.

`-max_expanded_paths` *`value`*

Specifies the number of paths to expand per set. This value is a positive integer value greater than zero. The default is 0.

`-format` *`value`*

Specifies the output format of the generated report.

Value	Description
text	Generates a text report; text is the default value
csv	Generates the report in a comma-separated value format which you can import into a spreadsheet

`filename`

Specifies the name and destination of the timing violations report.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Examples

The following example generates a timing violations report named timg_viol.txt. The report considers an analysis using maximum delays and does not filter paths based on slack threshold. It reports 2 paths per section and 1 expanded path per section.

```
report -type timing_violations \
-analysis max -use_slack_threshold no \
-limit_max_paths -yes \
-max_paths 2 \
-max_expanded_paths 1 \
timg_viol.txt
```

See Also

Tcl documentation conventions

`report (Timing) using SmartTime`

`report (Datasheet) using SmartTime`

Designer Tcl Command Reference

save_design

Tcl command; the save_design command saves the current design in Designer to a file. If filename is not a complete path name, the ADB file is written into the current working directory.

```
save_design filename
```

Arguments

The design is written to a file denoted by the variable filename as an ADB file.

Supported Families

SmartFusion, IGLOO, ProASIC3 and Fusion

Exceptions

None

Example

Example 1: Saves the design to a file “test.adb” in the current folder.

```
save_design {test.adb}
```

Example 2: Save design and check if it saved successfully.

```
set designFile {d:/test/my_design.adb}
if  { [catch { save_design $designFile }] {
              Puts “Failed to save design”
              # Handle Failure
} else {
              puts “Design saved successfully”
              # Proceed to make further changes
}
```

See Also

`close_design`

`new_design`

`open_design`

Designer Tcl Command Reference

set_clock_latency

Tcl command; defines the delay between an external clock source and the definition pin of a clock within SmartTime.

```
set_clock_latency -source [-rise][-fall][-early][-late] delay clock
```

Arguments

`-source`

Specifies the source latency on a clock pin, potentially only on certain edges of the clock.

`-rise`

Specifies the edge for which this constraint will apply. If neither or both rise are passed, the same latency is applied to both edges.

`-fall`

Specifies the edge for which this constraint will apply. If neither or both rise are passed, the same latency is applied to both edges.

`-invert`

Specifies that the generated clock waveform is inverted with respect to the reference clock.

`-late`

Optional. Specifies that the latency is late bound on the latency. The appropriate bound is used to provide the most pessimistic timing scenario. However, if the value of "-late" is less than the value of "-early", optimistic timing takes place which could result in incorrect analysis. If neither or both "-early" and "-late" are provided, the same latency is used for both bounds, which results in the latency having no effect for single clock domain setup and hold checks.

`-early`

Optional. Specifies that the latency is early bound on the latency. The appropriate bound is used to provide the most pessimistic timing scenario. However, if the value of "-late" is less than the value of "-early", optimistic timing takes place which could result in incorrect analysis. If neither or both "-early" and "-late" are provided, the same latency is used for both bounds, which results in the latency having no effect for single clock domain setup and hold checks.

`delay`

Specifies the latency value for the constraint.

`clock`

Specifies the clock to which the constraint is applied. This clock must be constrained.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Description

Clock source latency defines the delay between an external clock source and the definition pin of a clock within SmartTime. It behaves much like an input delay constraint. You can specify both an "early" delay and a"late" delay for this latency, providing an uncertainty which SmartTime propagates through its calculations. Rising and falling edges of the same clock can have different latencies. If only one value is provided for the clock source latency, it is taken as the exact latency value, for both rising and falling edges.

Exceptions

None

Examples

The following example sets an early clock source latency of 0.4 on the rising edge of main_clock. It also sets a clock source latency of 1.2, for both the early and late values of the falling edge of main_clock. The late value for the clock source latency for the falling edge of main_clock remains undefined.

```
set_clock_latency –source –rise –early 0.4 { main_clock  }
set_clock_latency –source –fall    1.2 { main_clock }
```

See Also

`create_clock`

`create_generated_clock`

Tcl Command Documentation Conventions

Designer Tcl Command Reference

set_clock_uncertainty

Tcl command; specifies a clock-to-clock uncertainty between two clocks (from and to) and returns the ID of the created constraint if the command succeeded.

```
set_clock_uncertainty uncertainty -from | -rise_from | -fall_from from_clock_list -to | -
rise_to | -fall_to to_clock_list -setup {value} -hold {value}
```

Arguments

`uncertainty`

Specifies the time in nanoseconds that represents the amount of variation between two clock edges.

`-from`

Specifies that the clock-to-clock uncertainty applies to both rising and falling edges of the source clock list. Only one of the `-from`, `-rise_from`, or `-fall_from` arguments can be specified for the constraint to be valid.

`-rise_from`

Specifies that the clock-to-clock uncertainty applies only to rising edges of the source clock list. Only one of the `-from`, `-rise_from`, or `-fall_from` arguments can be specified for the constraint to be valid.

`-fall_from`

Specifies that the clock-to-clock uncertainty applies only to falling edges of the source clock list. Only one of the `-from`, `-rise_from`, or `-fall_from` arguments can be specified for the constraint to be valid.

`from_clock_list`

Specifies the list of clock names as the uncertainty source.

`-to`

Specifies that the clock-to-clock uncertainty applies to both rising and falling edges of the destination clock list. Only one of the `-to`, `-rise_to` , or `-fall_to` arguments can be specified for the constraint to be valid.

`-rise_to`

Specifies that the clock-to-clock uncertainty applies only to rising edges of the destination clock list. Only one of the `-to`, `-rise_to` , or `-fall_to` arguments can be specified for the constraint to be valid.

`-fall_to`

Specifies that the clock-to-clock uncertainty applies only to falling edges of the destination clock list. Only one of the `-to`, `-rise_to` , or `-fall_to` arguments can be specified for the constraint to be valid.

`to_clock_list`

Specifies the list of clock names as the uncertainty destination.

`-setup`

Specifies that the uncertainty applies only to setup checks. If none or both `-setup` and `-hold` are present, the uncertainty applies to both setup and hold checks.

`-hold`

Specifies that the uncertainty applies only to hold checks. If none or both `-setup` and `-hold` are present, the uncertainty applies to both setup and hold checks.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Description

The set_clock_uncertainty command sets the timing uncertainty between two clock waveforms or maximum clock skew. Timing between clocks have no uncertainty unless you specify it.

Exceptions

None

Examples

```
set_clock_uncertainty 10 -from Clk1 -to Clk2
set_clock_uncertainty 0 -from Clk1 -fall_to { Clk2 Clk3 } -setup
set_clock_uncertainty 4.3 -fall_from { Clk1 Clk2 } -rise_to *
set_clock_uncertainty 0.1 -rise_from [ get_clocks { Clk1 Clk2 } ] -fall_to { Clk3 Clk4 }
-setup
set_clock_uncertainty 5 -rise_from Clk1 -to [ get_clocks {*} ]
```

See Also

`create_clock`

`create_generated_clock`

`remove_clock_uncertainty`

Designer Tcl Command Reference

set_current_scenario

Tcl command; specifies the timing scenario for the Timing Analyzer to use. All commands that follow this command will apply to the specified timing scenario.

```
set_current_scenario name
```

Arguments

`name`

Specifies the name of the timing scenario to which to apply all commands from this point on.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Description

A timing scenario is a set of timing constraints used with a design. If the specified scenario is already the current one, this command has no effect.

After setting the current scenario, constraints can be listed, added, or removed, the checker can be invoked on the set of constraints, and so on.

This command uses the specified timing scenario to compute timing analysis.

Exceptions

None

Example

```
set_current_scenario scenario_A
```

See Also

`get_current_scenario`

Tcl Command Documentation Conventions

Designer Tcl Command Reference

set_defvar (Designer Only)

Tcl command; the set_defvar command sets an internal variable in the Designer system. You must specify at least one argument for this command.

```
set_defvar variable value
```

Arguments

`Variable` must be a valid Designer internal variable and could be accompanied by an optional value. If the *`value`* is provided, the *`variable`* is set the value. If the *`value`* is null the *`variable`* is reset.

Supported Families

SmartFusion, IGLOO, ProASIC3 and Fusion

Exceptions

None.

Example

Example 1:

```
set_defvar “FORMAT” “VHDL”
Sets the FORMAT internal variable to VHDL.
```

Example 2:

```
set variableToSet "DESIGN"
set valueOfVariable “VHDL”
set_defvar $variableToSet $valueOfVariable
```

These commands set the FORMAT variable to VHDL, shows the use of variables for this command.

See Also

`get_defvar`

Designer Tcl Command Reference

set_design

Tcl command; this set_design command specifies the design name, family and path in which Designer will process the design. This step is absolutely required before importing the source files.

```
set_design -name design_name -family family_name –pathpath_name
```

Note: Note: You need all three arguments for this command to set up your design.

Arguments

`-name design_name`

The name of the design. This is used as the base name for most of the files generated from Designer.

`-family family_name`

The device family for which the design is being targeted.

`-path path_name`

The physical path of the directory in which the design files will be created.

Supported Families

SmartFusion, IGLOO, ProASIC3 and Fusion

Example

Example 1: Sets up the design and checks if there are any errors

```
set_design -name "test" -family "Axcelerator" -path {.}
set desName “test
set famName “ACT3”
set path {d:/examples/test}

if  { [catch { set_design -name $desName -family $famName -path $path }] {
              Puts “Failed setup design”
              # Handle Failure
} else {
              puts “Design setup successful”
              # Proceed to Import source files
}
```

See Also

new_design

set_device

Designer Tcl Command Reference

set_device

Tcl command; the set_device command specifies the type of device and its parameters. You must specify at least one option for this command. Some of the options may not apply for certain families that do not support the features.

```
set_device -family family_name -die die_name -package package_name -speed speed_grade -voltage
voltage -voltrange volt_range -temprange temp_range -iostd default_io_std -pci value -jtag value
-probe value -trst value -radexp value  -vcci_1.2_voltrange value -vcci_1.2_widerange value -
vcci_1.5_voltrange value -vcci_1.8_voltrange value -vcci_2.5_voltrange value -
vcci_3.3_voltrange value -vcci_3.3_widerange value
```

Arguments

`-family family_name`

Specifies the name of the FPGA device family.

`-die die_name`

Specifies the part name.

`-package package_name`

Specifies the selected package for the device.

`-speed speed_grade`

`Specifies the speed grade of the part.`

`-voltage` *`voltage`*

Specifies the core voltage of the device. You can also use it to define the I/O voltage of the part. For example, if you are using a RTSX with a 3.3 to 2.5 voltage, you can use

-voltage 3.3/2.5

`-voltrange` *`volt_range`*

Specifies the voltage range to be applied for the device. It is generally MIL, COM and IND denoting Military, Commercial and Industrial respectively.

Alternatively, you can also specify custom values for Best, Typical, and Worst: `-voltrange "1.60 1.50 1.40"`

`-temprange` *`temp_range`*

Specifies the temperature range to be applied for the device. Temperature ranges are MIL, COM and IND denoting Military, Commercial and Industrial respectively. Automotive applications generally use the Automotive, TGrade1, or TGrade2 temperature range.

`-iostd` *`default_io_std`*

Specifies the default I/O standard of the part.

`-pci` *`value`*

Used if the device needs to configure the I/Os for PCI specification. This parameter is equivalent to setting your I/O attributes to PCI in the Project Settings. Values are summarized in the table below.

Value	Description
yes	Device is configured for PCI specification
no	Device is not configured for PCI specification

`-jtag` *`value`*

Specifies if pins need to be reserved for JTAG. Values are summarized in the table below.

Value	Description
yes	Pins are reserved for JTAG
no	Pins are not reserved for JTAG

`-probe` *`value`*

Specifies if the pins need to be preserved for probing. Values are summarized in the table below.

Value	Description
yes	Pins are preserved for probing
no	Pins not preserved for probing

`-trst` *`value`*

Specifies if the pins need to be reserved for JTAG test reset. Values are summarized in the table below.

Value	Description
yes	Pins are preserved for JTAG test reset
no	Pins are not preserved for JTAG test reset

`-radexp` *`value`*

Specifies the radiation value (in Krad) for radiation tolerant devices.

`-vcci_1.2_voltrange` *`value`* `-vcci_1.5_voltrange`*`value`* `-vcci_1.8_voltrange`*`value`* `-vcci_2.5_voltrange`*`value`*`-vcci_3.3_voltrange`*`value`*

Specifies the voltage range for VCCIx.x. Values are summarized in the table below.

Value	Description
MIL	Sets the voltage range for VCCIx.x to Military
COM	Sets the voltage range for VCCIx.x to Commercial
IND	Sets the voltage range for VCCIx.x to Industrial

Alternatively, you can also specify custom values for Best, Typical, and Worst: `-vcci_x.x_voltrange "1.26 1.20 1.14"`

`-vcci_1.2_widerange` *`value`*

Specifies the voltage range for VCCI1.2 as wide range. Values are summarized in the table below.

Value	Description
yes	Specifies the voltage range for VCCI1.2 as wide range and sets the def variable IS_VCCI_1.2_WR as "1"
no	Does not specify the voltage range for VCCI1.2 as wide range

`-vcci_3.3_widerange` *`value`*

Specifies the voltage range for VCCI3.3 as wide range. Values are summarized in the table below.

Value	Description
yes	Specifies the voltage range for VCCI3.3 as wide range and sets the def variable IS_VCCI_3.3_WR as "1"
no	Does not specify the voltage range for VCCI3.3 as wide range

Supported Families

SmartFusion, IGLOO, ProASIC3 and Fusion

Example

Example 1: Setting up a design.

```
set_device -die "APA075" -package "208 PQFP" -speed "STD" -voltage "2.5" \
-jtag "yes" -trst "yes" -temprange "COM" -voltrange "COM"\
-vcci_1.2_voltrange "COM" -vcci_1.2_widerange "no" -vcci_1.5_voltrange "1.60 1.50 1.40"
```

See Also

`new_design`

`set_design`

Designer Tcl Command Reference

set_disable_timing

Tcl command; disables timing arcs within a cell and returns the ID of the created constraint if the command succeeded.

```
set_disable_timing -from value -to value name
```

Arguments

`-from from_port`

Specifies the starting port. The –from and –to arguments must either both be present or both omitted for the constraint to be valid.

`-to to_port`

Specifies the ending port. The –from and –to arguments must either both be present or both omitted for the constraint to be valid.

`name`

Specifies the cell name where the timing arcs will be disabled.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Example

```
set_disable_timing -from A -to Y a2
```

See Also

Tcl documentation conventions

Designer Tcl Command Reference

set_false_path

Tcl command; identifies paths that are considered false and excluded from the timing analysis in the current timing scenario.

```
set_false_path [-from from_list] [-through through_list] [-to to_list]
```

Arguments

`-from from_list`

Specifies a list of timing path starting points. A valid timing starting point is a clock, a primary input, an inout port, or a clock pin of a sequential cell.

`-through through_list`

Specifies a list of pins, ports, cells, or nets through which the disabled paths must pass.

`-to  to_list`

Specifies a list of timing path ending points. A valid timing ending point is a clock, a primary output, an inout port, or a data pin of a sequential cell.

Supported Families

IGLOO, ProASIC3, SmartFusion, Fusion

Description

The `set_false_path` command identifies specific timing paths as being false. The false timing paths are paths that do not propagate logic level changes. This constraint removes timing requirements on these false paths so that they are not considered during the timing analysis. The path starting points are the input ports or register clock pins, and the path ending points are the register data pins or output ports. This constraint disables setup and hold checking for the specified paths.

The false path information always takes precedence over multiple cycle path information and overrides maximum delay constraints. If more than one object is specified within one -through option, the path can pass through any objects.

You must specify at least one of the `–from`, `-to`, or `–through` arguments for this constraint to be valid.

Examples

The following example specifies all paths from clock pins of the registers in clock domain clk1 to data pins of a specific register in clock domain clk2 as false paths:

```
set_false_path –from [get_clocks {clk1}] –to reg_2:D
```

The following example specifies all paths through the pin U0/U1:Y to be false:

```
set_false_path -through U0/U1:Y
```

See Also

[Tcl Command Documentation Conventions]()

[Designer Tcl Command Reference]()

set_input_delay

Tcl command; creates an input delay on a port list by defining the arrival time of an input relative to a clock in the current scenario.

```
set_input_delay delay_value -clock clock_ref [–max] [–min] [–clock_fall] input_list
```

Arguments

`delay_value`

Specifies the arrival time in nanoseconds that represents the amount of time for which the signal is available at the specified input after a clock edge.

`-clock` *`clock_ref`*

Specifies the clock reference to which the specified input delay is related. This is a mandatory argument. If you do not specify -max or -min options, the tool assumes the maximum and minimum input delays to be equal.

`-max`

Specifies that delay_value refers to the longest path arriving at the specified input. If you do not specify -max or -min options, the tool assumes maximum and minimum input delays to be equal.

`-min`

Specifies that delay_value refers to the shortest path arriving at the specified input. If you do not specify -max or -min options, the tool assumes maximum and minimum input delays to be equal.

`-clock_fall`

Specifies that the delay is relative to the falling edge of the clock reference. The default is the rising edge.

`input_list`

Provides a list of input ports in the current design to which delay_value is assigned. If you need to specify more than one object, enclose the objects in braces ({}).

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion and IGLOOe, except ProASIC3 nano and ProASIC3L

Description

The set_input_delay command sets input path delays on input ports relative to a clock edge. This usually represents a combinational path delay from the clock pin of a register external to the current design. For in/out (bidirectional) ports, you can specify the path delays for both input and output modes. The tool adds input delay to path delay for paths starting at primary inputs.

A clock is a singleton that represents the name of a defined clock constraint. This can be:

- a single port name used as source for a clock constraint
- a single pin name used as source for a clock constraint; for instance reg1:CLK. This name can be hierarchical (for instance toplevel/block1/reg2:CLK)
- an object accessor that will refer to one clock: [get_clocks {clk}]

Examples

The following example sets an input delay of 1.2ns for port data1 relative to the rising edge of CLK1:

```
set_input_delay 1.2 -clock [get_clocks CLK1] [get_ports data1]
```

The following example sets a different maximum and minimum input delay for port IN1 relative to the falling edge of CLK2:

```
set_input_delay 1.0 -clock_fall -clock CLK2 –min {IN1}
set_input_delay 1.4 -clock_fall -clock CLK2 –max {IN1}
```

See Also

`set_output_delay`

Tcl Command Documentation Conventions

Designer Tcl Command Reference

set_max_delay

Tcl command; specifies the maximum delay for the timing paths in the current scenario.

```
set_max_delay delay_value [-from from_list] [-to to_list] [-through through_list]
```

Arguments

`delay_value`

Specifies a floating point number in nanoseconds that represents the required maximum delay value for specified paths.

- If the path starting point is on a sequential device, the tool includes clock skew in the computed delay.
- If the path starting point has an input delay specified, the tool adds that delay value to the path delay.
- If the path ending point is on a sequential device, the tool includes clock skew and library setup time in the computed delay.
- If the ending point has an output delay specified, the tool adds that delay to the path delay.

`-from` *`from_list`*

Specifies a list of timing path starting points. A valid timing starting point is a clock, a primary input, an inout port, or a clock pin of a sequential cell.

`-to` *`to_list`*

Specifies a list of timing path ending points. A valid timing ending point is a clock, a primary output, an inout port, or a data pin of a sequential cell.

`-through` *`through_list`*

Specifies a list of pins, ports, cells, or nets through which the timing paths must pass.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Description

This command specifies the required maximum delay for timing paths in the current design. The path length for any startpoint in from_list to any endpoint in to_list must be less than delay_value.

The timing engine automatically derives the individual maximum delay targets from clock waveforms and port input or output delays.

The maximum delay constraint is a timing exception. This constraint overrides the default single cycle timing relationship for one or more timing paths. This constraint also overrides a multicycle path constraint.

You must specify at least one of the `–from`, `-to`, or `–through` arguments for this constraint to be valid.

Examples

The following example sets a maximum delay by constraining all paths from ff1a:CLK or ff1b:CLK to ff2e:D with a delay less than 5 ns:

```
set_max_delay 5 -from {ff1a:CLK ff1b:CLK} -to {ff2e:D}
```

The following example sets a maximum delay by constraining all paths to output ports whose names start by “out” with a delay less than 3.8 ns:

```
set_max_delay 3.8 -to [get_ports out*]
```

See Also

`set_min_delay`

`remove_max_delay`

Tcl Command Documentation Conventions

Designer Tcl Command Reference

set_min_delay

Tcl command; specifies the minimum delay for the timing paths in the current scenario.

```
set_min_delay delay_value [-from from_list] [-to to_list] [-through through_list]
```

Arguments

`delay_value`

Specifies a floating point number in nanoseconds that represents the required minimum delay value for specified paths.

- If the path starting point is on a sequential device, the tool includes clock skew in the computed delay.
- If the path starting point has an input delay specified, the tool adds that delay value to the path delay.

- If the path ending point is on a sequential device, the tool includes clock skew and library setup time in the computed delay.
- If the ending point has an output delay specified, the tool adds that delay to the path delay.

`-from` *`from_list`*

Specifies a list of timing path starting points. A valid timing starting point is a clock, a primary input, an inout port, or a clock pin of a sequential cell.

`-to` *`to_list`*

Specifies a list of timing path ending points. A valid timing ending point is a clock, a primary output, an inout port, or a data pin of a sequential cell.

`-through` *`through_list`*

Specifies a list of pins, ports, cells, or nets through which the timing paths must pass.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Description

This command specifies the required minimum delay for timing paths in the current design. The path length for any startpoint in from_list to any endpoint in to_list must be less than delay_value.

The timing engine automatically derives the individual minimum delay targets from clock waveforms and port input or output delays.

The minimum delay constraint is a timing exception. This constraint overrides the default single cycle timing relationship for one or more timing paths. This constraint also overrides a multicycle path constraint.

You must specify at least one of the `–from`, `-to`, or `–through` arguments for this constraint to be valid.

Examples

The following example sets a minimum delay by constraining all paths from ff1a:CLK or ff1b:CLK to ff2e:D with a delay less than 5 ns:

```
set_min_delay 5 -from {ff1a:CLK ff1b:CLK} -to {ff2e:D}
```

The following example sets a minimum delay by constraining all paths to output ports whose names start by “out” with a delay less than 3.8 ns:

```
set_min_delay 3.8 -to [get_ports out*]
```

See Also

`set_max_delay`

`remove_min_delay`

Tcl Command Documentation Conventions

Designer Tcl Command Reference

set_multicycle_path

Tcl command; defines a path that takes multiple clock cycles in the current scenario.

```
set_multicycle_path ncycles [-setup] [-hold] [-from from_list[–through through_list[-to
to_list
```

Arguments

`ncycles`

Specifies an integer value that represents a number of cycles the data path must have for setup or hold check. The value is relative to the starting point or ending point clock, before data is required at the ending point.

`-setup`

Optional. Applies the cycle value for the setup check only. This option does not affect the hold check. The default hold check will be applied unless you have specified another set_multicycle_path command for the hold value.

`-hold`

Optional. Applies the cycle value for the hold check only. This option does not affect the setup check.

Note: Note: If you do not specify "-setup" or "-hold", the cycle value is applied to the setup check and the default hold check is performed (*ncycles* -1).

`-from` *`from_list`*

Specifies a list of timing path starting points. A valid timing starting point is a clock, a primary input, an inout port, or a clock pin of a sequential cell.

`-through` *`through_list`*

Specifies a list of pins or ports through which the multiple cycle paths must pass.

`-to` *`to_list`*

Specifies a list of timing path ending points. A valid timing ending point is a clock, a primary output, an inout port, or a data pin of a sequential cell.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Description

Setting multiple cycle paths constraint overrides the single cycle timing relationships between sequential elements by specifying the number of cycles that the data path must have for setup or hold checks. If you change the multiplier, it affects both the setup and hold checks.

