

Introduction [\(Ask a Question\)](#)

Libero® System-on-Chip (SoC) software provides a fully integrated Field Programmable Gate Array (FPGA) design environment. However, a few users might want to use third-party synthesis and simulation tools outside the Libero SoC environment. Libero is now integrated into the FPGA design environment. It is recommended to use Libero SoC to manage the entire FPGA design flow.

This user guide describes the Custom Flow, a process to integrate Libero as a part of the larger FPGA design flow.

Supported Device Families

The following table lists the device families that Libero SoC supports. However, some information in this guide might only apply to a specific family of devices. In this case, such information is clearly identified.

Table 1. Device Families Supported by Libero® SoC

Device Family	Description
SmartFusion® 2	SmartFusion 2 device addresses the fundamental requirements for advanced security, high reliability, and low power in critical industrial, military, aviation, communications, and medical applications.
IGLOO® 2	IGLOO 2 device is a low-power mixed-signal programmable solution.
RTG4™	RTG4 is Microchip's family of radiation-tolerant FPGAs.

Table of Contents

Introduction.....	1
1. Overview.....	3
1.1. Component Life Cycle.....	4
1.2. Libero SoC Project Creation.....	5
1.3. Custom Flow.....	5
2. Component Configuration.....	9
2.1. Component Configuration Using Libero.....	9
2.2. Component Manifests.....	11
2.3. Interpreting Manifest Files.....	12
3. Constraint Generation.....	14
4. Synthesizing Your Design.....	15
5. Simulating Your Design.....	16
6. Implementing Your Design.....	20
7. Building Your Firmware Project.....	24
8. Appendix A—Libero-Generated Hardware Configuration Files.....	26
9. Appendix B—Sample SDC and PDC Constraints.....	27
9.1. SDC Timing Constraints.....	27
9.2. PDC Physical Design Constraints.....	28
10. Appendix C—Importing Simulation Libraries into Simulation Environment.....	29
11. Appendix D—Derive Constraints.....	30
11.1. Derive Constraints Tcl Commands.....	30
12. Revision History.....	39
Microchip FPGA Support.....	40
Microchip Information.....	41
Trademarks.....	41
Legal Notice.....	41
Microchip Devices Code Protection Feature.....	41


1. Overview [\(Ask a Question\)](#)

While Libero SoC provides a fully-integrated end-to-end design environment to develop SoC and FPGA designs, it also provides the flexibility to run synthesis and simulation with third-party tools outside the Libero SoC environment. However, some design steps must remain within the Libero SoC environment.

The following table lists the major steps in the FPGA design flow and indicates the steps for which Libero SoC must be used.

Table 1-1. FPGA Design Flow

Design Flow Step	Must Use Libero	Description
Design Entry: HDL	No	Use third-party HDL editor/checker tool outside Libero® SoC if desired.
Design Entry: Configurators	Yes	Create first Libero project for IP catalog core component generation.
Design Entry: System Builder (SmartFusion® 2 and IGLOO® 2 devices only)	Yes	Stay in first Libero project
Automatic PDC or SDC constraint generation	No	Derived constraints need all HDL files and a derive_constraints utility when performed outside of Libero SoC, as described in Appendix D—Derive Constraints .
Simulation	No	Use third-party tool outside Libero SoC, if desired. Requires download of pre-compiled simulation libraries for target device, target simulator, and target