False path information always takes precedence over multiple cycle path information. A specific maximum delay constraint overrides a general multiple cycle path constraint.

If you specify more than one object within one -through option, the path passes through any of the objects.

You must specify at least one of the `–from`, `-to`, or `–through` arguments for this constraint to be valid.

Exceptions

- Multiple priority management is not supported in Microsemi SoC designs. All multiple cycle path constraints are handled with the same priority.

Examples

The following example sets all paths between reg1 and reg2 to 3 cycles for setup check. Hold check is measured at the previous edge of the clock at reg2.

```
set_multicycle_path 3 -from [get_pins {reg1}] –to [get_pins {reg2}]
```

The following example specifies that four cycles are needed for setup check on all paths starting at the registers in the clock domain ck1. Hold check is further specified with two cycles instead of the three cycles that would have been applied otherwise.

```
set_multicycle_path 4 -setup -from [get_clocks {ck1}]
set_multicycle_path 2 -hold -from [get_clocks {ck1}]
```

See Also

`remove_multicycle_path`

Tcl Command Documentation Conventions

Designer Tcl Command Reference

set_output_delay

Tcl command; defines the output delay of an output relative to a clock in the current scenario.

```
set_output_delay delay_value -clock clock_ref [–max] [–min] [–clock_fall] output_list
```

Arguments

delay_value

Specifies the amount of time before a clock edge for which the signal is required. This represents a combinational path delay to a register outside the current design plus the library setup time (for maximum output delay) or hold time (for minimum output delay).

-clock *clock_ref*

Specifies the clock reference to which the specified output delay is related. This is a mandatory argument. If you do not specify -max or -min options, the tool assumes the maximum and minimum input delays to be equal.

-max

Specifies that delay_value refers to the longest path from the specified output. If you do not specify -max or -min options, the tool assumes the maximum and minimum output delays to be equal.

-min

Specifies that delay_value refers to the shortest path from the specified output. If you do not specify -max or -min options, the tool assumes the maximum and minimum output delays to be equal.

-clock_fall

Specifies that the delay is relative to the falling edge of the clock reference. The default is the rising edge.

output_list

Provides a list of output ports in the current design to which delay_value is assigned. If you need to specify more than one object, enclose the objects in braces ({}).

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Description

The set_output_delay command sets output path delays on output ports relative to a clock edge. Output ports have no output delay unless you specify it. For in/out (bidirectional) ports, you can specify the path delays for both input and output modes. The tool adds output delay to path delay for paths ending at primary outputs.

Examples

The following example sets an output delay of 1.2ns for port OUT1 relative to the rising edge of CLK1:

```
set_output_delay 1.2 -clock [get_clocks CLK1] [get_ports OUT1]
```

The following example sets a different maximum and minimum output delay for port OUT1 relative to the falling edge of CLK2:

```
set_output_delay 1.0 -clock_fall -clock CLK2 –min {OUT1}
set_output_delay 1.4 -clock_fall -clock CLK2 –max {OUT1}
```

See Also

remove_output_delay

set_input_delay

Tcl Command Documentation Conventions

Designer Tcl Command Reference

smartpower_add_new_custom_mode

Tcl command; creates a new custom mode.

```
smartpower_add_new_custom_mode -name {mode_name} -base_mode {base_mode} -description
{mode_description}
```

Arguments

`-name {mode_name}`

Specifies the name of the new custom mode.

`-base_mode {base_mode}`

Specifies the name of the base mode used to create the new custom mode.

`-description {mode_description}`

Specifies the description of the new custom mode.

Supported Families

IGLOO, ProASIC3, SmartFusion, Fusion

Exceptions

None

Examples

This example creates a new custom mode "Cust_1" based on the Active mode:

```
smartpower_add_new_custom_mode -name {Cust_1} -base_mode {Active} -description
{frequency 10 MHz}
```

See Also

`smartpower_remove_custom_mode`

Designer Tcl Command Reference

smartpower_add_new_scenario

Tcl command; creates a new scenario.

```
smartpower_add_new_scenario -name {value} -description {value} -mode {value}
```

Arguments

`-name {value}`

Specifies the name of the new scenario.

`-description {value}`

Specifies the description of the new scenario.

`-mode {<operating mode>:<duration>}+`

Specifies the mode(s) and duration(s) for the specified scenario.

Supported Families

IGLOO, ProASIC3, SmartFusion, Fusion

Exceptions

None

Examples

This example creates a new scenario called myscenario:

```
smartpower_add_new_scenario -name myscenario -description mynewscenario -mode active:30
+ shutdown:30 + active:40
```

See Also

Tcl documentation conventions

Designer Tcl Command Reference

smartpower_add_pin_in_domain

Tcl command; adds a pin into a clock or set domain.

```
smartpower_add_pin_in_domain -pin_name {pin_name} -pin_type {value} –domain_name
{domain_name} -domain_type {value}
```

Arguments

`-pin_name {pin_name}`

Specifies the name of the pin to add to the domain.

`-pin_type {value}`

Specifies the type of the pin to add. The following table shows the acceptable values for this argument:

Value	Description
clock	The pin to add is a clock pin
data	The pin to add is a data pin

`-domain_name {domain_name}`

Specifies the name of the domain in which to add the specified pin.

`-domain_type {value}`

Specifies the type of domain in which to add the specified pin. The following table shows the acceptable values for this argument:

Value	Description
clock	The domain is a clock domain
set	The domain is a set domain

Supported Families

IGLOO, ProASIC3, SmartFusion, Fusion

Notes

- The `domain_name` must be a name of an existing domain.
- The `pin_name` must be a name of a pin that exists in the design.

Exceptions

None

Examples

The following example adds a clock pin to an existing Clock domain:

```
smartpower_add_pin_in_domain -pin_name { XCMP3/U0/U1:Y } -pin_type {clock} –domain_name
{clk1} -domain_type {clock}
```

The following example adds a data pin to an existing Set domain:

```
smartpower_add_pin_in_domain -pin_name {XCMP3/U0/U1:Y} -pin_type {data} -domain_name
{myset} -domain_type {set}
```

See Also

Tcl documentation conventions

Designer Tcl Command Reference

`smartpower_remove_pin_of_domain`

smartpower_change_clock_statistics

Tcl command; changes the default frequencies and probabilities for a specific domain.

```
smartpower_change_clock_statistics -domain_name {value} -clocks_freq {value} -
clocks_proba {value} -registers_freq {value} -registers_proba {value} -set_reset_freq
{value} -set_reset_proba {value} -primaryinputs_freq {value} -primaryinputs_proba {value} -
combinational_freq {value} -combinational_proba {value}
```

Arguments

`-domain_name{value}`

Specifies the domain name in which to initialize frequencies and probabilities.

`-clocks_freq {value}`

Specifies the user input frequency in Hz, KHz, or MHz for all clocks.

`-clocks_proba {value}`

Specifies the user input probability in % for all clocks.

`-registers_freq {value}`

Specifies the user input frequency (in Hz, KHz, or MHz) or the toggle rate (in %). If the unit is not provided and toggle rate is active, the value is handled as a toggle rate; if toggle rate is not active, the value is handled as a frequency.

`-registers_proba {value}`

Specifies the user input probability in % for all registers.

`-set_reset_freq {value}`

Specifies the user input frequency (in Hz, KHz, or MHz) or the toggle rate (in %). If the unit is not provided and toggle rate is active, the value is handled as a toggle rate; if toggle rate is not active, the value is handled as a frequency.

`-set_reset_proba {value}`

Specifies the user input probability in % for all set/reset nets.

`-primaryinputs_freq {value}`

Specifies the user input frequency (in Hz, KHz, or MHz) or the toggle rate (in %). If the unit is not provided and toggle rate is active, the value is handled as a toggle rate; if toggle rate is not active, the value is handled as a frequency.

`-primaryinputs_proba {value}`

Specifies the user input probability in % for all primary inputs.

`-combinational_freq {value}`

Specifies the user input frequency (in Hz, KHz, or MHz) or the toggle rate (in %). If the unit is not provided and toggle rate is active, the value is handled as a toggle rate; if toggle rate is not active, the value is handled as a frequency.

`-combinational_proba {value}`

Specifies the user input probability in % for all combinational combinational output.

Supported Families

IGLOO, ProASIC3, SmartFusion, Fusion

Notes

This command is associated with the functionality of Initialize frequencies and probabilities dialog box.

Exceptions

None

Examples

The following example initializes all clocks withs:

```
smartpower_change_clock_statistics -domain_name {my_domain} -clocks_freq {10 MHz} -
clocks_proba {20} -registers_freq {10 MHz} -registers_proba {20} -set_reset_freq {10
MHz} -set_reset_proba {20} -primaryinputs_freq {10 MHz} -primaryinputs_proba {20} -
combinational_freq {10 MHz} -combinational_proba {20}
```

See Also

Tcl documentation conventions

Designer Tcl Command Reference

smartpower_change_setofpin_statistics

Tcl command; changes the default frequencies and probabilities for a specific set.

```
smartpower_change_setofpin_statistics -domain_name {value} -data_freq {value} -
data_proba {value}
```

Arguments

`-domain_name{value}`

Specifies the domain name in which to initialize data frequencies and probabilities.

`-data_freq {value}`

Specifies the user input data frequency in Hz, KHz, or MHz for all sets of pins.

`-data_proba {value}`

Specifies the user input data probability in % for all sets of pins.

Supported Families

IGLOO, ProASIC3, SmartFusion, Fusion

Notes

This command is associated with the functionality of Initialize frequencies and probabilities dialog box.

Exceptions

None

Examples

The following example initializes all clocks withs:

```
smartpower_change_setofpin_statistics -domain_name {my_domain} -data_freq {10 MHz} -
data_proba {20}
```

See Also

Tcl documentation conventions

Designer Tcl Command Reference

smartpower_commit

Tcl command; saves the changes to the design (.adb) file.

```
smartpower_commit
```

Arguments

None

Supported Families

IGLOO, ProASIC3, SmartFusion, Fusion

Exceptions

None

Examples

```
smartpower_commit
```

See Also

Tcl documentation conventions

Designer Tcl Command Reference

`smartpower_restore`

smartpower_create_domain

Tcl command; creates a new clock or set domain.

```
smartpower_create_domain -domain_type {value} -domain_name {domain_name}
```

Arguments

`-domain_type {value}`

Specifies the type of domain to create. The following table shows the acceptable values for this argument:

Value	Description
clock	The domain is a clock domain
set	The domain is a set domain

`-domain_name {domain_name}`

Specifies the name of the new domain.

Supported Families

IGLOO, ProASIC3, SmartFusion, Fusion

Notes

The `domain name` cannot be the name of an existing domain.

The domain `type` must be either clock or set.

Exceptions

None

Examples

The following example creates a new clock domain named "clk2":

```
smartpower_create_domain -domain_type {clock} -domain_name {clk2}
```

The following example creates a new set domain named "myset":

```
smartpower_create_domain -domain_type {set} -domain_name {myset}
```

See Also

Tcl documentation conventions

Designer Tcl Command Reference

`smartpower_remove_domain`

smartpower_edit_custom_mode

Tcl command; edits a custom mode.

```
smartpower_edit_custom_mode -name {old_mode_name} new_name {new_mode_name} -description
{mode_description}
```

Arguments

`-name {old_mode_name}`

Specifies the name of the custom mode you want to edit.

`-new_name {new_mode_name}`

Specifies the new name of the custom mode.

`-description {mode_description}`

Specifies the description of the new custom mode.

Supported Families

IGLOO, ProASIC3, SmartFusion, Fusion

Exceptions

None

Examples

This example edits custom mode "Cust_1" and renames it "Cust_2":

```
smartpower_edit_custom_mode -name {Cust_1} -new_name {Cust_2} -description {frequency 10
MHz}
```

See Also

Tcl documentation conventions

Designer Tcl Command Reference

`smartpower_remove_custom_mode`

`smartpower_add_custom_mode`

smartpower_edit_scenario

Tcl command; edits a scenario.

```
smartpower_edit_scenario -name {value} -description {value} -mode {value} -new_name {value}
```

Arguments

`-name {value}`

Specifies the name of the scenario.

`-description {value}`

Specifies the description of the scenario.

`-mode {<operating mode>:<duration>}`

Specifies the mode(s) and duration(s) for the specified scenario.

`-new_name {value}`

Specifies the new name for the scenario

Supported Families

IGLOO, ProASIC3, SmartFusion, Fusion

Exceptions

None

Examples

This example edits the name of myscenario to finalscenario:

```
smartpower_edit_scenario -name myscenario -new_name finalscenario
```

See Also

Tcl documentation conventions

Designer Tcl Command Reference

smartpower_init_do

Tcl command; initializes the frequencies and probabilities for clocks, registers, set/reset nets, primary inputs, combinational outputs, enables and other sets of pins, and selects a mode for initialization.

```
smartpower_init_do -with {value} -opmode {value} -clocks {value} -registers {value} -
set_reset {value} -primaryinputs {value} -combinational {value} -enables {value} -othersets
{value}
```

Arguments

`-with{value}`

This sets the option of initializing frequencies and probabilities with vectorless analysis or with fixed values. The following table shows the acceptable values for this argument:

Value	Description
vectorless	Initializes frequencies and probabilities with vectorless analysis
fixed	Initializes frequencies and probabilities with fixed values

`-opmode {value}`

Specifies the mode in which to initialize frequencies and probabilities. The mode needs to be based on an Active mode.

`-clocks {value}`

This sets the option of initializing frequencies and probabilities for all clocks. The following table shows the acceptable values for this argument:

Value	Description
true	Initializes frequencies and probabilities for all clocks
false	Does not initialize frequencies and probabilities for all clocks

`-registers {value}`

This sets the option of initializing frequencies and probabilities for all registers. The following table shows the acceptable values for this argument:

Value	Description
true	Initializes frequencies and probabilities for all registers
false	Does not initialize frequencies and probabilities for all registers

`-set_reset {value}`

This sets the option of initializing frequencies and probabilities for all set/reset nets. The following table shows the acceptable values for this argument:

Value	Description
true	Initializes frequencies and probabilities for all set/reset nets
false	Does not initialize frequencies and probabilities for all set/reset nets

`-primaryinputs{value}`

This sets the option of initializing frequencies and probabilities for all primary inputs. The following table shows the acceptable values for this argument:

Value	Description
true	Initializes frequencies and probabilities for all primary inputs
false	Does not initialize frequencies and probabilities for all primary inputs

`-combinational {value}`

This sets the option of initializing frequencies and probabilities for all combinational outputs. The following table shows the acceptable values for this argument:

Value	Description
true	Initializes frequencies and probabilities for all combinational outputs
false	Does not initialize frequencies and probabilities for all combinational outputs

`-enables {value}`

This sets the option of initializing frequencies and probabilities for all enable sets of pins. The following table shows the acceptable values for this argument:

Value	Description
true	Initializes frequencies and probabilities for all enable sets of pins
false	Does not initialize frequencies and probabilities for all enable sets of pins

```
-othersets {value}
```

This sets the option of initializing frequencies and probabilities for all other sets of pins. The following table shows the acceptable values for this argument:

Value	Description
true	Initializes frequencies and probabilities for all other sets of pins
false	Does not initialize frequencies and probabilities for all other sets of pins

Supported Families

IGLOO, ProASIC3, SmartFusion, Fusion

Notes

This command is associated with the functionality of Initialize frequencies and probabilities dialog box.

Exceptions

None

Examples

The following example initializes all clocks with:

```
smartpower_init_do -with {vectorless} -opmode {my_mode} -clocks {true} -registers {true}
-asynchronous {true} -primaryinputs {true} -combinational {true} -enables {true} -
othersets {true}
```

See Also

Tcl documentation conventions

Designer Tcl Command Reference

smartpower_init_set_clocks_options

Tcl command; initializes the clock frequency options of all clock domains.

```
smartpower_init_set_clocks_options -with_clock_constraints {value} -
with_default_values {value} -freq {value} -duty_cycle {value}
```

Arguments

```
-with_clock_constraints {value}
```

This sets the option of initializing the clock frequencies with frequency constraints from SmartTime. The following table shows the acceptable values for this argument:

Value	Description
true	Sets initialize clock frequencies with clock constraints ON
false	Sets initialize clock frequencies with clock constraints OFF

`-with_default_values {value}`

This sets the option of initializing the clock frequencies with a user input default value. The following table shows the acceptable values for this argument:

Value	Description
true	Sets initialize clock frequencies with default values ON
false	Sets initialize clock frequencies with default values OFF

`-freq {value}`

Specifies the user input frequency in Hz, KHz, or MHz.

`-duty_cycle {value}`

Specifies the user input duty cycles in %.

Supported Families

IGLOO, ProASIC3, SmartFusion, Fusion

Notes

This command is associated with the functionality of Initialize frequencies and probabilities dialog box.

Exceptions

None

Examples

The following example initializes all clocks after executing `smartpower_init_do` with `-clocks {true}`:

```
smartpower_init_set_clocks_options -with_clock_constraints {true} -with_default_values
{true} -freq {10 MHz} -duty_cycle {20}
```

See Also

Tcl documentation conventions

Designer Tcl Command Reference

smartpower_init_set_combinational_options

Tcl commands; initializes the frequency and probability of all combinational outputs.

```
smartpower_init_set_combinational_options -freq {value} -proba {value}
```

Arguments

`-freq {value}`

Specifies the user input frequency (in Hz, KHz, or MHz) or the toggle rate (in %). If the unit is not provided and toggle rate is active, the value is handled as a toggle rate; if toggle rate is not active, the value is handled as a frequency.

`-proba {value}`

Specifies the user input probability in %.

Supported Families

IGLOO, ProASIC3, SmartFusion, Fusion

Notes

This command is associated with the functionality of Initialize frequencies and probabilities dialog box.

Exceptions

None

Examples

The following example initializes all combinational signals after executing `smartpower_init_do` with `-combinational {true}`:

```
smartpower_init_set_combinational_options -freq {10 MHz} -proba {20}
```

See Also

Tcl documentation conventions

Designer Tcl Command Reference

smartpower_init_setofpins_values

Tcl command; initializes the frequency and probability of all sets of pins.

```
smartpower_init_setofpins_values -domain_name {name} -freq {value} -proba {value}
```

Arguments

`-domain_name{name}`

Specifies the set of pins that will be initialized. The following table shows the acceptable values for this argument:

Value	Description
IOsEnableSet	Specifies that the IOsEnableSet set of pins will be initialized
MemoriesEnableSet	Specifies that the MemoriesEnableSet set of pins will be initialized

`-freq {value}`

Specifies the user input frequency in Hz, MHz, or KHz.

`-proba {value}`

Specifies the user input probability in %.

Supported Families

IGLOO, ProASIC3, SmartFusion, Fusion

Notes

This command is associated with the functionality of Initialize frequencies and probabilities dialog box.

Exceptions

None

Examples

The following example initializes all primary inputs after executing `smartpower_init_do` with `-othersets {true}`:

```
smartpower_init_setofpins_values -domain_name {IOsEnableSet} -freq {10 MHz} -proba {20}
```

See Also

Tcl documentation conventions

Designer Tcl Command Reference

smartpower_init_set_enables_options

Tcl command; initializes the clock frequency of all enable clocks with the initialization options.

```
smartpower_init_set_enables_options -freq {value} -proba {value}
```

Arguments

`-freq {value}`

Specifies the user input frequency (in Hz, KHz, or MHz).

`-proba {value}`

Specifies the user input probability in %.

Supported Families

IGLOO, ProASIC3, SmartFusion, Fusion

Notes

This command is associated with the functionality of Initialize frequencies and probabilities dialog box.

Exceptions

None

Examples

The following example initializes all clocks after executing `smartpower_init_do` with `-enables {true}`:

```
smartpower_init_set_enables_options -freq {10 MHz} -proba {20}
```

See Also

Tcl documentation conventions

Designer Tcl Command Reference

smartpower_init_set_othersets_options

Tcl command; initializes the frequency and probability of all other sets.

```
smartpower_init_set_othersets_options [-freq "decimal value [ unit { Hz | KHz | MHz } ]"]
[-proba "decimal value"]
[-with "vectorless | default"]
[-input_freq "decimal value [ unit { Hz | KHz | MHz } ]"]
[-input_proba "decimal value"]
```

Arguments

`-freq "decimal value [unit {Hz | KHz| MHz}"`

Specifies the default frequency and units.

`-proba {decimal value}`

Specifies the default probability.

`-with "vectorless | default"`

Specifies vectorless or default analysis.

`-input_freq "decimal value [unit {Hz | KHz| MHz}"`

Specifies the input frequency and units.

`-input_proba {decimal value}`

Specifies the input probability.

Supported Families

IGLOO, ProASIC3, SmartFusion, Fusion

Notes

This command is associated with the functionality of Initialize Frequencies and Probabilities dialog box.

Exceptions

None

Examples

The following example initializes all other sets after executing `smartpower_init_do` with `-othersets {true}`:

```
smartpower_init_set_othersets_options -freq {10 MHz} -proba {20} [-with default]
```

See Also

Tcl documentation conventions

Designer Tcl Command Reference

smartpower_init_set_primaryinputs_options

Tcl command; initializes the frequency and probability of all primary inputs.

```
smartpower_init_set_primaryinputs_options -freq {value} -proba {value}
```

Arguments

`-freq {value}`

Specifies the user input frequency (in Hz, KHz, or MHz) or the toggle rate (in %). If the unit is not provided and toggle rate is active, the value is handled as a toggle rate; if toggle rate is not active, the value is handled as a frequency.

`-proba {value}`

Specifies the user input probability in %.

Supported Families

IGLOO, ProASIC3, SmartFusion, Fusion

Notes

This command is associated with the functionality of Initialize frequencies and probabilities dialog box.

Exceptions

None

Examples

The following example initializes all primary inputs after executing `smartpower_init_do` with `-primaryinputs {true}`:

```
smartpower_init_set_primaryinputs_options -freq {10 MHz} -proba {20}
```

See Also

Tcl documentation conventions

Designer Tcl Command Reference

smartpower_init_set_registers_options

Tcl command; initializes the frequency and probability of all register outputs.

```
smartpower_init_set_registers_options -freq {value} -proba {value}
```

Arguments

`-freq {value}`

Specifies the user input frequency (in Hz, KHz, or MHz) or the toggle rate (in %). If the unit is not provided and toggle rate is active, the value is handled as a toggle rate; if toggle rate is not active, the value is handled as a frequency.

`-proba {value}`

Specifies the user input probability in %.

Supported Families

IGLOO, ProASIC3, SmartFusion, Fusion

Notes

This command is associated with the functionality of Initialize frequencies and probabilities dialog box.

Exceptions

None

Examples

The following example initializes all register outputs after executing `smartpower_init_do` with `-registers {true}`:

```
smartpower_init_set_registers_options -freq {10 MHz} -proba {20}
```

See Also

Tcl documentation conventions

Designer Tcl Command Reference

smartpower_init_set_set_reset_options

Tcl command; initializes the frequency and probability of all set and reset nets.

```
smartpower_init_set_set_reset_options -freq {value} -proba {value}
```

Arguments

`-freq {value}`

Specifies the user input frequency (in Hz, KHz, or MHz) or the toggle rate (in %). If the unit is not provided and toggle rate is active, the value is handled as a toggle rate; if toggle rate is not active, the value is handled as a frequency.

`-proba {value}`

Specifies the user input probability in %.

Supported Families

IGLOO, ProASIC3, SmartFusion, Fusion

Notes

This command is associated with the functionality of Initialize frequencies and probabilities dialog box.

Exceptions

None

Examples

The following example initializes all set/reset nets after executing `smartpower_init_do` with `-set_reset {true}`:

```
smartpower_init_set_set_reset_options -freq {10 MHz} -proba {20}
```

See Also

Tcl documentation conventions

Designer Tcl Command Reference

smartpower_remove_all_annotations

Tcl command; removes all initialization annotations for the specified mode.

```
smartpower_remove_all_annotations -opmode {value}
```

Arguments

`-opmode {value}`

Removes all initialization annotations for the specified mode.

Supported Families

IGLOO, ProASIC3, SmartFusion, Fusion

Notes

This command is associated with the functionality of Initialize frequencies and probabilities dialog box.

Exceptions

None

Examples

The following example initializes all clocks withs:

```
smartpower_remove_all_annotations -opmode {my_mode}
```

See Also

Tcl documentation conventions

Designer Tcl Command Reference

smartpower_remove_custom_mode

Tcl command; removes a custom mode.

```
smartpower_remove_custom_mode -name {deleted_mode_name}
```

Arguments

```
-name {deleted_mode_name}
```

Specifies the name of the custom mode you want to delete.

Supported Families

IGLOO, ProASIC3, SmartFusion, Fusion

Exceptions

None

Examples

This example delets custom mode "Cust_1":

```
smartpower_delete_custom_mode -name {Cust_1}
```

See Also

Tcl documentation conventions

Designer Tcl Command Reference

`sp_add_custom_mode`

`sp_edit_custom_mode`

smartpower_remove_domain

Tcl command; removes an existing clock or set domain.

```
smartpower_remove_domain -domain_type {value} -domain_name {domain_name}
```

Arguments

```
-domain_type {value}
```

This specifies the type of domain to remove.The following table shows the acceptable values for this argument:

Value	Description

Value	Description
clock	The domain is a clock domain
set	The domain is a set domain

`-domain_name {domain_name}`

This specifies the name of the domain to remove

Supported Families

IGLOO, ProASIC3, SmartFusion, Fusion

Notes

The domain name must be the name of an existing domain.

The domain type must be either clock or set.

Exceptions

None

Examples

The following example removes the clock domain named "clk2":

```
smartpower_remove_domain -domain_type {clock} -domain_name {clk2}
```

The following example removes the set domain named "myset":

```
smartpower_remove_domain -domain_type {set} -domain_name {myset}
```

See Also

Tcl documentation conventions

Designer Tcl Command Reference

`smartpower_create_domain`

smartpower_remove_pin_enable_rate

This command was obsoleted in SmartPower v8.5. Update your script to use

`smartpower_remove_pin_probability` to remove the pin probability.

Note: Note: The information below is obsolete and should only be used as reference when executing previously-created scripts. Update your scripts to use `smartpower_remove_pin_probability`.

Removes the probability value associated with a specific pin. This pin will have a default probability based on the domain set it belongs to.

```
smartpower_remove_pin_enable_rate –pin_name {pin_name}
```

Arguments

`-pin_name {pin_name}`

Specifies the name of the pin with the probability to remove. This pin must be the direct driver of an enable pin.

Supported Families

IGLOO, ProASIC3, SmartFusion, Fusion

Exceptions

None

Examples

The following example removes the probability of the pin driving the enable pin of a bidirectional I/O:

```
Smartpower_remove_pin_enable_rate –pin_name mybibuf/U0/U1:EOUT
```

smartpower_remove_pin_frequency

Tcl command; removes the frequency associated with a specific pin. This pin will have a default frequency based on its domain.

```
smartpower_remove_pin_frequency -pin_name {pin_name}
```

Arguments

```
-pin_name {pin_name}
```

Specifies the name of the pin for which the frequency will be removed.

Supported Families

IGLOO, ProASIC3, SmartFusion, Fusion

Notes

The *pin_name* must be the name of a pin that already exists in the design and already belongs to a domain.

Exceptions

None

Examples

The following example removes the frequency from the pin named "count8_clock":

```
smartpower_remove_pin_frequency -pin_name {count8_clock}
```

See Also

Tcl documentation conventions

Designer Tcl Command Reference

`smartpower_set_pin_frequency`

smartpower_remove_pin_of_domain

Tcl command; removes a clock pin or a data pin from a clock or set domain, respectively.

```
smartpower_remove_pin_of_domain -pin_name {pin_name} -pin_type {value} -domain_name
{domain_name} -domain_type {value}
```

Arguments

```
-pin_name {pin_name}
```

Specifies the name of the pin to remove from the domain.

```
-pin_type {value}
```

Specifies the type of the pin to remove. The following table shows the acceptable values for this argument:

Value	Description
clock	The pin to remove is a clock pin
data	The pinto remove is a data pin

`-domain_name {`*`domain_name`*`}`

Specifies the name of the domain from which to remove the pin.

`-domain_type {`*`value`*`}`

Specifies the type of domain from which the pin is being removed. The following table shows the acceptable values for this argument:

Value	Description
clock	The domain is a clock domain
set	The domain is a set domain

Supported Families

IGLOO, ProASIC3, SmartFusion, Fusion

Notes

The domain name must be the name of an existing domain.

The pin name must be the name of an existing pin.

Exceptions

None

Examples

The following example removes the clock pin named "XCMP3/UO/U1:Y" from the clock domain named "clockh":

```
smartpower_remove_pin_of_domain -pin_name {XCMP3/U0/U1:Y}
-pin_type {clock} -domain_name {clockh} -domain_type {clock}
```

The following example removes the data pin named "count2_en" from the set domain named "InputSet":

```
smartpower_remove_pin_of_domain -pin_name {count2_en} -pin_type
{data} -domain_name {InputSet} -domain_type {set}
```

See Also

Tcl documentation conventions

Designer Tcl Command Reference

`smartpower_add_pin_in_domain`

smartpower_remove_pin_probability

Tcl command; removes the probability value associated with a specific pin. This pin will have a default probability based on the domain set it belongs to.

```
smartpower_remove_pin_probability –pin_name {pin_name}
```

Arguments

```
-pin_name {pin_name}
```

Specifies the name of the pin with the probability to remove. This pin must be the direct driver of an enable pin.

Supported Families

IGLOO, ProASIC3, SmartFusion, Fusion

Exceptions

None

Examples

The following example removes the probability of the pin driving the enable pin of a bidirectional I/O:

```
Smartpower_remove_pin_probability –pin_name mybibuf/U0/U1:EOUT
```

See Also

Tcl documentation conventions

Designer Tcl Command Reference

smartpower_set_pin_probability

smartpower_remove_scenario

Tcl command; removes a scenario from the current design.

```
smartpower_remove_scenario -name {value}
```

Arguments

```
-name {value}
```

Specifies the name of the scenario.

Supported Families

IGLOO, ProASIC3, SmartFusion, Fusion

Exceptions

None

Examples

This example removes a scenario from the current design:

```
smartpower_remove_scenario -name myscenario
```

See Also

Tcl documentation conventions

Designer Tcl Command Reference

smartpower_remove_vcd

Tcl command; removes an existing VCD file from a mode or entire design.

```
smartpower_remove_vcd -from {value} -mode {value} -file
```

Arguments

`-from {value}`

This specifies the if the VCD is removed for a specific mode or for the entire project. The following table shows the acceptable values for this argument:

Value	Description
mode	The VCD file is removed for a mode
project	The VCD file is removed from the project

`-mode {value}`

This specifies the name of the mode for which the VCD will be removed

`-filename`

This specifies the name of the VCD file to be removed

Supported Families

IGLOO, ProASIC3, SmartFusion, Fusion

Exceptions

None

Examples

The following example removes the VCD file named *my_vcd.vcd* from the active mode

```
smartpower_remove_vcd -from {mode} -mode {active} -my_vcd.vcd
```

See Also

Tcl documentation conventions

Designer Tcl Command Reference

`smartpower_create_domain`

smartpower_restore

Tcl command; restores all power information previously committed in SmartPower.

```
smartpower_restore
```

Arguments

None

Supported Families

IGLOO, ProASIC3, SmartFusion, Fusion

Exceptions

None

Examples

```
smartpower_restore
```

See Also

Tcl documentation conventions

Designer Tcl Command Reference

`smartpower_commit`

smartpower_set_battery_capacity

Tcl command; sets the battery capacity.

```
smartpower_set_battery_capacity {value}
```

Arguments

value

Sets the battery capacity to a value in mA/h.

Supported Families

IGLOO, ProASIC3, SmartFusion, Fusion

Exceptions

None

Examples

The following example sets the battery capacity to 40 A/h:

```
smartpower_set_battery_capacity {40}
```

See Also

Tcl documentation conventions

Designer Tcl Command Reference

smartpower_set_cooling

Tcl command; sets the cooling style to one of the predefined types, or a custom value.

```
smartpower_set_cooling -style {value} -teta {value}
```

Arguments

`-style {value}`

Specifies the cooling style to custom value or to one of the predefined types with a default thermal resistance value. The following table shows the acceptable values for this argument:

Value	Description
300_lfm	Predefined cooling style
case_cooling	Predefined cooling style
still_air	Predefined cooling style
custom	Cooling style defined by user input

`-teta {value}`

Specifies the thermal resistance in °C/W. This argument is only available when style is set to Custom.

Supported Families

IGLOO, ProASIC3, SmartFusion, Fusion

Notes

To compute the junction temperature, set the following three commands: smartpower_set_thermalmode, smartpower_set_tambient and smartpower_set_cooling. The junction temperature will be updated when an output command is executed, such as report(Power).

Exceptions

None

Examples

The following example sets the cooling style to still air:

```
smartpower_set_cooling -style {still_air}
```

See Also

Tcl documentation conventions

Designer Tcl Command Reference

smartpower_set_mode_for_analysis

Tcl command; sets the mode for cycle-accurate power analysis.

```
smartpower_set_mode_for_analysis -mode {value}
```

Arguments

```
-mode {value}
```

Specifies the mode for cycle-accurate power analysis.

Supported Families

IGLOO, ProASIC3, SmartFusion, Fusion

Exceptions

None

Examples

The following example sets the mode for analysis to active:

```
smartpower_set_mode_for_analysis -mode {active}
```

See Also

Tcl documentation conventions

Designer Tcl Command Reference

smartpower_set_operating_condition

Tcl command; sets the operating conditions used in SmartPower to one of the pre-defined types.

```
smartpower_set_operating_condition -opcond {value}
```

Arguments

`-opcond {value}`

Specifies the value of the operating condition. The following table shows the acceptable values for this argument:

Value	Description
best	Sets the operating conditions to best
typical	Sets the operating conditions to typical
worst	Sets the operating conditions to worst

Supported Families

IGLOO, ProASIC3, SmartFusion, Fusion

Exceptions

None

Examples

This example sets the operating conditions to best:

```
smartpower_set_operating_condition -opcond {best}
```

See Also

Tcl documentation conventions

Designer Tcl Command Reference

smartpower_set_pin_enable_rate

This command was obsoleted in SmartPower v8.5. Update your script to use `smartpower_set_pin_probability` to set the pin probability.

Note: Note: The information below is obsolete and should only be used as reference when executing previously-created scripts. Update your scripts to use smartpower_set_pin_probability.

Enables you to set the probability value of a pin driving an enable pin. For I/Os, if you do not use this command, the probability of the IOEnableSet is used. For memories, if you do not use this command, the probability of the MemoriesEnableSet is used.

```
smartpower_set_pin_enable_rate -pin_name {pin_name} –pin_enable_rate {value}
```

Arguments

`-pin_name {pin_name}`

Specifies the name of a pin for which the probability will be set. This pin must be the direct driver of an enable pin.

`-pin_enable_rate {value}`

Specifies the value of the pin probability as a percentage, which can be any positive decimal between 0 and 100, inclusive.

Supported Families

IGLOO, ProASIC3, SmartFusion, Fusion

Exceptions

None

Examples

The following example sets the probability of the pin driving the enable pin of a bidirectional I/O

```
smartpower_set_pin_enable_rate -pin_name mybibuf/U0/U1:EOUT   \
–pin_enable_rate 50.4
```

smartpower_set_pin_frequency

Tcl command; sets the frequency of a pin in megahertz (MHz). If you do not use this command, each pin will have default frequency based on its domain.

```
smartpower_set_pin_frequency -pin_name {pin_name} -pin_freq {value}
```

Arguments

`-pin_name {pin_name}`

Specifies the name of the pin for which the frequency will be set.

`-pin_freq {value}`

Specifies the value of the frequency in MHz, which can be any positive decimal number.

Supported Families

IGLOO, ProASIC3, SmartFusion, Fusion

Notes

The *pin_name* must be the name of a pin that already exists in the design and already belongs to a domain.

When specifying the unit, a space must be between the frequency value and the unit.

Exceptions

None

Examples

This example sets the frequency of the pin named "count8_clock" to 100 MHz:

```
smartpower_set_pin_frequency -pin_name {count8_clock} -pin_freq {100}
```

See Also

Tcl documentation conventions

Designer Tcl Command Reference

`smartpower_remove_pin_frequency`

smartpower_set_preferences

Tcl command; sets the following preferences: power unit, frequency unit, operating mode, operating conditions, and toggle. These preferences can also be set from the preferences dialog box.

```
smartpower_set_preferences -powerunit {value} -frequnit {value} -opmode {value} -opcond
{value} -toggle {value}
```

Arguments

`-powerunit {value}`

Specifies the unit in which power is set. The following table shows the acceptable values for this argument:

Value	Description
W	The power unit is set to watts
mW	The power unit is set to milliwatts
uW	The power unit is set to microwatts

`-frequnit {value}`

Specifies the unit in which frequency is set. The following table shows the acceptable values for this argument:

Value	Description
Hz	The frequency unit is set to hertz
kHz	The frequency unit is set to kilohertz
MHz	The frequency unit is set to megahertz

`-opmode {value}`

Specifies the operating mode. The following table shows the acceptable values for this argument:

Value	Description
active	The operating mode is set to active
static	The operating mode is set to static
sleep	The operating mode is set to sleep
Flash*Freeze	The operating mode is set to Flash*Freeze
shutdown	The operating mode is set to shutdown

`-opcond {value}`

Specifies the operating condition. The following table shows the acceptable values for this argument:

Value	Description
worst	The operating condition is set to worst case
typical	The operating condition is set to typical case
best	The operating condition is set to best case

`-toggle {value}`

Specifies the toggle. The following table shows the acceptable values for this argument:

Value	Description
true	The toggle is set to true
false	The toggle is set to false

Supported Families

IGLOO, ProASIC3, SmartFusion, Fusion

Notes

- The following arguments have been removed. Running the script will trigger a warning message: Warning: Invalid argument: -argname "argvalue" Ignored. Ignore the warning.

-maxblocks {integer > 0}

-maxpins [{integer > 0}

-sortorder {ascending, descending}

-sortby {powervalues, alphabetical}

- Flash*Freeze, Sleep, and Shutdown are available only for certain families and devices.
- Worst and Best operating conditions are available only for certain families and devices.

Exceptions

None

Examples

This example sets the frequency of the power unit to "watts", the frequency unit to "Hz", the operating mode to "active", the operating condition to "typical", and the toggle to "true":

```
smartpower_set_setpreferences -powerunit {w} -frequnit {hz} -opmode {active} -opcond
{typical} -toggle {true}
```

See Also

Tcl documentation conventions

Designer Tcl Command Reference

SmartPower Preferences

smartpower_set_scenario_for_analysis

Tcl command; sets the scenario for cycle-accurate power analysis.

```
smartpower_set_scenario_for_analysis -scenario{value}
```

Arguments

`-scenario {value}`

Specifies the mode for cycle-accurate power analysis.

Supported Families

IGLOO, ProASIC3, SmartFusion, Fusion

Exceptions

None

Examples

The following example sets the scenario for analysis to my_scenario:

```
smartpower_set_scenario_for_analysis -scenario {my_scenario}
```

See Also

Tcl documentation conventions

Designer Tcl Command Reference

smartpower_set_temperature_opcond

Tcl command; sets the temperature in the operating conditions to one of the pre-defined types.

```
smartpower_set_temperature_opcond -use{value}
```

Arguments

```
-use{value}
```

Specifies the temperature in the operating conditions. The following table shows the acceptable values for this argument:

Value	Description
oprange	Sets the temperature in the operating conditions as specified in your Project Settings.
design	Sets the temperature in the operating conditions as specified in the SmartPower design-wide operating range. Applies to SmartPower only.
mode	Sets the temperature in the operating conditions as specified in the SmartPower mode-specific operating range. Applies to SmartPower only.

Supported Families

IGLOO, ProASIC3, SmartFusion, Fusion

Exceptions

None

Examples

This example sets the temperature in the operating conditions as specified in the custom mode-settings:

```
smartpower_set_temperature_opcond -use{mode}
```

See Also

Tcl documentation conventions

Designer Tcl Command Reference

smartpower_set_thermalmode

Tcl command; sets the mode of computing junction temperature.

```
smartpower_set_thermalmode -mode {value}
```

Arguments

```
-mode {value}
```

Specifies the mode in which the junction temperature is computed. The following table shows the acceptable values for this argument:

Value	Description
ambient	The junction temperature will be iteratively computed with total static power
opcond	The junction temperature will be given as one of the operating condition range values specified in the device selection

Supported Families

IGLOO, ProASIC3, SmartFusion, Fusion

Notes

To compute the junction temperature, set the following three commands: smartpower_set_thermalmode, smartpower_set_tambient and smartpower_set_cooling. The junction temperature will be updated when an output command is executed, such as report(Power).

Exceptions

None

Examples

The following example sets the computing of the junction temperature to ambient mode:

```
smartpower_set_thermalmode -mode {ambient}
```

See Also

Tcl documentation conventions

Designer Tcl Command Reference

smartpower_set_voltage_opcond

Tcl command; sets the voltage in the operating conditions.

```
smartpower_set_voltage_opcond -voltage{value} -use{value}
```

Arguments

```
-voltage{value}
```

Specifies the voltage supply in the operating conditions. The following table shows the acceptable values for this argument:

Value	Description
VCCA	Sets the voltage operating conditions for VCCA
VCCI 3.3	Sets the voltage operating conditions for VCCI 3.3
VCCI 2.5	Sets the voltage operating conditions for VCCI 2.5
VCCI 1.8	Sets the voltage operating conditions for VCCI 1.8
VCCI 1.5	Sets the voltage operating conditions for VCCI 1.5
VCC33A	Sets the voltage operating conditions for VCC33A
VCCDA	Sets the voltage operating conditions for VCCDA

```
-use{value}
```

Specifies the voltage in the operating conditions for each voltage supply. The following table shows the acceptable values for this argument:

Value	Description
oprange	Sets the voltage in the operating conditions as specified in your Project Settings.
design	Sets the voltage in the operating conditions as specified in the SmartPower design-wide operating range. Applies to SmartPower only.
mode	Sets the voltage in the operating conditions as specified in the SmartPower mode-specific operating range. Applies to SmartPower only.

Supported Families

IGLOO, ProASIC3, SmartFusion, Fusion

Exceptions

None

Examples

This example sets the VCCA as specified in the SmartPower mode-specific settings:

```
smartpower_set_voltage_opcond -voltage{vcca} -use{mode}
```

See Also

Tcl documentation conventions

Designer Tcl Command Reference

smartpower_temperature_opcond_set_design_wide

Tcl command; sets the temperature for SmartPower design-wide operating conditions.

```
smartpower_temperature_opcond_set_design_wide -best{value} -typical{value} -worst{value} -
thermal_mode{value}
```

Arguments

`-best{value}`

Specifies the best temperature (in degrees Celsius) used for design-wide operating conditions.

`-typical{value}`

Specifies the typical temperature (in degrees Celsius) used for design-wide operating conditions.

`-worst{value}`

Specifies the worst temperature (in degrees Celsius) used for design-wide operating conditions.

`-thermal_mode{value}`

Specifies the mode in which the junction temperature is computed. The following table shows the acceptable values for this argument:

Value	Description
ambient	The junction temperature will be iteratively computed with total static power
opcond	The junction temperature will be given as one of the operating condition range values specified in the device selection

Supported Families

IGLOO, ProASIC3, SmartFusion, Fusion

Exceptions

None

Examples

This example sets the temperature for design-wide operating conditions to Best 20, Typical 30, and Worst 60:

```
smartpower_temperature_opcond_set_design_wide -best{20} -typical{30} -worst{60}
```

See Also

Tcl documentation conventions

Designer Tcl Command Reference

smartpower_temperature_opcond_set_mode_specific

Tcl command; sets the temperature for SmartPower mode-specific operating conditions.

```
smartpower_temperature_opcond_set_mode_specific -mode{value} -best{value} -typical{value} -
worst{value} -thermal_mode{value}
```

Arguments

`-mode{value}`

Selects the mode to which apply the operating condition settings. You can select a pre-defined mode or any custom mode in your design.

`-best{value}`

Specifies the best temperature (in degrees Celsius) for the selected mode.

`-typical{value}`

Specifies the typical temperature (in degrees Celsius) for the selected mode.

`-worst{value}`

Specifies the worst temperature (in degrees Celsius) for the selected mode.

`-thermal_mode{value}`

Specifies the mode in which the junction temperature is computed. The following table shows the acceptable values for this argument:

Value	Description
ambient	The junction temperature will be iteratively computed with total static power
opcond	The junction temperature will be given as one of the operating condition range values specified in the device selection

Supported Families

IGLOO, ProASIC3, SmartFusion, Fusion

Exceptions

None

Examples

This example sets the temperature for mode-specific operating conditions for mode1:

```
smartpower_temperature_opcond_set_mode_specific -mode{mode1} -best{20} -typical{30} -
worst{60}
```

See Also

Tcl documentation conventions

Designer Tcl Command Reference

smartpower_voltage_opcond_set_design_wide

Tcl command; sets the voltage settings for SmartPower design-wide operating conditions.

```
smartpower_voltage_opcond_set_design_wide -voltage{value} -best{value} -typical{value} -
worst{value}
```

Arguments

`-voltage{value}`

Specifies the voltage supply in the operating conditions. The following table shows the acceptable values for this argument:

Value	Description
VCCA	Sets the voltage operating conditions for VCCA

Value	Description
VCCI 3.3	Sets the voltage operating conditions for VCCI 3.3
VCCI 2.5	Sets the voltage operating conditions for VCCI 2.5
VCCI 1.8	Sets the voltage operating conditions for VCCI 1.8
VCCI 1.5	Sets the voltage operating conditions for VCCI 1.5
VCC33A	Sets the voltage operating conditions for VCC33A
VCCDA	Sets the voltage operating conditions for VCCDA

```
-best{value}
```

Specifies the best voltage used for design-wide operating conditions.

```
-typical{value}
```

Specifies the typical voltage used for design-wide operating conditions.

```
-worst{value}
```

Specifies the worst voltage used for design-wide operating conditions.

Supported Families

IGLOO, ProASIC3, SmartFusion, Fusion

Exceptions

None

Examples

This example sets VCCA for design-wide to best 20, typical 30 and worst 40:

```
smartpower_voltage_opcond_set_design_wide -voltage{VCCA} -best{20} -typical{30} -
worst{40}
```

See Also

Tcl documentation conventions

Designer Tcl Command Reference

smartpower_voltage_opcond_set_mode_specific

Tcl command; sets the voltage settings for SmartPower mode-specific use operating conditions.

```
smartpower_voltage_opcond_set_mode_specific -opmode{value} -voltage{value} -best{value} -
typical{value} -worst{value}
```

Arguments

```
-opmode{value}
```

Selects the mode to which apply the operating condition settings. You can select a pre-defined mode or any custom mode in your design.

```
-voltage{value}
```

Specifies the voltage in the operating conditions. The following table shows the acceptable values for this argument:

Value	Description
VCCA	Sets the voltage operating conditions for VCCA
VCCI 3.3	Sets the voltage operating conditions for VCCI 3.3
VCCI 2.5	Sets the voltage operating conditions for VCCI 2.5
VCCI 1.8	Sets the voltage operating conditions for VCCI 1.8
VCCI 1.5	Sets the voltage operating conditions for VCCI 1.5
VCC33A	Sets the voltage operating conditions for VCC33A
VCCDA	Sets the voltage operating conditions for VCCDA

`-best{`*`value`*`}`

Specifies the best voltage used for mode-specific operating conditions.

`-typical{`*`value`*`}`

Specifies the typical voltage used for mode-specific operating conditions.

`-worst{`*`value`*`}`

Specifies the worst voltage used for mode-specific operating conditions.

Supported Families

IGLOO, ProASIC3, SmartFusion, Fusion

Exceptions

None

Examples

This example sets the voltage for the static mode and sets best to 20, typical to 30 and worst to 40:

```
smartpower_voltage_opcond_set_mode_specific -opmode{active} -voltage{VCCA} -best{20} -
typical{30} -worst{40}
```

See Also

Tcl documentation conventions

Designer Tcl Command Reference

st_commit

Tcl command; saves the changes made in SmartTime to the design (.adb) file

```
st_commit
```

Arguments

None

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Examples

```
st_commit
```

See Also

`st_restore`

Tcl documentation conventions

Designer Tcl Command Reference

st_create_set

Tcl command; creates a set of paths to be analyzed. Use the arguments to specify which paths to include. To create a set that is a subset of a clock domain, specify it with `-clock` and `-type`. To create a set that is a subset of an inter-clock domain set, specify it with `-source_clock` and `-sink_clock`. To create a set that is a subset (filter) of an existing named set, specify the set to be filtered with `-from_set`.

To create a set that is not derived from an existing set, you must provide both the `-source` *`pin_list`* and `-sink`*`pin_list`* derived. Otherwise, the `-source` and `-sink` arguments act as filters on the pins from the parent set. You must give each new set a unique name in the design.

```
st_create_set -name name
[-parent_set name ]
[-clockclock_id -type value ]
[-in_to_out]
[-source_clock clock_id -sink_clock clock_id]
[-source pin_list ] -sink pin_list ]
```

Arguments

`-name` *`name`*

Specifies a unique name for the newly create path set.

-parent_set *`name`*

Specifies the name of the set to filter.

-clock *`clock_id`*

Specifies that the set is to be a subset of the given clock domain. This argument is valid only if you also specify the -type argument.

`-type` *`value`*

Specifies the predefined set type on which to base the new path set. You can only use this argument with the -clock argument, not by itself.

Value	Description
reg_to_reg	Paths between registers in the design
async_to_reg	Paths from asynchronous pins to registers
reg_to_async	Paths from registers to asynchronous pins
external_recovery	The set of paths from inputs to asynchronous pins
external_removal	The set of paths from inputs to asynchronous pins
external_setup	Paths from input ports to registers

Value	Description
external_hold	Paths from input ports to registers
clock_to_out	Paths from registers to output ports

`-in_to_out`

Specifies that the set is based on the “Input to Output” set, which includes paths that start at input ports and end at output ports.

`-source_clock` *`clock_id`*

Specifies that the set will be a subset of an inter-clock domain set with the given source clock.

You can only use this option with the -sink_clock option, not by itself.

`-sink_clock` *`clock_id`*

Specifies that the set will be a subset of an inter-clock domain set with the given sink clock.

You can only use this option with the -source_clock option, not by itself.

`-source` *`pin_list`*

Specifies a filter on the source pins of the parent set. If you do not specify a parent set, this option filters all pins in the current design.

`-sink` *`pin_list`*

Specifies a filter on the sink pins of the parent set. If you do not specify a parent set, this option filters all pins in the current design.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Examples

```
st_create_set –name { my_user_set } –source { C* } –sink { D* }
st_create_set –name { my_other_user_set } –from_set { my_user_set } –source { CL* }
st_create_set –name { adder } –clock { ALU_CLOCK } –type { REG_TO_REG }  -sink { ADDER*
}
st_create_set –name { another_set } –source_clock { EXTERN_CLOCK } –sink_clock {
MY_GEN_CLOCK }
st_create_set –name { some_p2p } –pin2pin –to { T* }
```

See Also

Designer Tcl Command Reference

Tcl documentation conventions

`st_remove_set`

st_edit_set

Tcl command; modifies the paths in a user set.

```
st_edit_set -name name
 [-source pin_list ] [-sink  pin_list ]
 [-rename_to name ]
```

Arguments

`-name` *`name`*

Specifies the name of the set to modify.

`-source` *`pin_list`*

If the set is a subset of another set, specifies a filter on the source pins from the parent set. Otherwise, this option specifies the source pins of the set.

`-sink` *`pin_list`*

If the set is a subset of another set, specifies a filter on the sink pins from the parent set. Otherwise, this option specifies the sink pins of the set.

`-rename_to` *`name`*

Specifies a new name for the set.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Examples

```
st_edit_set –name { my_user_set} –rename_to { my_critical_pins }
st_edit_set –name { adder } –sink { ADD* }
```

See Also

Designer Tcl Command Reference

Tcl documentation conventions

st_create_set

`st_remove_set`

st_expand_path

Tcl command; displays expanded path information (path details) for paths. The paths to be expanded are identified by the parameters required to display these paths with st_list_paths. For example, to expand the first path listed with st_list_paths -clock {MYCLOCK} -type {register_to_register}, use the command st_expand_path -clock {MYCLOCK} -type {register_to_register}. Path details contain the pin name, type, net name, cell name, operation, delay, total delay, and edge as well as the arrival time, required time, and slack. These details are the same as details available in the SmartTime Expanded Path window.

```
st_expand_path [-set name]
[-clock clock_id -type value]
[-in_to_out]
[-source_clock clock_id -sink_clock clock_id]
[-source pin_list] [-sink pin_list]
[-analysis value]
[-index list_of_indices]
[-format value]
```

Arguments

-set *`name`*

Displays a list of paths from the named set. You can either use the -set option to specify a set name, or use both -clock and -type to specify a set. A list of valid set names includes "in_to_out", as well as any user set names.

-clock *clock_id*

Displays the set of paths belonging to the specified clock domain. You can either use this option along with -type to specify a set or use the -set option to specify the name of the set to display.

`-in_to_out`

Specifies that the paths should be from the set "Input to Output, which includes paths that start at input ports and end at output ports.

`-type` *value*

Specifies the type of paths in the clock domain to display in a list. You can only use this option with the -clock option, not by itself. You can either use this option along with -clock to specify a set or use the -set option to specify a set name.

Value	Description
reg_to_reg	Paths between registers in the design
async_to_reg	Paths from asynchronous pins to registers
reg_to_asyn	Paths from registers to asynchronous pins
external_recovery	The set of paths from inputs to asynchronous pins
external_removal	The set of paths from inputs to asynchronous pins
external_setup	Paths from input ports to registers
	Paths from input ports to registers
clock_to_out	Paths from registers to output ports

`-source_clock` *clock_id*

Displays a list of timing paths for an inter-clock domain set belonging to the source clock specified. You can only use this option with the -sink_clock option, not by itself.

`-sink_clock` *clock_id*

Displays a list of timing paths for an inter-clock domain set belonging to the sink clock specified. You can only use this option with the -source_clock option, not by itself.

`-source` *pin_list*

Specifies a filter on the source pins of the paths to be listed.

`-sink` *pin_list*

Specifies a filter on the sink pins of the paths to be listed.

`-analysis` *name*

Specifies the analysis type for the paths to be listed. The following table shows the acceptable values for this argument:

Value	Description
maxdelay	Maximum delay analysis
mindelay	Minimum delay analysis

`-index` *list_of_indices*

Specifies which paths to display. The index starts at 1 and defaults to 1. Only values lower than the max_paths option will be expanded.

`-format` *value*

Specifies the file format of the output. The following table shows the acceptable values for this argument:

Value	Description
text	ASCII text format
csv	Comma separated value fie format

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Examples

Note: The following example returns a list of five paths:

```
st_expand_path –clock { myclock } –type {reg_to_reg }
st_expand_path –clock {myclock} –type {reg_to_reg} –index { 1 2 3 } –format text
```

See Also

Designer Tcl Command Reference

Tcl documentation conventions

st_list_paths

st_list_paths

Tcl command; displays the list of paths in the same tabular format shown in SmartTime.

```
st_list_paths [-set name ]
[-clock clock_id -type value ]
 [-in_to_out]
 [-source_clock clock_id -sink_clock  clock_id]
 [-source pin_list ] [-sink pin_list ]
 [-analysis value ]
 [-format value ]
```

Arguments

`-set name`

Displays a list of paths from the named set. You can either use the –set option to specify a set name, or use both –clock and –type to specify a set. A list of valid set names includes “in_to_out”, as well as any user set names.

-clock `clock_id`

Displays the set of paths belonging to the specified clock domain. You can either use this option along with -type to specify a set or use the -set option to specify the name of the set to display.

`-in_to_out`

Specifies that the paths should be from the set “Input to Output”, which includes paths that start at input ports and end at output ports.

`-type value`

Specifies the type of paths in the clock domain to display in a list. You can only use this option with the -clock option, not by itself. You can either use this option along with -clock to specify a set or use the -set option to specify a set name.

Value	Description
reg_to_reg	Paths between registers in the design
async_to_reg	Paths from asynchronous pins to registers
reg_to_asyn	Paths from registers to asynchronous pins
external_recovery	The set of paths from inputs to asynchronous pins
external_removal	The set of paths from inputs to asynchronous pins
external_setup	Paths from input ports to registers
	Paths from input ports to registers
clock_to_out	Paths from registers to output ports

`-source_clock` **`clock_id`**

Displays a list of timing paths for an inter-clock domain set belonging to the source clock specified. You can only use this option with the -sink_clock option, not by itself.

`-sink_clock` **`clock_id`**

Displays a list of timing paths for an inter-clock domain set belonging to the sink clock specified. You can only use this option with the -source_clock option, not by itself.

`-source` **`pin_list`**

Specifies a filter on the source pins of the paths to be listed.

`-sink` **`pin_list`**

Specifies a filter on the sink pins of the paths to be listed.

`-analysis` *`name`*

Specifies the analysis type for the paths to be listed. The following table shows the acceptable values for this argument:

Value	Description
maxdelay	Maximum delay analysis
mindelay	Minimum delay analysis

`-format` *`value`*

Specifies the file format of the output. The following table shows the acceptable values for this argument:

Value	Description
text	ASCII text format
csv	Comma separated value fie format

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Examples

```
st_list_paths -set  { myset }
st_list_paths -analysis mindelay -clock { myclock } -type { reg_to_reg } -format csv
```

The list of paths can be written to a file with the following Tcl commands:

```
set outfile [ open “pathlisting.csv” w]
puts $outfile [ st_list_paths -format csv -set { myset} ]
close $outfile
```

See Also

Designer Tcl Command Reference

Tcl documentation conventions

st_expand_path

st_remove_set

Tcl command; deletes a user set from the design.

```
st_remove_set -name name
```

Arguments

```
-name name
```

Specifies the name of the set to delete.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Examples

```
st_remove_set { clockset1 }
```

See Also

Designer Tcl Command Reference

Tcl documentation conventions

st_create_set

st_restore

Tcl command; restores constraints previously committed in SmartTime.

```
st_restore
```

Arguments

None

Supported Families

IGLOO, ProASIC3, SmartFusion2, SmartFusion, Fusion

Exceptions

None

Examples

```
st_restore
```

See Also

st_commit

Tcl documentation conventions

Designer Tcl Command Reference

st_set_options

Tcl command; sets options for timing analysis. With no parameters given, it will display the current settings of the options. For IGLOO, ProASIC3, SmartFusion, Fusion families, these options also affect timing-driven place-and-route.

```
st_set_options [-max_opcond value ]
 [-min_opcond value]
 [-interclockdomain_analysis value]
 [-use_bibuf_loopbacks value]
 [-enable_recovery_removal_checks value]
 [-break_at_async value]
 [-filter_when_slack_below value]
 [-filter_when_slack_above value]
 [-remove_slack_filters]
 [-limit_max_paths value]
 [-expand_clock_network value]
 [-expand_parallel_paths value]
 [-analysis_scenario value]
 [-tdpr_scenario value]
 [-reset]
```

Arguments

-max_opcond *value*

Sets the operating condition to use for Maximum Delay Analysis. The following table shows the acceptable values for this argument:

Value	Description
worst	Use Worst Case conditions for Maximum Delay Analysis
typ	Use Typical conditions for Maximum Delay Analysis
best	Use Best Case conditions for Maximum Delay Analysis

-min_opcond *value*

Sets the operating condition to use for Minimum Delay Analysis. The following table shows the acceptable values for this argument:

Value	Description
best	Use Best Case conditions for Minimum Delay Analysis
typ	Use Typical conditions for Minimum Delay Analysis
worst	Use Worst Case conditions for Minimum Delay Analysis

-interclockdomain_analysis *value*

Enables or disables inter-clock domain analysis.

Value	Description
yes	Enables inter-clock domain analysis
no	Disables inter-clock domain analysis

-use_bibuf_loopbacks *value*

Enables or disables loopback in bibufs.

Value	Description
yes	Enables loopback in bibufs
no	Disables loopback in bibufs

-enable_recovery_removal_checks *value*

Enables or disables recovery and removal checks.

Value	Description
yes	Enables recovery and removal checks
no	Disables recovery and removal checks

-break_at_async *value*

Enables or disables breaking paths at asynchronous ports.

Value	Description
yes	Enables breaking paths at asynchronous ports
no	Disables breaking paths at asynchronous ports

-filter_when_slack_below *value*

Do not show paths with slack below x.

-filter_when_slack_above *value*

Do not show paths with slack above y.

-remove_slack_filters

Remove all existing slack filters.

-limit_max_paths *value*

Limit path reporting commands to a maximum of <n> paths, where n is a value of 0 or higher.

-expand_clock_network *value*

Enables or disables expanded clock network information in expanded paths.

Value	Description
yes	Enables expanded clock network information in paths
no	Disables expanded clock network information in paths

-expand_parallel_paths *value*

Expand a maximum of <n> parallel paths, where n is a value of 0 or higher. If n is 0 or 1, only one path will be expanded when viewing expanded paths.

-analysis_scenario *value*

Set the timing constraints scenario to be used for both maximum delay and minimum delay analysis. The argument must be a valid scenario name.

Note: Note: This option does not affect the timing scenario used for TDPR.

-tdpr_scenario *value*

Set the timing constraints scenario to be used by the place and route engine. The argument must be a valid scenario name.

Note: Note: This option does not affect the timing scenario used for analysis.

-reset

Reset all options to their default values, except for scenarios used for analysis and TDPR that remain unchanged.

Supported Families

IGLOO, ProASIC3, SmartFusion2, SmartFusion, Fusion

Exceptions

None

Examples

```
st_set_options –max_opcond worst \
-min_opcond best \
-interclockdomain_analysis true \
-enable_removal_recovery_checks true
st_set_options –limit_max_paths 50 –remove_slack_filters \
–filter_when_slack_above 3
```

See Also

Tcl documentation conventions

Designer Tcl Command Reference

timer_get_path

Tcl command; displays the path between the specified pins in the Log window.

```
timer_get_path -from source_pin -to destination_pin
[-exp value]\
[-sort value]\
[-order value]\
[-case value]\
[-maxpath maximum_paths]\
[-maxexpath maximum_paths_to_expand]\
[-mindelay minimum_delay]\
[-maxdelay maximum_delay]\
[-breakatclk value]\
[-breakatclr value]
```

Arguments

`-from` *`source_pin`*

Specifies the name of the source pin for the path.

`-to` *`destination_pin`*

Specifies the name of the destination pin for the path.

`-exp` *`value`*

Specifies whether to expand the path. The following table shows the acceptable values for this argument:

Value	Description
yes	Expands the path
no	Does not expand the path

`-sort` *`value`*

Specifies whether to sort the path by either the actual delay or slack value. The following table shows the acceptable values for this argument:

Value	Description
actual	Sorts the path by the actual delay value
slack	Sorts the path by the slack value

`-order` *`value`*

Specifies whether the list is based on maximum or minimum delay analysis. The following table shows the acceptable values for this argument:

Value	Description
long	The paths are listed based on the maximum delay analysis
short	The paths are listed based on the minimum delay analysis

`-case` *`value`*

Specifies whether the report will include the worst, typical, or best case timing numbers. The following table shows the acceptable values for this argument:

Value	Description
worst	Includes worst case timing numbers
typ	Includes typical case timing numbers
best	Includes best case timing numbers

`-maxpath` *maximum_paths*

Specifies the maximum number of paths to display.

`-maxexpath` *maximum_paths_to_expand*

Specifies the maximum number of paths to expand.

`-mindelay` *minimum_delay*

Specifies the path delay in the minimum delay analysis mode.

`-maxdelay` *maximum_delay*

Specifies the path delay in the maximum delay analysis mode.

`-breakatclk` *value*

Specifies whether to break the paths at the register clock pins. The following table shows the acceptable values for this argument:

Value	Description
yes	Breaks the paths at the register clock pins
no	Does not break the paths at the register clock pins

`-breakatclr` *value*

Specifies whether to break the paths at the register clear pins. The following table shows the acceptable values for this argument:

Value	Description
yes	Breaks the paths at the register clear pins
no	Does not break the paths at the register clear pins

Supported Families

SmartFusion, IGLOO, ProASIC3 and Fusion

Exceptions

None

Examples

The following example returns the paths from input port headdr_dat<31> to the input pin of register u0_headdr_data1_reg/data_out_t_31 under typical conditions.

```
timer_get_path -from "headdr_dat<31>" \
-to  "u0_headdr_data1_reg/data_out_t_31/U0:D" \
-case typ  \
-exp "yes" \
-maxpath "100" \
-maxexpapth "10"
```

The following example returns the paths from the clock pin of register gearbox_inst/bits64_out_reg<55> to the output port pma_tx_data_64bit[55]

```
timer_get_path -from "gearbox_inst/bits64_out_reg<55>/U0:CLK" \
     -to {pma_tx_data_64bit[55]} \
     -exp "yes"
```

See Also

Tcl documentation conventions

Designer Tcl Command Reference

timer_get_clock_actuals

Tcl command; displays the actual clock frequency in the Log window, when the timing analysis tool is initiated.

```
timer_get_clock_actuals -clock clock_name
```

Arguments

-clock *clock_name*

Specifies the name of the clock with the frequency (or period) to display.

Supported Families

SmartFusion, IGLOO, ProASIC3 and Fusion

Exceptions

None

Examples

This example displays the actual clock frequency of clock clk1 in the Log window:

```
timer_get_clock_actuals -clock clk1
```

See Also

timer_get_clock_constraints

Tcl documentation conventions

Designer Tcl Command Reference

timer_get_clock_constraints

Tcl command; returns the constraints (period, frequency, and duty cycle) on the specified clock.

```
timer_get_clock_constraints -clock clock_name
```

Arguments

-clock *clock_name*

Specifies the name of the clock with the constraint to display.

Supported Families

SmartFusion, IGLOO, ProASIC3 and Fusion

Exceptions

None

Examples

The following example displays the clock constraints on the clock clk in the Log window:

```
timer_get_clock_constraints -clock clk
```

See Also

`timer_get_clock_actuals`

Tcl documentation conventions

Designer Tcl Command Reference

timer_get_maxdelay

Tcl command; displays the maximum delay constraint between two pins in a path in the Log window.

```
timer_get_maxdelay -from source_pin -to destination_pin
```

Arguments

`-from source_pin`

Specifies the name of the source pin in the path.

`-to destination_pin`

Specifies the name of the destination pin in the path.

Supported Families

SmartFusion, IGLOO, ProASIC3 and Fusion

Exceptions

None

Examples

The following example displays the maximum delay constraint from the pin clk166 to the pin reg_q_a_9_/U0:CLK in the Log window:

```
timer_get_maxdelay -from {clk166} -to {reg_q_a_9_/U0:CLK}
```

See Also

`timer_set_maxdelay`

Tcl documentation conventions

Designer Tcl Command Reference

timer_get_path_constraints

Tcl command; displays the path constraints that were set as the maximum delay constraint in the timing analysis tool.

```
timer_get_path_constraints
```

Arguments

None

Supported Families

SmartFusion, IGLOO, ProASIC3 and Fusion

Description

This command lists the paths constrained by maximum delay values. The information is displayed in the Log window. If no maximum delay constraints are set, this command does not report anything.

Exceptions

None

Examples

Invoking timer_get_path_constraints displays the following paths and their delay constraints in the Log window:

```
max_delay -from [all_inputs] -to [all_outputs] = 12 ns
```

```
max_delay -from p_f_testwdata0 p_f_testwdata1 -to p_f_dacuwdata0 p_f_dacuwdata1
r_f_dacuwdata0 r_f_dacuwdata1 = 8 ns
```

See Also

Tcl documentation conventions

Designer Tcl Command Reference

timer_remove_stop

Tcl command; removes the previously entered path stop constraint on the specified pin.

```
timer_remove_stop -pin pin_name
```

Arguments

```
-pin pin_name
```

Specifies the name of the pin from which to remove the path stop constraint.

Supported Families

All

Description

If you remove a path stop constraint using the Timer GUI, and then export a script using **File > Export** > **Script files**, the resulting script will contain `timer_remove_pass -pin pin_name` instead of `timer_remove_stop -pin pin_name`.

Exceptions

- For the IGLOO, ProASIC3, SmartFusion2, SmartFusion, Fusion families, best practice is to use the following flow:
  1. Open **SmartTime** > **Set False Path Constraint dialog box**.
  2. Look for the pin name in the **Through:** list (Note: You must not have any entry selected in the **From** or **To** lists).
  3. Delete this pin.

Examples

The following example removes the path stop constraint on the clear pin reg_q_a_0_:CLR:

```
timer_remove_stop -pin {reg_q_a_0_:CLR}
```

See Also

Tcl documentation conventions

Set False Path Constraint dialog box

Designer Tcl Command Reference

timer_restore

Tcl command; restores constraints previously committed in Timer.

```
timer_restore
```

Arguments

None

Supported Families

SmartFusion, IGLOO, ProASIC3 and Fusion

Exceptions

None

Examples

```
timer_restore
```

See Also

Tcl documentation conventions

Designer Tcl Command Reference

timer_remove_all_constraints

Tcl command; removes all timing constraints in the current design.

```
timer_remove_all_constraints
```

Arguments

None

Supported Families

All

Exceptions

None

Examples

The following example removes all of the constraints from the design and then commits the changes:

```
timer_remove_all_constraints
timer_commit
```

See Also

Tcl documentation conventions

Designer Tcl Command Reference

use_file

Tcl command; specifies which file in your project to use.

```
use_file
-file value
 -module value
 -designer_view value
```

Arguments

`-filevalue`

Specifies the EDIF or ADB file you wish to use in the project. Value is the name of the file you wish use (including the full pathname).

`-module value`

Specifies the module in which you want to use the file.

`-designer_view value`

Specifies the Designer View in which you wish to use the file.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Example

Specify file1.edn in the ./project/synthesis directory, in the module named top, in the Designer View named impl1.

```
use_file –file “./project/synthesis/file1.edn” –module “top” –designer_view “Impl1”
```

See Also

use_source_file

Project Manager Tcl Command Reference

use_source_file

Tcl command; defines a module for your project.

```
use_source_file
-file value
 -module value
```

Arguments

`-file value`

Specifies the Verilog or VHDL file. Value is the name of the file you wish use (including the full pathname).

`-module value`

Specifies the module in which you want to use the file.

Supported Families

SmartFusion2, SmartFusion, IGLOO, ProASIC3, Fusion

Exceptions

None

Example

Specify file1.vhd in the ./project/hdl directory, in the module named top.

```
use_source_file –file “./project/hdl/file1.vhd” –module “top"
```

See Also

`use_file`

Project Manager Tcl Command Reference

Application Notes

Application notes are available for all Microsemi SoC devices. A full list of application notes is available at the Microsemi SoC website.

Application notes are organized by product or type. For example, you can view a full list of application notes for SmartFusion, or you can view a list of application notes on Design Entry that includes documents for all available families.

The following is a short list of popular application notes covering a range of applications and devices.

- AC333: Connecting User Logic to the SmartFusion Microcontroller Subsystem App Note (design files required - 23 MB) - Describes how to create AHB Lite or APB3 wrapper on custom logic and how to connect it to the MSS System via the Fabric Interface Controller.
- AC225 Programming Antifuse Devices App Note - Provides an overview of the programming options available for the antifuse families.
- AC362: SmartFusion cSoC: Programming FPGA Fabric and eNVM Using In-Application Programming Interface App Note (design files required - 50 MB)
- AC335: Building an APB3 Core for SmartFusion cSoC FPGAs App Note (design files required - 13 MB) - Describes how to create an APB3 wrapper interface for your logic or IP and connect it to the MSS via the Fabric Interface Controller.
- AC265: Clock Generation and Distribution Design Example App Note (design files required - 1 MB) - Demonstrates the use of the IGLOO and ProASIC3 clock conditioning circuits and phase-locked loops (PLLs) to generate multiple clock signals with different phases and frequencies.
-

Tutorials and Training Modules

Software tutorials, webcasts and online training modules are available on the Microsemi website. See the website for a full list.

The following list is an example of the tutorials available. Training modules may require you to register to enter the Microsemi Training Portal. Registration is free.

Example Tutorials

ARM Cortex M1-Embedded Processor Tutorial (design files required - 105 MB) - Describes how to create a Cortex-M1 processor system that runs on the Fusion development kit board available from Microsemi SoC.

SmartFusion cSoC Webserver Demo Using uIP and FreeRTOS - Demonstrates the SmartFusion device capabilities using the SmartFusion Development Kit Board. Requires the following design files and the SmartFusion Development Kit Board.

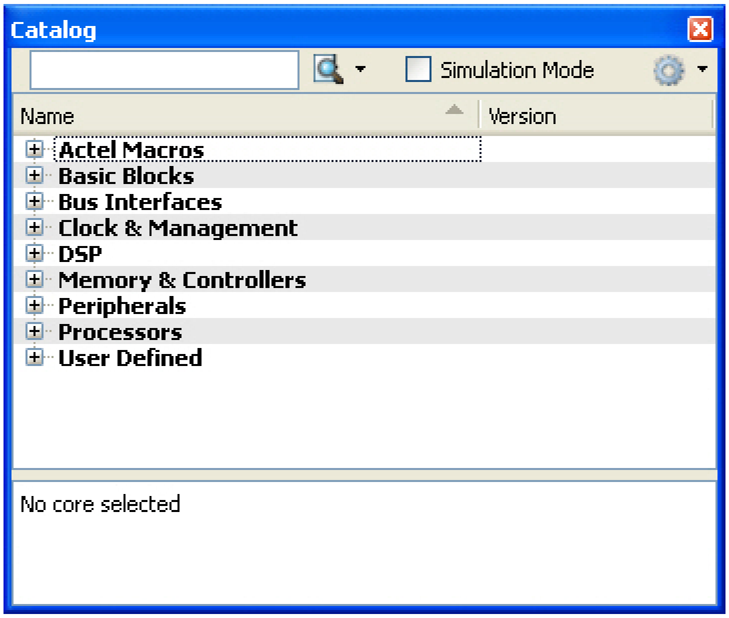
- Design files using Softconsole (RAR, 15.2 MB, 5/12)
- Design files using IAR (RAR, 11.9 MB, 5/12)
- Design files using Keil (RAR, 13.5 MB, 5/12)
- Programming File (RAR, 226 KB, 5/12)

Using Keil µVision and Microsemi SmartFusion (programming files required - 91 KB)- Describes the process of operating an ARM Keil MDL Toolkit featuring µVision and Microsemi's SmartFusion family.

Catalog

In the Libero SoC, from the **View** menu choose **Windows > Catalog**.

The Catalog displays a list of available cores, busses and macros (see image below).



- Actel Macros
- Basic Blocks
- Bus Interfaces
- Clock & Management
- DSP
- Memory & Controllers
- Peripherals
- Processors
- User Defined

No core selected

Figure 84 · Libero SoC Catalog

From the Catalog, you can create a component from the list of available cores, add a processor or peripheral, add a bus interface to your SmartDesign component, instantiate simulation cores or add a macro (Arithmetic, Basic Block, etc.) to your SmartDesign component.

Double-click a core to configure it and add it to your design. Configured cores are added to your list of Components/Modules in the Design Explorer.

Click the Simulation Mode checkbox to instantiate simulation cores in your SmartDesign Testbench. Simulation cores are basic cores that are useful for stimulus, such as driving clocks, resets, and pulses.

Viewing Cores in the Catalog

The font indicates the status of the core:

- Plain text - In vault and available for use
- Asterisk after name (*) - Newer version of the core (VLN) available for download
- *Italics* - Core is available for download but not in your vault
- ~~Strikethrough~~ - core is not valid for this version of Libero SoC

The colored icons indicate the license status. Blank means that the core is not license protected in any way. Colored icons mean that the core is license protected, with the following meanings:

Green Key - Fully licensed; supports the entire design flow.

Yellow Key - Has a limited or evaluation license only. Precompiled simulation libraries are provided, enabling the core to be instantiated and simulated within Libero SoC. Using the Evaluation version of the core it is possible to create and simulate the complete design in which the core is being included. The design is not synthesizable (RTL code is not provided). No license feature in the license.dat file is needed to run the core in evaluation mode.You can purchase a license to generate an obfuscated or RTL netlist.

Yellow Key with Red Circle - License is protected; you are not licensed to use this core.

Right-click any item in the Catalog and choose Show Details for a short summary of the core specifications. Choose Open Documentation for more information on the Core. Right-click and choose Configure Core to open the core generator.

Click the **Name** column heading to sort the cores alphabetically.

You can filter the cores according to the data in the Name and Description fields. Type the data into the filter field to view the cores that match the filter. You may find it helpful to set the Catalog Display Options to **List cores alphabetically** when using the filters to search for cores. By default the filter contains a beginning and ending '*', so if you type 'controller' you get all cores with controller in the core name (case insensitive search) or in the core description. For example, to list all the Accumulator cores, in the filter field type:

```
accu
```

Catalog Options

Click the Options button ⚙ to customize the Catalog Display Options. Click the Catalog Options drop-down arrow to import a core, reload the Catalog, or enter advanced download mode.

You may want to import a core from a file when:

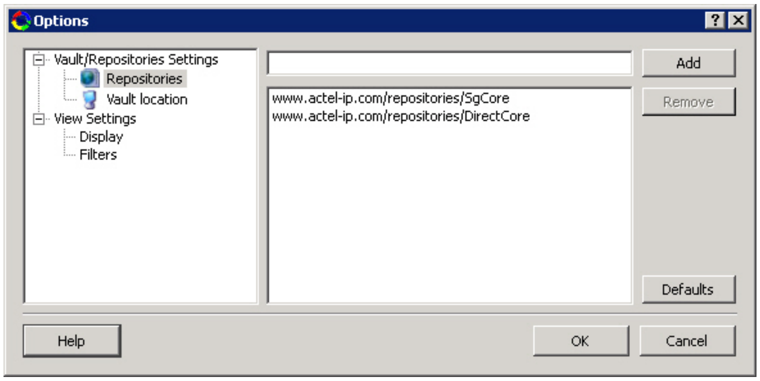
- You do not have access to the internet and cannot download the core, or
- A core is not complete and has not been posted to the web (you have an evaluation core)

See Also

Project Manager - Cores Dialog Box (Advanced Download Mode)

Catalog Options Dialog Box

The Catalog Options dialog box (as shown below) enables you to customize your Catalog display. You can add a repository, set the location of your vault, and change the View Settings for the Catalog. To display this dialog box, click the Catalog Options button ⚙.



www.actel-ip.com/repositories/SgCore
www.actel-ip.com/repositories/DirectCore

Figure 85 · Catalog Display Options Dialog Box

Vault/Repositories Settings

Repositories

A repository is a location on the web that contains cores that can be included in your design.

The Catalog Options dialog box enables you to specify which repositories you want to display in your Vault. The Vault displays a list of cores from all your repositories, and the Catalog displays all the cores in your Vault.

The default repository cannot be permanently deleted; it is restored each time you open the Manage Repositories dialog box.

Any cores stored in the repository are listed by name in your Vault and Catalog; repository cores displayed in your Catalog can be filtered like any other core.

Type in the address and click the **Add** button to add new repositories. Click the **Remove** button to remove a repository (and its contents) from your Vault and Catalog. Removing a repository from the list removes the repository contents from your Vault.

Vault location

Use this option to choose a new vault location on your local network. Enter the full domain pathname in the Select new vault location field. Use the format:

```
\\server\share
```

and the cores in your Vault will be listed in the Catalog.

View Settings

Display

Group cores by function - Displays a list of cores, sorted by function. Click any function to expand the list and view specific cores.

List cores alphabetically - Displays an expanded list of all cores, sorted alphabetically. Double click a core to configure it. This view is often the best option if you are using the filters to customize your display.

Show core version - Shows/hides the core version.

Filters

Filter field - Type text in the Filter Field to display only cores that match the text in your filter. For example, to view cores that include 'sub' in the name, set the Filter Field to **Name** and type **sub**.

Display only latest version of a core - Shows/hides older versions of cores; this feature is useful if you are designing with an older family and wish to use an older core.

Show all local and remote cores - Displays all cores in your Catalog.

Show local cores only - Displays only the cores in your local vault in your Catalog; omits any remote cores.

Show remote cores that are not in my vault - Displays remote cores that have not been added to your vault in your Catalog.

Changing Device Information

Device and package information, device variations, and operating conditions are set when you import a netlist and compile a new design. However, you can change this information for existing designs.

To change device information for existing designs:

1. In the **Project** menu, choose **Project Settings**. The Project Settings dialog box opens.
2. Select your updated options, such as Die, Package, and Speed.
3. Click **Close**.

Refer to the Microsemi FPGA Data Book or call your local Microsemi Sales Representative for information about device, package, speed grade, variations, and operating conditions.

Compatible Die Change

When you change the device, some design information can be preserved depending on the type of change.

Changing Die Revisions

If you change the die from one technology to another, all information except timing is preserved. An example is changing an A1020A (1.2µm) to an A1020B (1.0µm) while keeping the package the same.

Device Change Only

Constraint and pin information is preserved, when possible. An example is changing an A1240A in a PL84 package to an A1280A in a PL84 package.

Repackager Function

When the package is changed (for the same device), the Repackager automatically attempts to preserve the existing pin and Layout information by mapping external pin names based on the physical bonding diagrams. This always works when changing from a smaller package to a larger package (or one of the same size). When changing to a smaller package, the Repackager determines if any of the currently assigned I/Os are mapped differently on the smaller package. If any of the I/Os are mapped differently, then the layout is invalidated and the unassigned pins identified.

Core Manager

The Core Manager only lists cores that are in your current project. If any of the cores in your current project are not in your vault, you can use the Core Manager to download them all at once.

For example, if you download a sample project and open it, you may not have all the cores in your local vault. In this instance you can use the Core Manager to view and download them with one click. Click **Download All** to add any missing cores to your vault. To add any individual core, click the green download button.

To view the Core Manager, from the **View** menu choose **Windows > Cores**.

The column headings in the Core Manager are:

- **Name** - Core name.
- **Vendor** - Source of the core.
- **Core Type** - Core type.
- **Version** - Version of the core used in your project; it may be a later version than you have in your vault. If so, click **Download All** to download the latest version.

Deleting Files

Files can be deleted from the current project or from the disk.

To delete a file from the project:

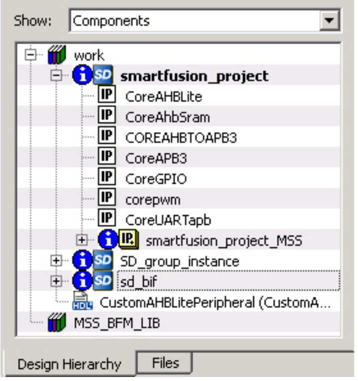
1. Select the **Files** tab in the Design Explorer window.
2. **Right-click** the file and choose **Delete from Project**. The file remains on your disk.

To delete a file from your project and the disk:

1. Select the **Files** tab in the Design Explorer window.
2. **Right-click** the file and choose **Delete from Disk and Project**. The file is deleted from your disk and is no longer part of any project.

Design Hierarchy in the Design Explorer

The Design Hierarchy tab displays a hierarchical representation of the design based on the source files in the project. The software continuously analyzes and updates source files and updates the content. The Design Hierarchy tab (see figure below) displays the structure of the modules and components as they relate to each other.



Show: Components

```
work
  smartfusion_project
    CoreAHBLite
    CoreAhbSram
    COREAHBTOAPB3
    CoreAPB3
    CoreGPIO
    corepwm
    CoreUARTapb
    smartfusion_project_MSS
  SD_group_instance
  sd_bif
  CustomAHBLitePeripheral (CustomA...
MSS_BFM_LIB
```

Figure 86 · Design Hierarchy

You can change the display mode of the Design Hierarchy by selecting **Components** or **Modules** from the **Show** drop-down list. The components view displays the entire design hierarchy; the modules view displays only schematic and HDL modules.

The file name (the file that defines the block) appears next to the block name in parentheses.

To view the location of a component, right-click and choose **Properties**. The Properties dialog box displays the pathname, created date, and last modified date.

All integrated source editors are linked with the SoC software. If a source is modified and the modification changes the hierarchy of the design, the Design Hierarchy automatically updates to reflect the change.

If you want to update the Design Hierarchy, from the **View** menu, choose **Refresh Design Hierarchy.**

To open a component:

Double-click a component in the Design Hierarchy to open it. Depending on the block type and design state, several possible options are available from the right-click menu. You can instantiate a component from the Design Hierarchy to the Canvas in SmartDesign.

Icons in the Hierarchy indicate the type of component and the state, as shown in the table below.

Table 13 · Design Hierarchy Icons

Icon	Description
SD	SmartDesign component
ℹ SD	SmartDesign component with HDL netlist not generated
IP	IP core was instantiated into SmartDesign but the HDL netlist has not been generated
	Core
✖	Error during core validation
⚠	Updated core available for download
HDL	HDL netlist

Design Menu - Libero SoC

Command	Icon	Function
Configure Firmware		Opens the firmware view
Export Back Annotated Files	▶	Runs the push-button flow from synthesis through layout, compile, and place and route.
Reports		Creates and/or opens the Datasheet Reports for your project

Designer in Libero SoC

Microsemi's Designer software is integrated with the Libero SoC Project Manager. The Designer interface opens only when you choose not to use the default settings in the push-button design flow.

To implement your design, click the Build button in the Libero SoC Design Flow window. If you wish to change the default settings for any element in the design flow, right-click the function and choose **Open Interactively**.

The following tools are available to run interactively:

SmartTime

SmartPower

NetlistViewer

PinEditor

ChipPlanner

I/O Attribute Editor

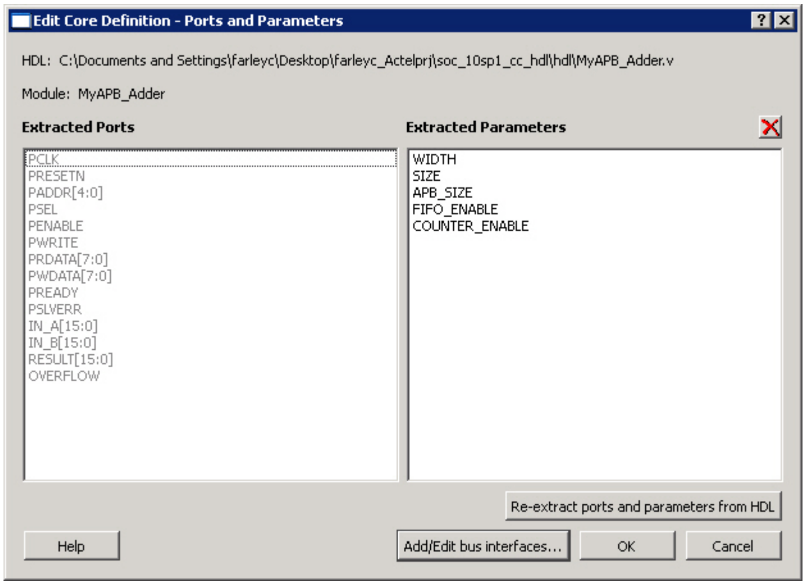
Edit Core Definition - Ports and Parameters Dialog Box

This dialog box appears when you add a core you created with HDL+.

Click to select any Extracted Parameter and click the Delete button to remove it from the list. Extracted Parameters may be configured if you add the HDL+ core to the Canvas.

If you delete an Extracted Parameter and want to re-add it to the list click the **Re-extract ports and parameters from HDL button**.

Click **Add/Edit bus interfaces** to open the Edit Core Definition - Bus Interfaces dialog box.



Edit Core Definition - Ports and Parameters

HDL: C:\Documents and Settings\farleyc\Desktop\farleyc_Actelprj\soc_10sp1_cc_hdl\hdl\MyAPB_Adder.v

Module: MyAPB_Adder

Extracted Ports

PCLK
PRESETN
PADDR[4:0]
PSEL
PENABLE
PWRITE
PRDATA[7:0]
PWDATA[7:0]
PREADY
PSLVERR
IN_A[15:0]
IN_B[15:0]
RESULT[15:0]
OVERFLOW

Extracted Parameters

WIDTH
SIZE
APB_SIZE
FIFO_ENABLE
COUNTER_ENABLE

Re-extract ports and parameters from HDL
Help
Add/Edit bus interfaces...
OK
Cancel

Figure 87 · Edit Core Definition - Ports and Parameters Dialog Box

Edit Menu - Libero SoC

Command	Icon	Shortcut	Function
Undo	↶	CTRL + Z	Reverses your last action
Redo	↷	CTRL + Y	Reverses the action of your last Undo command
Find		CTRL + F	Displays the Find dialog box, which you use to locate instances, nets, ports, and regions
Find Next		F3	Finds the next occurrence of the text in the Find field
Replace		CTRL + H	Displays the Replace dialog box; enables you to search and replace content in your files (files must be open and selected to use this feature)

Execute Script Dialog box

You can use the Execute Script dialog box to run Tcl scripts from within Libero SoC. You do not need to have a design open in order to run a script.

Specify a script file, enter Arguments (if necessary), and click Run to execute.

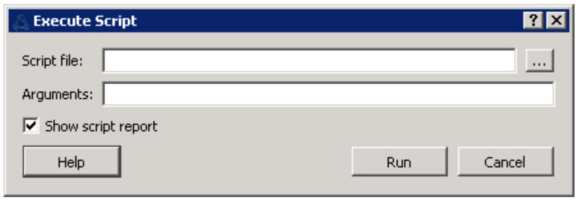


Figure 88 · Execute Script Dialog Box

Script file

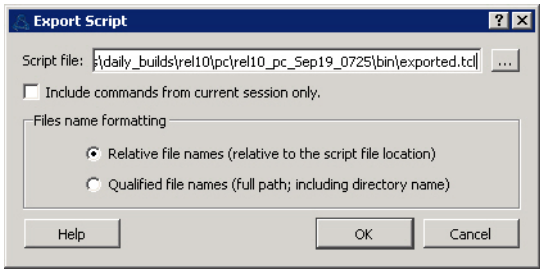
Specify a script file. Browse to Select a script file with a valid extension (*.tcl or *.dsf).

Arguments

Input your arguments for your script file (if necessary).

Export Script Dialog Box

The Export Script Files dialog box enables you to export Tcl script file, useful if you want to run Libero SoC in batch mode or run operations from the command line.



Script file: :\daily_builds\rel10\pc\rel10_pc_Sep19_0725\bin\exported.tcl

Export Script
?
X
...
□ Include commands from current session only.
Files name formatting
● Relative file names (relative to the script file location)
○ Qualified file names (full path; including directory name)
Help
OK
Cancel

Figure 89 · Export Script Dialog Box

Script file

Specifies the location of the file you are about to save.

Include commands from current session only limits your commands to the current session. De-select if you wish to include commands from other sessions.

File name formatting

Relative file names (relative to the script file location) truncates all the directories in the script with relative filenames. Select this option if you do not plan to move the script file.

Qualified file names (full path; including directory name) includes the full pathname for all the files and directories. Select this option if you want to move the file to a different directory.

File Menu - Libero SoC

Command	Icon	Shortcut	Sub-menu	Function
New			SmartDesign	Opens the appropriate New file dialog box and prompts you to enter a name and specify additional options (if necessary)
			HDL	
			SmartDesign Testbench	
			HDL Testbench	
			SDC (sdc)	
			Physical Design Constraint (PDC)	
			Simulation Script (do)	
Open				Opens the Open dialog box; enables you to select a file to open
Close <filename>				Closes the current file; the Project Manager remains open
Save <filename>	💾	Ctrl + S		Saves the current file
Save <filename> As				Saves the current file as a different type (such as a TXT file)
Import Files				Opens the Import Files dialog box; enables you to import project files into the Project Manager
Link Files			Create Link	Opens the Create Link dialog box; browse to select the file you wish to link. Linked files are added to the Design Explorer in the Modules defined in multiple files list.

Command	Icon	Shortcut	Sub-menu	Function
			Change All Links	Opens the Change All Links dialog box; enables you to update/change all the links for the files in your project at once.
			Unlink All: Copy Files Locally	Copies all linked files to your local project.
VHDL Library >			Add Library	Adds VHDL library to your Design Hieararchy
			Rename Library	Renames an existing VHDL library
			Remove Library	Removes an existing VHDL library from your project
Print	🖨			Displays the Print dialog box (if available); allows you to print whatever element of the project you are working on

Files Tab and File Types

The Files tab displays all the files associated with your project, listed in the directories in which they appear.

Right-clicking a file in the Files tab provides a menu of available options specific to the file type. You can also delete files from the project by selecting **Delete from Project** from the right-click menu. You can delete files from the project and the disk by selecting **Delete from Disk and Project** from the right-click menu.

You can instantiate a component by dragging the component to a SmartDesign Canvas or by selecting **Instantiate in SmartDesign** from the right-click menu.

You can configure a component by double-clicking the component or by selecting **Open Component** from the right-click menu.

File Types

When you create a new project in the Libero SoC it automatically creates new directories and project files. Your project directory contains all of your 'local' project files. If you import files from outside your current project, the files must be copied into your local project folder. (The Project Manager enables you to manage your files as you import them.)

Depending on your project preferences and the version of Libero SoC you installed, the software creates directories for your project.

The top level directory (<project_name>) contains your PRJ file; only one PRJ file is enabled for each Libero SoC project.

component directory - Stores your SmartDesign components (SDB and CXF files) for your Libero SoC project.

constraint directory **-** All your constraint files (SDC, PDC, GCF, DCF, etc.)

designer directory **-** ADB files (Microsemi Designer project files), -_ba.SDF, _ba.v(hd), STP, PRB (for Silicon Explorer), TCL (used to run designer), impl.prj_des (local project file relative to revision), designer.log (logfile)

Note: Note: The Microsemi ADB file memory requirement is equivalent to 2x the size of the ADB file. If your computer does not have 2x the size of your ADB file's memory available, please make memory available on your hard drive.

hdl directory **-** all hdl sources. *.vhd if VHDL, *.v and *.h if Verilog

phy_synthesis directory **-** _palace.edn, _palace.gcf, palace_top.rpt (palace logfile) and other files generated by PALACE

simulation directory - meminit.dat, modelsim.ini files

smartgen directory - GEN files and LOG files from generated cores

stimulus directory - BTIM and VHD stimulus files

synthesis directory - *.edn, *_syn.prj (Synplify log file), *.psp (Precision project file), *.srr (Synplify logfile), precision.log (Precision logfile), *.tcl (used to run synthesis) and many other files generated by the tools (not managed by Libero SoC)

viewdraw directory - viewdraw.ini files

HDL Templates in the Libero SoC

Use the templates in the Libero SoC Project Manager to create HDL.

To use the templates included with the Project Manager, from the View menu, choose Windows > HDL Templates. Find the template you want to use and double-click to add it to your HDL file.

Place the cursor where you want to add the template, browse the list of VHDL and Verilog templates, and double-click the template to add it to your design.

The VHDL and Verilog templates are useful if you want to modify your netlist but are unfamiliar with the language. The templates facilitate the writing of HDL files by inserting predefined language constructs. You can also save your own template files to reuse in other designs (for example, if you wanted to add the same header in all your files).

To create a user template:

- Import an HDL file as a template, or
- Save an open HDL file from the text editor as a template. To do so, right-click in the text editor and choose Export as Template.

Help Menu - Libero SoC

The Help menu enables you to access the Libero SoC online help, reference manuals, check for updates, and view your license and version information.

Command	Function
Help Topics	Opens the Libero Project Manager online help
Core	Displays a list of PDF files associated with the cores in your project
Microsemi Technical Support	Displays the Microsemi customer support web page in your default browser
Microsemi Web Site	Displays the main Microsemi page in your default browser
Check for Software Updates	Checks for updates to the Libero Project Manager software
License Details	Displays detailed license information for your version of Libero SoC

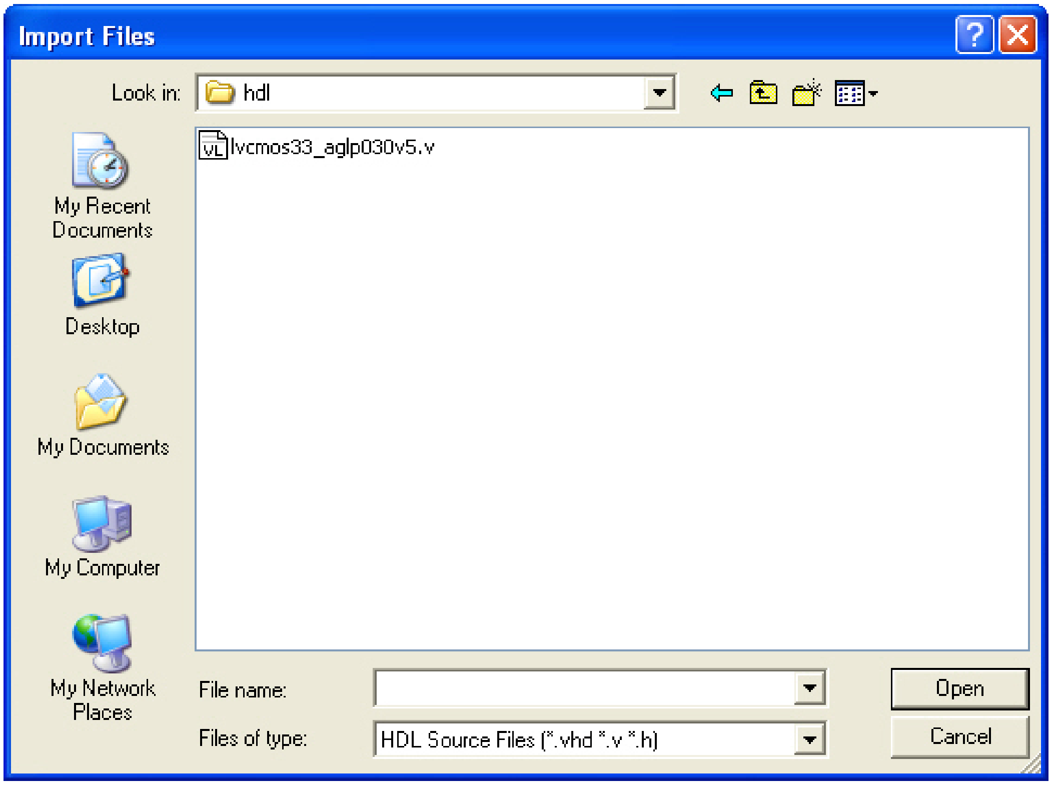
Command	Function
About Libero	Displays version and release numbers for Libero SoC

Import Files Dialog Box (Project Manager)

Use the Import Files dialog box to add new files to your project in the Libero SoC Project Manager.

You can import schematics, VHDL or Verilog source files, stimulus files, SDC, PDC, VCD, and SAIF files, cores, and even tool profiles (from other Libero SoC projects).

Browse to and select the file you wish to add and click **Import**, or click **Cancel** to return to the Project Manager.



Look in

Specifies your current directory. Browse to find your file if it is not listed here. If you are in the correct directory and your file is not listed here, select the **File of type** extension to match it.

File name

Type the file name, or browse to its location and select it.

File of type

Specify the file type displayed in the dialog box.

To access this dialog: from the **File** menu, choose **Import Files**.

Importing Schematics

You can import any schematic created with ViewDraw AE.

To import a schematic file:

1. From the **File** menu, choose **Import Files**.
2. In **Files of type**, choose **Schematics**.
3. In **Look in**, navigate to the drive/folder where the file is located.
4. Select the file to import and click **Open**. The schematic is imported into your project and appears in the Files tab, under Schematic files.

To open the schematic, click ViewDraw AE in the Design Flow window, or right-click the file in the File Manager and select **Open Schematic**.

License Details

To display information about your license:

From the **Help** menu, choose **License Details**. The software displays your complete license configuration, all Microsemi-installed software and versions, as well as your HostID and disk volume serial number.

Link Files

You can add or change links for individual files in your project, or change all the links in your files at once.

To add a link to an individual file, right-click the file in the Files list and choose **Create Link From File**. Navigate to the file you wish to link to your project and click **Create Link**. The Project Manager adds the file to your Files list; a small link icon indicates that the source file is not stored with the project.

If you have a single project file with a broken link , right-click the file and choose **Change Link**. This opens the Change Link dialog box and enables you to specify a new file location.

You can update all the links in your project at once. This is useful when you are linking to shared network folders that may have been renamed or moved. To change links for your entire project, from the **File** menu, choose **Change All Links**. This opens the Change All Links dialog box. Enter (or browse) your old and new paths to update the links for your project.

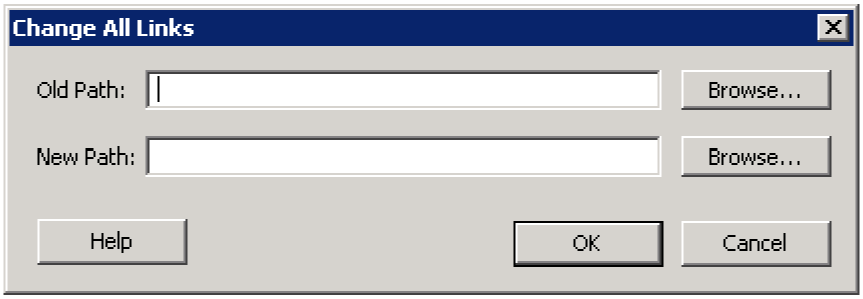


Figure 90 · Change All Links Dialog Box

To unlink a file, right-click the file in the Files tab and choose **Unlink: copy file locally**. This copies the file to the directory in your project folder that corresponds to the file type.

To unlink all files and copy them to your local project, from the **File** menu choose **Unlink All: Copy files locally**.

You can also change/remove links from the Design Explorer; to do so, right-click the file in the **Design Explorer > Modules defined in multiple files** and choose **Change Link**.

Log Window

Colors and Symbols

The log window displays Messages, Errors, Warnings, and Information. Messages are represented by symbols and color-coded. The default colors are:

Type	Color
Error	Red
Warning	Blue
Information	Black

The colors can be changed by using the Preferences dialog box.

Linked Messages

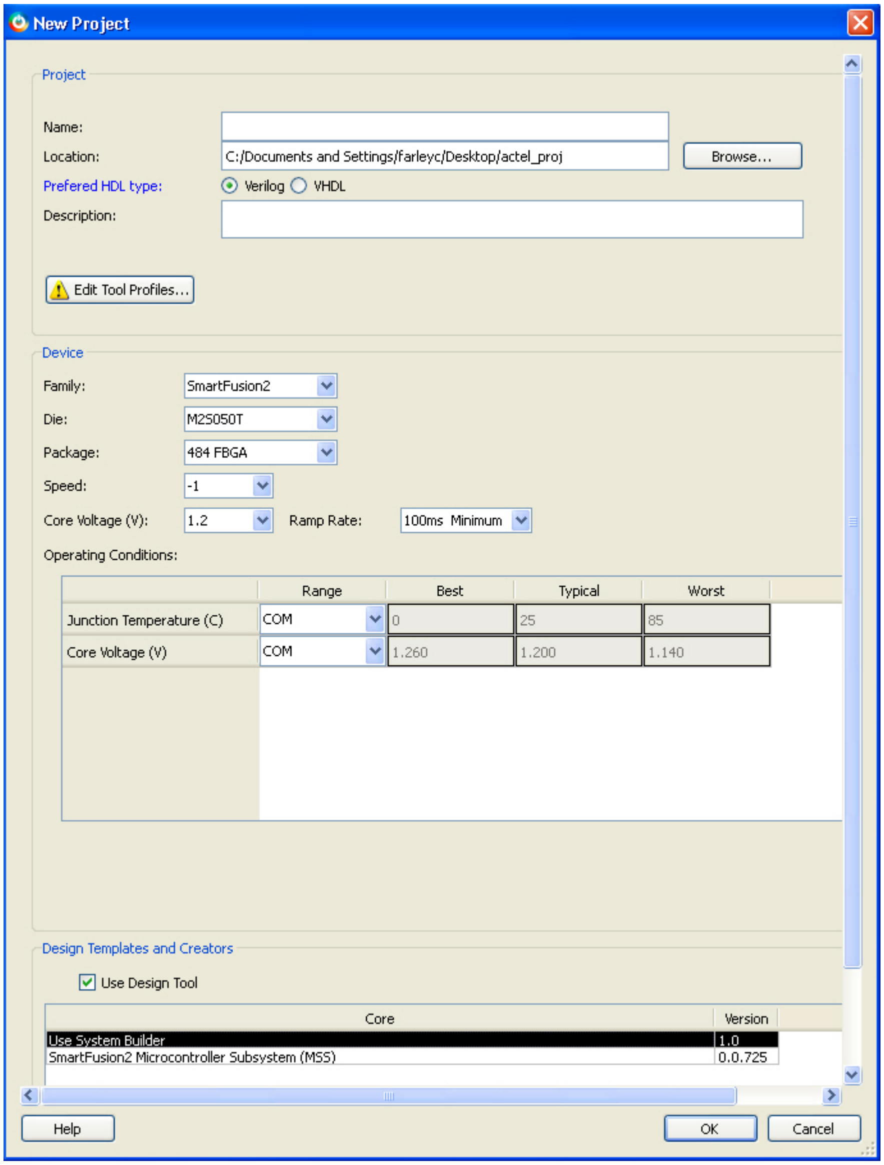
Error and warning messages that are dark blue and underlined are linked to online help to provide you with more details or helpful workarounds. Click them to open online help.

New Project Dialog Box

The New Project dialog box sets your Project and Device settings and select your Design Template (if necessary).

Device information (such as Family, Die and Package) can be modified later in the Project Settings dialog box.

The Project Description appears in your Reports. **You cannot edit your Description after you create your project.**



New Project

Project

Name:

Location: C:/Documents and Settings/farleyc/Desktop/actel_proj

Prefered HDL type: ● Verilog ○ VHDL

Description:

Device

Family: SmartFusion2

Die: M2S050T

Package: 484 FBGA

Speed: -1

Core Voltage (V): 1.2

Ramp Rate: 100ms Minimum

Operating Conditions:

	Range	Best	Typical	Worst
Junction Temperature (C)	COM	0	25	85
Core Voltage (V)	COM	1.260	1.200	1.140

Design Templates and Creators

☑ Use Design Tool

Core	Version
Use System Builder	1.0
SmartFusion2 Microcontroller Subsystem (MSS)	0.0.725

Libero SoC New Project Dialog Box

Project

Name - Identifies your project name; use **Project > Save As** to change the name of your project at any time.

Location - Project location; use **Project > Save As** to change the location of your project and preserve the pathnames of linked files.

Preferred HDL type - Sets your HDL type; Libero SoD supports mixed-HDL designs.

Description - Appears in your Datasheet Report View; you cannot edit your Description after you create a new project.

Device

Family - Sets your device family.

Die / Package / Speed - Sets your device die / package / speed grade, respectively.

Core Voltage - Two numbers separated by a "/" are shown if mixed voltages are supported. Two numbers separated by a "~" are shown if a wide range voltage is supported. If two voltages are shown, the first number is the I/O voltage and the second number is the core (array) voltage.

Ramp Rate - Sophisticated power-up management circuitry is designed into every SmartFusion2 SoC FPGA. These circuits ensure easy transition from the powered-off state to powered-up state of the device. The SmartFusion2 system controller is responsible for systematic power-on reset whenever the device is powered on or reset. All the I/Os are held in a high-impedance state by the system controller until all power supplies are at their required levels and the system controller has completed the reset sequence.

The power-on reset circuitry in SmartFusion2 devices requires the VDD and VPP supplies to ramp monotonically from 0 V to the minimum recommended operating voltage within a predefined time. There is no sequencing requirement on VDD and VPP. Four ramp rate options are available during design generation: 50 µs, 1 ms, 10 ms, and 100 ms. Each selection represents the maximum ramp rate to apply to VDD and VPP.

Operating Conditions

Operating Conditions enable you to define the voltage and temperature ranges a device encounters in a working system. The operating condition range entered here is used by SmartTime, the timing report, and the back-annotation function. These tools enable you to analyze worst-, typical-, and best-case timing.

Supported ranges include:

- Commercial (COM)
- Industrial (IND)
- Military (MIL)
- Custom

Consult the Microsemi Data Sheet for your device, available at http://www.actel.com/techdocs/ds/, to find out which temperature range you should use.

The temperature range represents the junction temperature of the device. For commercial and industrial devices, the junction temperature is a function of ambient temperature, air flow, and power consumption.

For military devices, the junction temperature is a function of the case temperature, air flow, and power consumption. Because Microsemi devices are CMOS, power consumption must be calculated for each design. For most low power applications (e.g. 250mW), the default conditions are adequate.

Performance decreases approximately 2.5% for every 10 degrees C that the temperature values increase. Refer to the SmartPower online help for more information about power consumption.

If you select Custom, you may enter the values for Best, Typical, and Worst.

Wide range voltage for die voltage (VCCA) and VCCI is available for ProASIC3L and 1.2V IGLOO devices. To specify the wide range voltage for VCCI check the **Wide Range** option and enter the values for Best, Typical, and Worst.

New File Dialog Box

The New File dialog box opens when you choose to create any of the following new files:

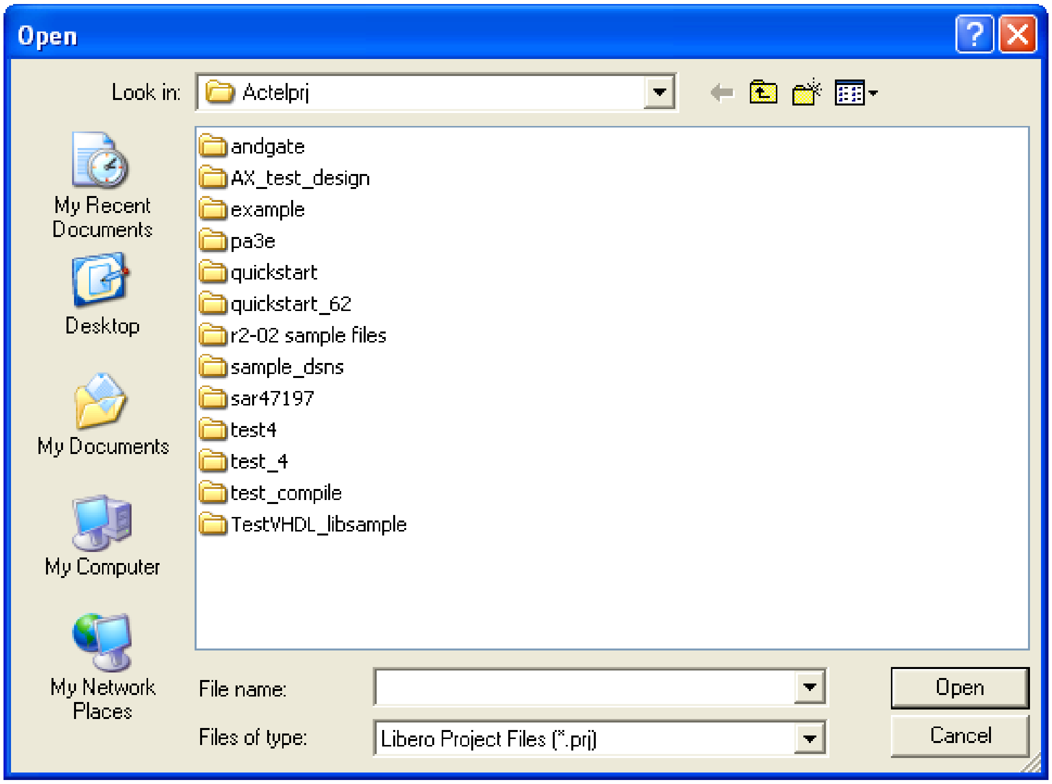
- SmartDesign
- SmartDesign Testbench - Use a SmartDesign to instantiate and connect stimulus cores or modules to drive your Root design.
- HDL
- HDL Testbench - Creates a new HDL testbench in your project. You can use a testbench to apply stimulus, analyze results or to compare the results of two different simulations.
- SDC (sdc)
- Physical Design Constraint (pdc)
- Simulation Script (do)

To create a new file:

1. From the **File** menu, choose **New > <file type>**.
2. Set any additional options (if necessary) and enter a name.
3. Click **OK**. The saved file is added to your Libero SoC project.

Open Project Dialog Box

Use the Open Project dialog box to navigate to and open existing projects in the Project Manager. Browse to your project and click **Open**, or click **Cancel** to return to the Project Manager.



Look in: Actelprj

andgate
AX_test_design
example
pa3e
quickstart
quickstart_62
r2-02 sample files
sample_dsns
sar47197
test4
test_4
test_compile
TestVHDL_libsample

File name:

Files of type: Libero Project Files (*.prj)

Look in

Specifies the directory that contains your project.

File name

Type the file name, or browse to its location and select it.

File of type

Specify the file type displayed in the dialog box.

To access this dialog: from the **Project** menu, select **Open Project**.

Opening your Libero SoC project

Libero SoC does not open your most recent project automatically. You can change your default startup preferences in the Startup tab.

To open a project in Libero SoC:

From the **File** menu, choose **Open Project** or **New Project**. If you create a new project the Project Manager opens the New Project dialog box.

Tip: Recent saved projects are available from the Project menu. From the **Project** menu, choose **Recent Projects**, and then select the project to open.

You can open an existing project by double-clicking the *.prj file or dragging the *.prj file over the Libero SoC desktop icon.

See Also

`open_project`

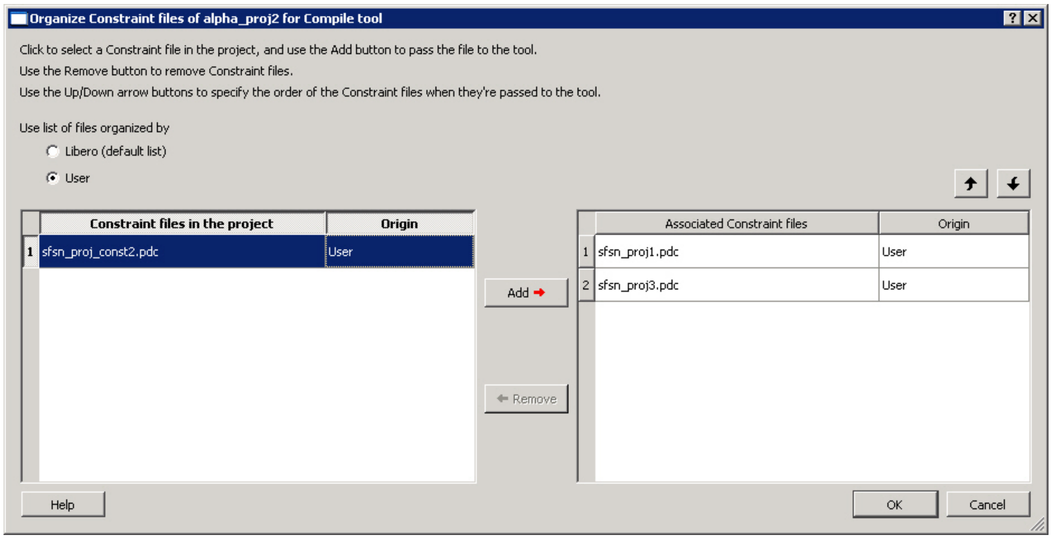
Organize Constraint Files

The Organize Constraint Files dialog box enables you to set the constraint file and order in the Libero SoC.

Click the **Use list of files organized by User** radio button to add or remove Associated Constraint files.

To specify the constraint file order:

1. In the Design Flow window under Implement Design, right-click **Compile** and choose **Organize Input Files > Organize Constraint Files**. The Organize Constraint Files dialog box appears.
2. Click the **Use list of files organized by User** radio button to Add/Remove source files for the selected tool.
3. Select a file and click the Add or Remove buttons as necessary. Use the Up and Down arrows to change the order.
4. Click **OK**.The files appear in the Design Flow window under **Implement Design > Compile > Constraints** with a green check mark to indicate that they are being used in the project.



Organize Constraint files of alpha_proj2 for Compile tool

Click to select a Constraint file in the project, and use the Add button to pass the file to the tool.
Use the Remove button to remove Constraint files.
Use the Up/Down arrow buttons to specify the order of the Constraint files when they're passed to the tool.

Use list of files organized by

○ Libero (default list)

● User

	Constraint files in the project	Origin
1	sfsn_proj_const2.pdc	User

	Associated Constraint files	Origin
1	sfsn_proj1.pdc	User
2	sfsn_proj3.pdc	User

Figure 91 · Organize Constraint Files Dialog Box

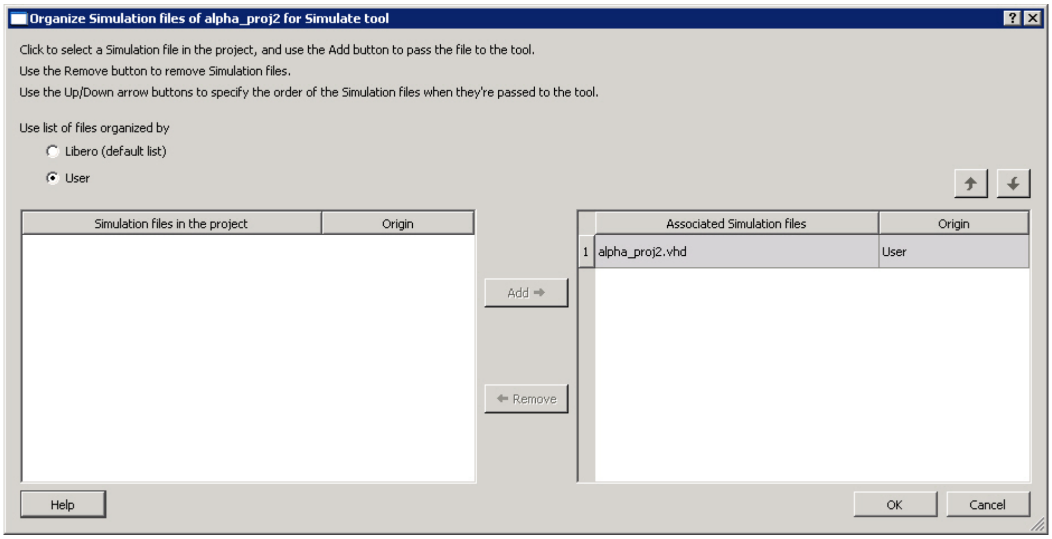
Organize Simulation Files

The Organize Simulation files dialog box enables you to set the constraint file order in the Libero SoC.

Click the **Use list of files organized by User** radio button to add or remove Associated Simulation files.

To specify the simulation file order:

1. In the Design Flow window under Implement Design > Verify Post Layout Implementation, right-click **Simulate** and choose **Organize Input Files > Organize Simulation Files**. The Organize Simulation Files dialog box appears.
2. Click the **Use list of files organized by User** radio button to Add/Remove source files for the selected tool.
3. Select a file and click the Add or Remove buttons as necessary. Use the Up and Down arrows to change the order.
4. Click **OK**.



Organize Simulation files of alpha_proj2 for Simulate tool

Click to select a Simulation file in the project, and use the Add button to pass the file to the tool.
Use the Remove button to remove Simulation files.
Use the Up/Down arrow buttons to specify the order of the Simulation files when they're passed to the tool.

Use list of files organized by

○ Libero (default list)

● User

Simulation files in the project	Origin

	Associated Simulation files	Origin
1	alpha_proj2.vhd	User

Figure 92 · Organize Simulation Files Dialog Box

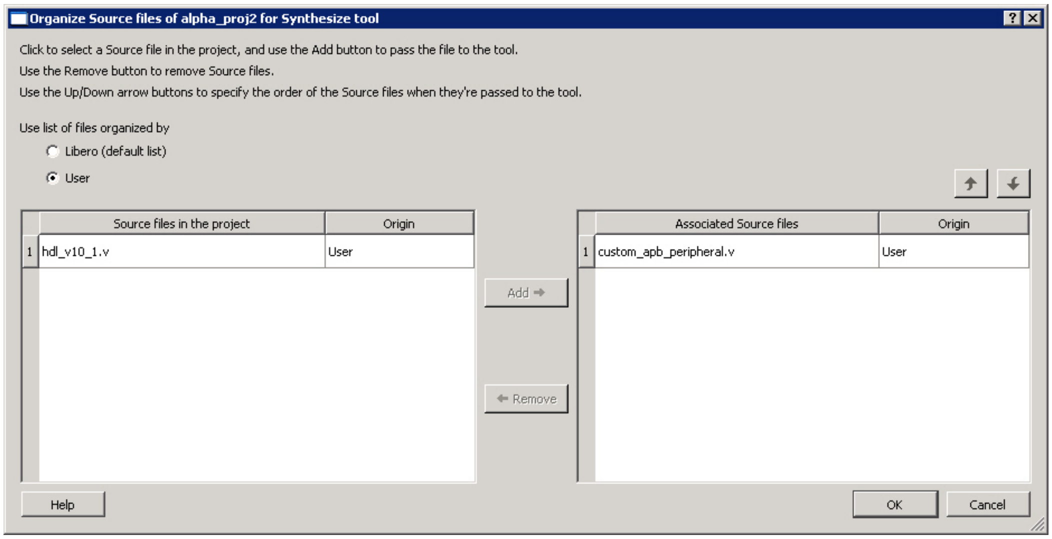
Organize Source Files

The Organize Source Files dialog box enables you to set the source file order in the Libero SoC.

Click the **Use list of files organized by User** radio button to Add/Remove source files for the selected tool.

To specify the file order:

1. In the Design Flow window under Implement Design, right-click **Synthesize** and choose **Organize Input Files > Organize Source Files**. The Organize Source Files dialog box appears.
2. Click the **Use list of files organized by User** radio button to Add/Remove source files for the selected tool.
3. Select a file and click the Add or Remove buttons as necessary. Use the Up and Down arrows to change the order of the Associated Source files.
4. Click **OK**.



Organize Source files of alpha_proj2 for Synthesize tool

Click to select a Source file in the project, and use the Add button to pass the file to the tool.
Use the Remove button to remove Source files.
Use the Up/Down arrow buttons to specify the order of the Source files when they're passed to the tool.

Use list of files organized by

○ Libero (default list)

● User

	Source files in the project	Origin
1	hdl_v10_1.v	User

	Associated Source files	Origin
1	custom_apb_peripheral.v	User

Figure 93 · Organize Source Files Dialog Box

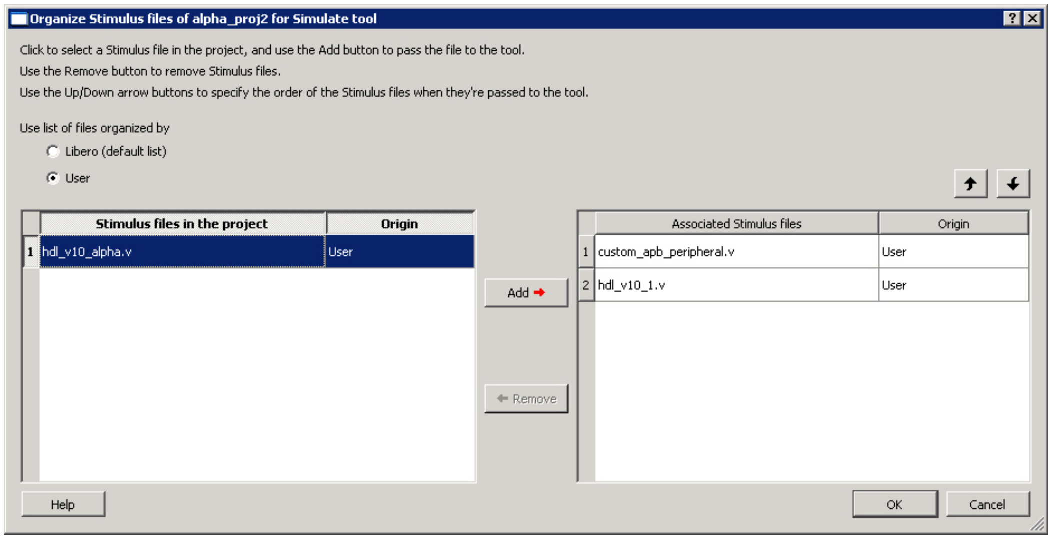
Organize Stimulus Files Dialog Box

The Organize Stimulus files dialog box enables you to set the stimulus file order in the Libero SoC.

Click the **Use list of files organized by User** radio button to add or remove Associated Stimulus files.

To specify the stimulus file order:

1. In the Design Flow window under Create Design > Verify Pre-Synthesized Design, right-click **Simulate** and choose **Organize Input Files > Organize Stimulus Files**. The Organize Stimulus Files dialog box appears.
2. Click the **Use list of files organized by User** radio button to Add/Remove source files for the selected tool.
3. Select a file and click the Add or Remove buttons as necessary. Use the Up and Down arrows to change the order.
4. Click **OK**.



Organize Stimulus files of alpha_proj2 for Simulate tool

Click to select a Stimulus file in the project, and use the Add button to pass the file to the tool.
Use the Remove button to remove Stimulus files.
Use the Up/Down arrow buttons to specify the order of the Stimulus files when they're passed to the tool.

Use list of files organized by

○ Libero (default list)

● User

	Stimulus files in the project	Origin
1	hdl_v10_alpha.v	User

	Associated Stimulus files	Origin
1	custom_apb_peripheral.v	User
2	hdl_v10_1.v	User

Figure 94 · Organize Stimulus Files Dialog Box

Physical Synthesis and the Libero SoC

If you want to run physical synthesis on your design (such as with PALACE) you must run it manually.

Automatic physical synthesis is not supported from within the Libero SoC.

Preferences

Preferences Dialog Box

Use the Preferences dialog to customize the Libero SoC Project Manager.

To set your preferences:

1. From the **Project** menu, choose **Preferences**.
2. Specify your preferences.

Software update

Log window

Vault update

Startup (File association)

Text Editor

IP Cores

Proxy

PDF reader (UNIX only)

Web browser (UNIX only)

3. Click **OK**.

Note: Note: These preferences are stored on a per-user basis; they are not project specific.

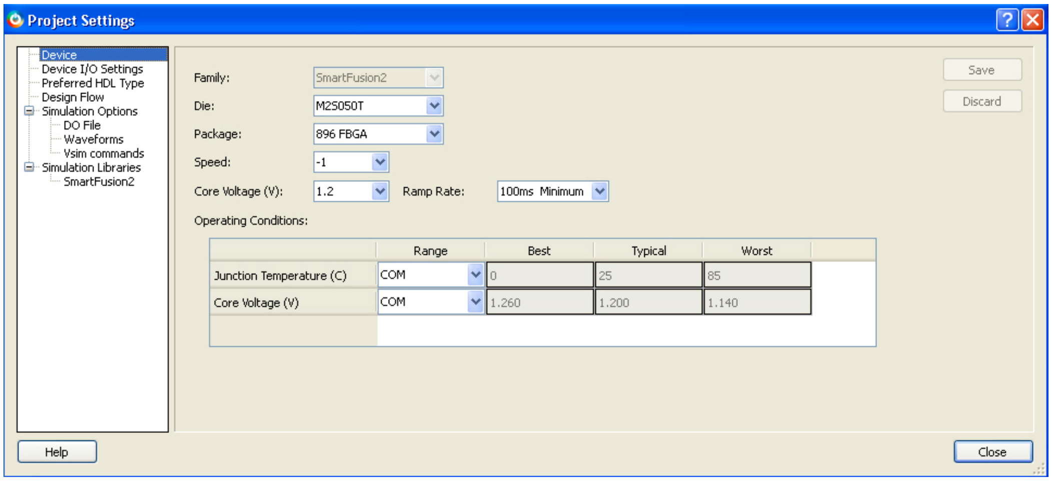
Project Menu - Libero SoC

Command	Sub-menu	Icon	Function
New Project		🗋	Starts the New Project Wizard
Open Project		📂	Opens the Open Project dialog box
Close <project name>			Closes the current active project; the Project Manager remains open
Save <project name>		💾	Saves the current project
Save <project name>			Saves the current project in a new directory; prompts you to enter a new project name
Project Settings			Opens the Project Settings dialog box, enables you to set your Device, HDL Type, Design Flow,

Command	Sub-menu	Icon	Function
			Simulation and Simulation Library options.
Tool Profiles			Opens the Project Profile dialog box; enables you to specify locations for your third-party synthesis, stimulus, and simulation tools. Libero SoC includes tools for synthesis, stimulus, and simulation.
Vault/Repositories Settings			Opens the Vault/Repositories Settings dialog box; enables you to view/change the location of your vault and repositories.
Preferences			Opens the Preferences dialog box
Execute Script			Opens Execute Script dialog box; enables you to run Tcl script from the Project Manager
Export Script			Opens the Export Script dialog box; enables you to export a Tcl script
Recent Projects			Opens list of recent projects.
Exit			Closes Libero SoC

Project Settings Dialog Box

The Project Settings dialog box enables you to modify your Device, HDL, and Design Flow settings and your Simulation Options.



Family: SmartFusion2

Die: M2S050T

Package: 896 FBGA

Speed: -1

Core Voltage (V): 1.2

Ramp Rate: 100ms Minimum

Operating Conditions:

	Range	Best	Typical	Worst
Junction Temperature (C)	COM	0	25	85
Core Voltage (V)	COM	1.260	1.200	1.140

Figure 95 · Libero SoC Project Settings Dialog Box

Device

Sets the device settings for your project. See the New Project dialog box for a detailed description of the options.

Device I/O Settings

Reserve Pins for Probes - Reserve your pins for probing if you intend to debug using SmartDebug.

Default I/O Technology - Sets all your I/Os to a default value. You can change the values for individual I/Os in the I/O Attribute Editor.

Default I/O Voltage Range displays your possible voltages based on your specified Operating Conditions; change your Operating Conditions to change the voltage range.

Preferred HDL Type

Sets your HDL type to VHDL or Verilog.

Design Flow

Block Flow

Enable Designer Block creation - Enables you to create Designer Blocks in the Libero SoC. Designer Blocks are useful if you want to create a block and re-use it in several designs. See the Block help for more information.

File Detection

Detect new files on disk automatically - Libero SoC checks and adds any new files that appear in your project directory. This feature is useful if you work on constraint files outside of the Project Manager and copy the files into your project directory when you are done.

Root <filename>

Enable Synthesis - Option to enable or disable synthesis for your root file; useful if you wish to skip synthesis on your root file by default.

ViewDraw

Click the checkbox to enable ViewDraw in the Design Flow window.

Generate HDL netlist after a Save&Check in ViewDraw - Useful if you wish to eliminate the manual step of generating your HDL netlist after a Save&Check.

Update viewdraw.ini automatically - May be useful if the Project Manager does not create a valid viewdraw.ini file. Click the checkbox to enable.

File Detection

Detect new files on disk automatically enables the software to see new files when you add them to your project. If you deselect this option, you must add the new files manually.

Root Datasheet

Enable Synthesis - Set this option to run synthesis on your root datasheet.

Compile

Default I/O Attribute sets your default global I/O attribute; useful if you are working with a device that has a unique I/O attribute.

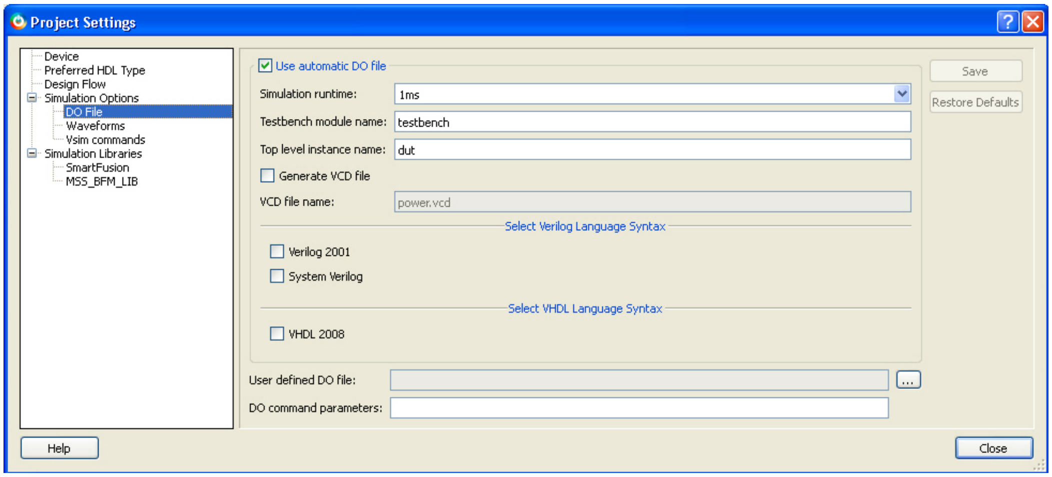
Simulation Options and Simulation Libraries

Sets your simulation options; see the Simulation Options topic for a full summary.

Project Settings: Simulation

To access this dialog box, from the **Project** menu choose **Project Settings** and click **Simulation Options > DO File**.

Use the Simulation tab to set your simulation values in your project. You can set change how Libero SoC handles Do files in simulation, import your own Do files, set simulation run time, and change the resolution of your simulation. You can also change your library mapping in this dialog box.



Project Settings

- Device
- Preferred HDL Type
- Design Flow
- Simulation Options
  - DO File
  - Waveforms
  - Vsim commands
- Simulation Libraries
  - SmartFusion
  - MSS_BFM_LIB

☑ Use automatic DO file

Simulation runtime: 1ms

Testbench module name: testbench

Top level instance name: dut

☐ Generate VCD file

VCD file name: power.vcd

Select Verilog Language Syntax

☐ Verilog 2001

☐ System Verilog

Select VHDL Language Syntax

☐ VHDL 2008

User defined DO file:

DO command parameters:

Figure 96 · Project Settings: Simulation Options - DO File

DO file

Use automatic DO file

Select if you want the Project Manager to automatically create a DO file that will enable you to simulate your design.

Simulation Run Time - Specify how long the simulation should run. If the value is 0, or if the field is empty, there will not be a run command included in the run.do file.

Testbench module name - Specify the name of your testbench entity name. Default is “testbench,” the value used by WaveFormer Pro.

Top Level instance name - Default is <top_0>, the value used by WaveFormer Pro. The Project Manager replaces <top> with the actual top level macro when you run ModelSim.

Generate VCD file - Click the checkbox to generate a VCD file.

VCD file name - Specifies the name of your generated VCD file. The default is power.vcd; click power.vcd and type to change the name.

Select Verilog Language Syntax

Sets your DO file Verilog language syntax.

Select VHDL Language Syntax

VHDL 2008 - Select if you wish to use VHDL 2008 for your DO file.

User defined DO file - Enter the DO file name or click the browse button to navigate to it.

DO command parameters - Text in this field is added to the DO command.

Waveforms

Include DO file - Including a DO file enables you to customize the set of signal waveforms that will be displayed in Model*Sim*.

Display waveforms for - You can display signal waveforms for either the top-level testbench or for the design under test. If you select **top-level testbench** then Project Manager outputs the line 'add wave /testbench/*' in the DO file run.do. If you select **DUT** then Project Manager outputs the line 'add wave /testbench/*' in the run.do file.

Log all signals in the design - Saves and logs all signals during simulation.

Vsim Commands

SDF timing delays - Select Minimum (Min), Typical (Typ), or Maximum (Max) timing delays in the back-annotated SDF file.

Resolution

The default is family specific (review the dialog box for your default setting), but you can customize it to fit your needs.

Additional options - Text entered in this field is added to the vsim command.

Simulation Libraries

Use default library path - Sets the library path to the default from your Libero SoC installation.

Library path - Enables you to change the mapping for your VHDL library. Type in the pathname or click the Browse button to navigate to your library directory.

Project Sources

Project sources are any design files that make up your design. These can include schematics, HDL files, simulation files, testbenches, etc. Anything that describes your design or is needed to program the device is a project source.

Source files appear in the Project Flow window. The Design Hierarchy tab displays the structure of the design modules as they relate to each other, while the Files tab displays all the files that make up the project.

The design description for a project is contained within the following types of sources:

- Schematics
- HDL Files (VHDL or Verilog)
- SmartDesign components

One source file in the project is the top-level source for the design. The top-level source defines the inputs and outputs that will be mapped into the devices, and references the logic descriptions contained in lower-level sources. The referencing of another source is called an *instantiation*. Lower-level sources can also instantiate sources to build as many levels of logic as necessary to describe your design.

File Linking

The Project Manager enables you to link to files not managed in your Libero project. Linked files are useful if you want to preserve a file in an archive, or if more than one person is using a file and it is impractical to store it on your local machine. If you link to external files and rename your project, the Project Manager asks if you want to copy the external files into your project or continue using the link. Note that some files (such as schematics) cannot be linked.

Some project sources can be imported.

Sources for your project can include:

Source	File Extension
Schematic	*.1-9
Verilog Module	*.v
VHDL Entity	*.vhd

Source	File Extension
SmartDesign Component	*.vhd
Testbench	*.vhd
Stimulus	*.tim
Programming Files	*.afm; *.prb

See Also

Creating HDL Sources

Generating a Bitstream file

Generating Programming files

Reserved Microsemi Keywords

For a complete list of reserved Microsemi keywords, see the online help.

Right-Click (Shortcut) Menu Options in Libero SoC Design Hierarchy

Right-click menu options vary depending on your design state.

The option in bold the right-click menu is the action performed when you double-click the tool. For example, if you expand Implement Design and right-click **Synthesize**, Run is bold, indicating that it is the default action when you double-click the tool in the Design Hierarchy.

- **Run** - Runs the current tool. If any predecessor tools are required to be in the PASSED state, then they will be run as well.
- **Clean and Run All**- Clean all predecessor tools (deletes Report and output files) and run up to this tool.
- **Clean** - Delete report and output files of this tool. Subsequent tools become OUT OF DATE.
- **Open Interactively** - Open the tool to set/change the tool options.
- **Update and Run** -- Available if a tool is in the OUT OF DATE state; it cleans all predecessor tools that are in the OUT OF DATE state and runs up to this tool.
- **Run Synthesize > Compile > Place and Route > Verify Timing > Generate Programming Data > Program Device** - Enables you to bypass the Fabric portion of the design flow.

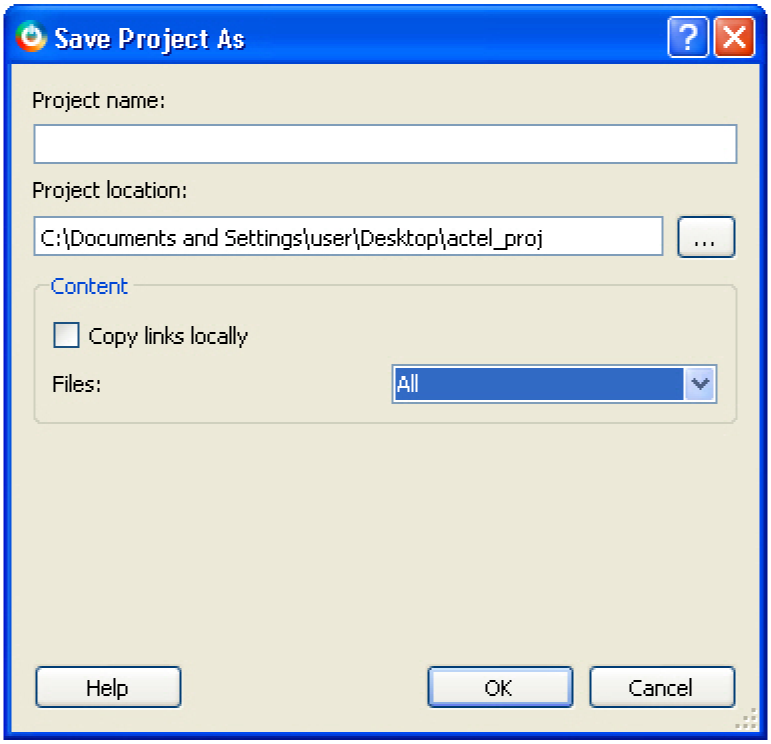
For example, in SmartFusion you can go directly from MSS configuration to Program Device by just using the .EFC file. For users who are not using any of the FPGA fabric, this is useful because you can skip the entire FPGA flow. In that instance you can select Run MSS Configurator > Program Device.

- **Organize Input Files** - Enables you to customize which project files are used by the tool.
- **Import Files** - Shortcut to import files that are relevant to that tool. For example, the relevant files for the Compile tool are PDC and SDC files, so the dialog is pre-filtered to only allow importing of those types
- **Edit Profile** - Shortcut to open the Tool Profiles dialog box.
- **View Report** - Opens the report of that tool in the Reports view.
- **Configure Options**- - Opens the Libero SoC tool options specific to that tool.

Save Project As Dialog Box

The Save Project As dialog box enables you to save your entire project with a new name and location.

Enter the name and location for your modified project and click OK to continue.



Project name:

Project location: C:\Documents and Settings\user\Desktop\actel_proj

Content

□ Copy links locally

Files: All

Figure 97 · Save Project As Dialog Box

Project Name

Type the project name for your modified project.

Project Location

Accept the default location or **Browse** to the new location where you can save and store your project. All files for your project are saved in this directory.

Content

Copy Links locally - Select this checkbox to copy the links from your current project into your new project. If you do not select this checkbox, the links will not be copied and you must add them manually.

Files

- All - Includes all your project and source files in your new project.
- Project files only - Copies only your project files into your new project (only the files listed in the Files window). Note that pin mapping files, PDC files created in MVN and other generated files are not preserved when you select Project files only.
- Source files only - Copies only your source files into your new project; for example, simulation files are not preserved.
- None - Saves a new empty project.

To access this dialog, from the **Project** menu, choose **Save Project As.**

Saving Files

Files and projects are saved when you close them.

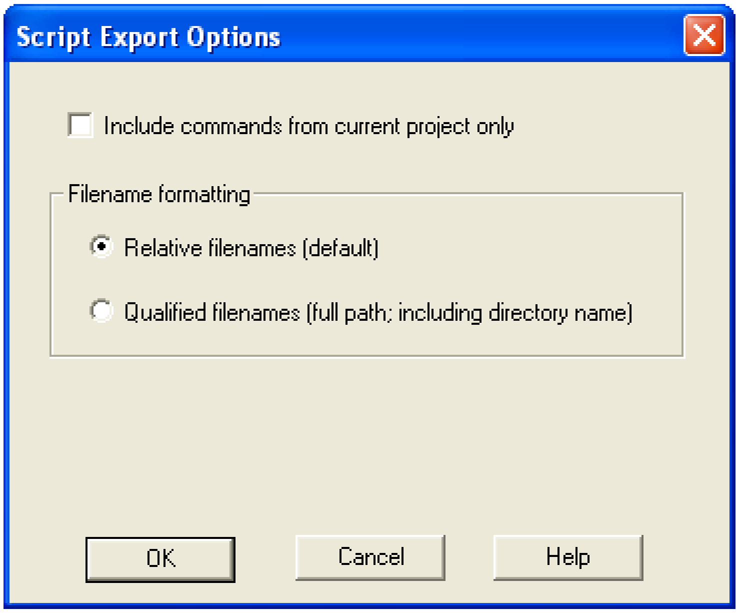
To save an active file:

- From the **Project** menu, choose **Save** or **Save As**.

- Click the **Save** button in the toolbar.

Script Export Options Dialog Box

If you export a Tcl script in the Project Manager, the Script Export Options dialog box appears.



Script Export Options

□ Include commands from current project only

Filename formatting

● Relative filenames (default)

○ Qualified filenames (full path; including directory name)

Figure 98 · Script Export Options Dialog Box

Include commands from current project only - Select this option if you want to include all the commands from your current project.

Filename Formatting - Choose **Relative filenames** if you do not intend to move the Tcl script from the saved location, or **Qualified filenames** if you plan to move the Tcl script on your machine.

Search in Libero SoC

Search options vary depending on your search type.

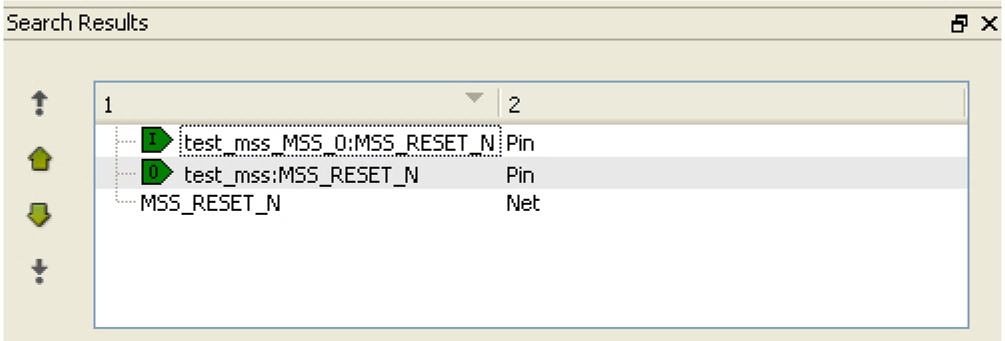
To find a file:

1. Use CTRL + F to open the Search window.
2. Enter the name or part of name of the object you wish to find in the Find field. '*' indicates a wildcard, and [*-*] indicates a range, such as if you search for a1, a2, ... a5 with the string a[1-5].
3. Set the Options for your search (see below for list); options vary depending on your search type.
4. Click **Find All** (or **Next** if searching Text).

   Searching an open text file, Log window or Reports highlights search results in the file itself.
   All other results appear in the Search Results window (as shown in the figure below).

Match case: Select to search for case-sensitive occurrences of a word or phrase. This limits the search so it only locates text that matches the upper- and lowercase characters you enter.

Match whole word: Select to match the whole word only.



1	2
test_mss_MSS_0:MSS_RESET_N	Pin
test_mss:MSS_RESET_N	Pin
MSS_RESET_N	Net

Figure 99 · Search Results

Current Open SmartDesign

Searches your open SmartDesign, returns results in the Search window.

Type: Choose Instance, Net or Pin to narrow your search.

Query: Query options vary according to Type.

Type	Query Option	Function
Instance	Get Pins	Search restricted to all pins
	Get Nets	Search restricted to all nets
	Get Unconnected Pins	Search restricted to all unconnected pins
Net	Get Instances	Searches all instances
	Get Pins	Search restricted to all pins
Pin	Get Connected Pins	Search restricted to all connected pins
	Get Associated Net	Search restricted to associated nets
	Get All Unconnected Pins	Search restricted to all unconnected pins

Current Open Text Editor

Searches the open text file. If you have more than one text file open you must place the cursor in it and click CTRL + F to search it.

Find All: Highlights all finds in the text file.

Next: Proceed to next instance of found text.

Previous: Proceed to previous instance of found text.

Replace with: Replaces the text you searched with the contents of the field.

Replace: Replaces a single instance.

Replace All: Replaces all instances of the found text with the contents of the field.

Design Hierarchy

Searches your Design Hierarchy; results appear in the Search window.

Find All: Displays all finds in the Search window.

Stimulus Hierarchy

Searches your Stimulus Hierarchy; results appear in the Search window.

Find All: Displays all finds in the Search window.

Log Window

Searches your Log window; results are highlighted in the Log window - they do not appear in the Search Results window.

Find All: Highlights all finds in the Log window.

Next: Proceed to next instance of found text.

Previous: Proceed to previous instance of found text.

Reports

Searches your Reports; returns results in the Reports window.

Find All: Highlights all finds in the Reports window.

Next: Proceed to next instance of found text.

Previous: Proceed to previous instance of found text.

Files

Searches your local project file names for the text in the Search field; returns results in the Search window.

Find All: Lists all search results in the Search window.

Files on disk

Searches the files' content in the specified directory and subdirectories for the text in the Search field; returns results in the Search window.

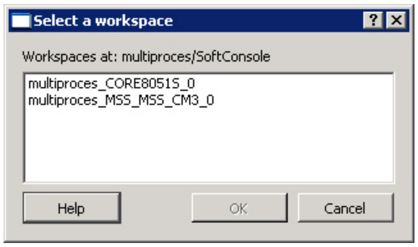
Find All: Lists all finds in the Search window.

File type: Select a file type to limit your search to specific file extensions, or choose *.* to search all file types.

Select a Workspace Dialog Box

This dialog box enables you to choose which processor you want to open when you have two or more processors in your design.

It is only available if you have two or more processors and double-click Develop Firmware > Write Application Code.



Select a workspace

Workspaces at: multiproces/SoftConsole

multiproces_CORE8051S_0
multiproces_MSS_MSS_CM3_0

Figure 100 · Select a Workspace Dialog Box

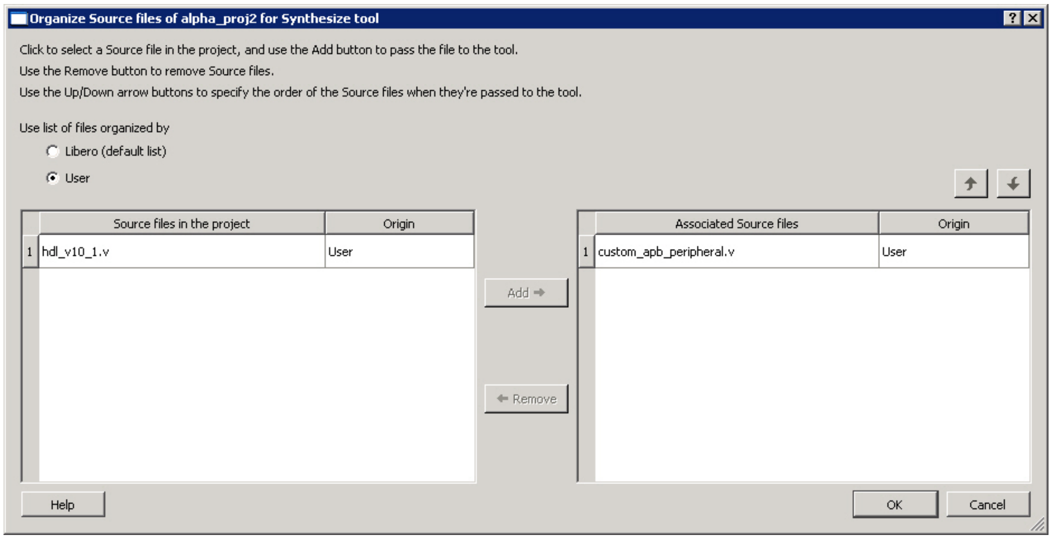
Organize Source Files

The Organize Source Files dialog box enables you to set the source file order in the Libero SoC.

Click the **Use list of files organized by User** radio button to Add/Remove source files for the selected tool.

To specify the file order:

1. In the Design Flow window under Implement Design, right-click **Synthesize** and choose **Organize Input Files > Organize Source Files**. The Organize Source Files dialog box appears.
2. Click the **Use list of files organized by User** radio button to Add/Remove source files for the selected tool.
3. Select a file and click the Add or Remove buttons as necessary. Use the Up and Down arrows to change the order of the Associated Source files.
4. Click **OK**.



Organize Source files of alpha_proj2 for Synthesize tool

Click to select a Source file in the project, and use the Add button to pass the file to the tool.
Use the Remove button to remove Source files.
Use the Up/Down arrow buttons to specify the order of the Source files when they're passed to the tool.

Use list of files organized by

○ Libero (default list)

● User

	Source files in the project	Origin
1	hdl_v10_1.v	User

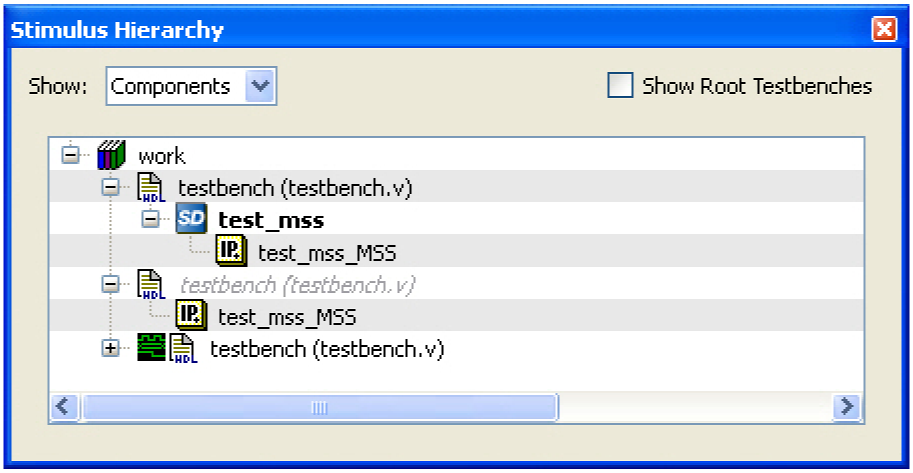
	Associated Source files	Origin
1	custom_apb_peripheral.v	User

Figure 101 · Organize Source Files Dialog Box

Stimulus Hierarchy

To view the Stimulus Hierarchy, from the **View** menu choose **Windows > Stimulus Hierarchy**.

The Stimulus Hierarchy tab displays a hierarchical representation of the stimulus and simulation files in the project. The software continuously analyzes and updates files and content. The tab (see figure below) displays the structure of the modules and component stimulus files as they relate to each other.



Stimulus Hierarchy

Show: Components

□ Show Root Testbenches

```
work
  testbench (testbench.v)
    test_mss
      test_mss_MSS
  testbench (testbench.v)
    test_mss_MSS
  testbench (testbench.v)
```

Figure 102 · Stimulus Hierarchy Dialog Box

Expand the hierarchy to view stimulus and simulation files. Right-click an individual component and choose **Show Module** to view the module for only that component.

Seelct **Components** or **Modules** from the **Show** drop-down list to change the display mode. The Components view displays the stimulus hierarchy; the modules view displays HDL modules and stimulus files.

The file name (the file that defines the module or component) appears in parentheses.

Click **Show Root Testbenches** to view only the root-level testbenches in your design.

Right-click and choose **Properties**; the Properties dialog box displays the pathname, created date, and last modified date.

All integrated source editors are linked with the SoC software; if you modify a stimulus file the Stimulus Hierarchy automatically updates to reflect the change.

To open a stimulus file:

Double-click a stimulus file to open it in the HDL text editor.

Right-click and choose **Delete from Project** to delete the file from the project. Right-click and choose **Delete from Disk and Project** to remove the file from your disk.

Icons in the Hierarchy indicate the type of component and the state, as shown in the table below.

Table 14 · Design Hierarchy Icons

Icon	Description
SD	SmartDesign component
i SD	SmartDesign component with HDL netlist not generated
SD TB	SmartDesign testbench
i SD TB	SmartDesign testbench with HDL netlist not generated
IP	IP core was instantiated into SmartDesign but the HDL netlist has not been generated
HDL	HDL netlist

Text Editor

You can use the Libero IDE HDL text editor or another text editor.

To set your text editor preferences:

1. From the **Project** menu, choose **Preferences**.

2. Click **Text Editor**.

2. Set your options and click **OK**.

Libero SoC text editor options:

- **Use Libero text editor**: Select to use the Libero HDL text editor.
- **Replace tab with spaces**: Enter the number of spaces you want entered when using the tab key.
- **Open programming/debugging files as read-only:**Select to specify read-only permission to .stp and .prb files.

User defined text editor

- **User defined text editor**: Deselect Use Libero text editor to activate this area. Enter the location of the the EXE for your alternative text editor.
- **Additional parameters:** Use to specify other settings to pass to the text editor. Typically, it is not necessary to modify this field.

User Template Location - Sets the path where your user templates are exported

Tool Profiles Dialog Box

The Tool Profiles dialog box enables you to add, edit, or delete your project tool profiles.

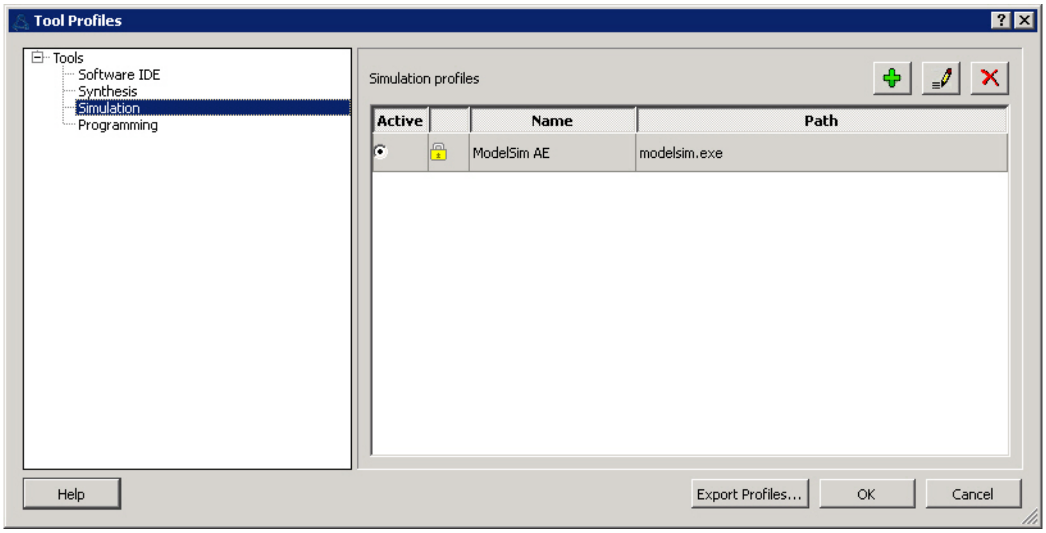
Each Libero SoC project can have a different profile, enabling you to integrate different tools with different projects.

To set or change your tool profile:

1. From the **File** menu, choose **Tool Profiles**. Select the type of tool you wish to add.

- **To add a tool**: Select the tool type and click the **Add** button . Fill out the tool profile and click **OK**.
- **To change a tool profile**: After selecting the tool, click the **Edit** button to select another tool, change the tool name, or change the tool location.
- **To remove a tool from the project**:After selecting a tool, click the **Remove** button.

2. When you are done, click **OK**.



Tool Profiles

- Tools
  - Software IDE
  - Synthesis
  - Simulation
  - Programming

Simulation profiles

Active		Name	Path
●	🔒	ModelSim AE	modelsim.exe

Figure 103 · Libero SoC Tool Profiles Dialog Box

Tools Menu - Libero SoC

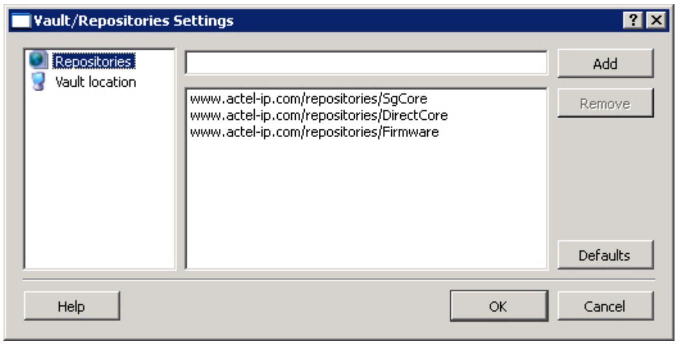
Command	Function
SmartDesign	Opens the Create New SmartDesign dialog box. Enter a filename and click OK to open SmartDesign.
HDL	Opens the Create a new HDL file dialog box. Enter a filename and click OK to open the editor.
ViewDraw	Opens the New file dialog box and defaults to Schematic. Enter a filename and click OK to open ViewDraw.
Synthesis	Starts synthesis
Simulation	Starts the simulation software and opens any existing simulation files in your project
FlashPro	Starts the FlashPro programming tool
Identify Debugger	Opens the Identify Debugger (from Synplify)

Command	Function
Write Application Code	Enables you to use a third-party IDE tool, such as Keil or IAR.

Vault/Repositories Settings Dialog Box

The Vault/Repositories Settings dialog box enables you to add, remove, or reset your repositories to default settings.

Use Vault location to specify a new location for your local vault.



www.actel-ip.com/repositories/SgCore
www.actel-ip.com/repositories/DirectCore
www.actel-ip.com/repositories/Firmware

Figure 104 · Vault/Repositories Settings Dialog Box

Videos - Libero SoC

There are short videos available that explain a variety of elements in Libero SoC. The maximum video length is 60 seconds, unless otherwise noted. See the SoC website for a complete list of the latest video content, as well as tutorials and online training.

Video Links

SoC Work Area Description - The SoC Work Window displays the HDL Editor, Report view, and SmartDesign Canvas.

Design Hierarchy Tab and Files Tab - Introduces the Design Hierarchy and Files tabs in the SoC GUI.

Design Flow Tab and Catalog - Introduces the Design Flow tab and the Catalog.

AutoConnect in SmartDesign - Demonstrates the Autoconnect feature in SmartDesign.

Connection Mode in SmartDesign - Demonstrates the manual Connection mode feature in SmartDesign.

View Design Datasheet/Report

The Design Datasheet/Report lists all the reports available for your design.

Reports are added automatically when you move through design development. For example, Timing reports are added when you run timing analysis on your design. The reports are updated each time you run timing analysis.

If a report is not listed you may have to open it manually. For example, you must double-click **Export IBIS Model** to display the IBIS Model report in the Design Datasheet.

You can view the following reports from here:

- Analyze Timing - Lists the following delay reports:

  - Timing violations report - Flat Slack report provides information about constraint violations.
  - Timing Report - Displays the timing information organized by clock domain.
- Compile - Summarizes your compile parameters and lists any related warnings, errors, PDC commands, device utilization and net information.
- Synthesize - Lists the following synthesis reports:
  - synplify.log - Outputs the Synplify log file output; identical to log file content in Synplify Pro AE if you run synthesis manually.
  - datasheet.srr - Lists the Pin Description, DC Electrical Characteristics, and AC electrical characteristics.
  - run_options.txt - Lists all the run options organized by category: project files; implementation; device options; compile/mapping options; mapper options.
- Export Pin Report - Lists the pins in your device sorted by I/O signal name and by package number.
- Place-and-Route - Lists the following reports:
  - Place-and-Route - Lists Compile and netlist information.
  - Global Net and Global Usage- Contains information about the net(s) that are assigned or routed using Global or LocalClock resources
  - I/O bank reports - Provides information on the I/O functionality, I/O technologies, I/O banks and I/O voltages.
- Export IBIS Model - Exports the IBIS model report, which provides a standard file format for recording parameters like driver output impedance, rise/fall time, and input loading, which may then be used by any software application.
- Programming - Lists the programming information for your design.

View Menu - Libero SoC

Command	Sub-menu	Shortcut	Function
Windows >	Catalog		Shows/hides the Catalog
	Cores		Shows/hides the list of cores used in your design
	Design Flow		Shows/hides the Design Flow window
	Design Hierarchy		Shows/hides the Design Hierarchy
	Files		Shows/hides the Files window
	HDL Templates		Shows/hides the HDL Templates window
	Log		Shows/hides the Log window
	Search		Shows/hides Search results

Command	Sub-menu	Shortcut	Function
	Results		
	Stimulus Hierarchy		Shows/hides the Stimulus Hierarchy
Start Page			Displays the Welcome to Libero SoC page; the page includes links to help and other pages that may be helpful for new users.
Refresh Design Hierarchy		F5	Updates the Hierarchy tab. Useful if you add files to the project and the software does not show them in the Hierarchy.
Maximize Work Area		CTRL+W	Hides the Catalog, Log Window, and Design Explorer windows (if open) and expands the selected tab in the Project Flow or SmartDesign work area.
Reset Layout			Returns the Libero SoC window layout to default.

VHDL Library - Add, Remove, or Rename

Libero SoC enables you to manage your VHDL libraries from within the Project Manager.

From the File menu, select **VHDL Library** and **Add**, **Rename**, or **Remove** to update your library.

When you add a library it appears in your Hierarchy.

Product Support

The Microsemi SoC Products Group backs its products with various support services including a Customer Technical Support Center and Non-Technical Customer Service. This appendix contains information about contacting the SoC Products Group and using these support services.

Contacting the Customer Technical Support Center

Microsemi staffs its Customer Technical Support Center with highly skilled engineers who can help answer your hardware, software, and design questions. The Customer Technical Support Center spends a great deal of time creating application notes and answers to FAQs. So, before you contact us, please visit our online resources. It is very likely we have already answered your questions.

Technical Support

Microsemi customers can receive technical support on Microsemi SoC products by calling Technical Support Hotline anytime Monday through Friday. Customers also have the option to interactively submit and track cases online at My Cases or submit questions through email anytime during the week.

Web: www.actel.com/mycases

Phone (North America): 1.800.262.1060

Phone (International): +1 650.318.4460

Email: soc_tech@microsemi.com

ITAR Technical Support

Microsemi customers can receive ITAR technical support on Microsemi SoC products by calling ITAR Technical Support Hotline: Monday through Friday, from 9 AM to 6 PM Pacific Time. Customers also have the option to interactively submit and track cases online at My Cases or submit questions through email anytime during the week.

Web: www.actel.com/mycases

Phone (North America): 1.888.988.ITAR

Phone (International): +1 650.318.4900

Email: soc_tech_itar@microsemi.com

Non-Technical Customer Service

Contact Customer Service for non-technical product support, such as product pricing, product upgrades, update information, order status, and authorization.

Microsemi’s customer service representatives are available Monday through Friday, from 8 AM to 5 PM Pacific Time, to answer non-technical questions.

Phone: +1 650.318.2470

Microsemi Corporate Headquarters
One Enterprise Drive, Aliso Viejo CA 92656
Within the USA: (800) 713-4113
Outside the USA: (949) 221-7100
Fax: (949) 756-0308 · www.microsemi.com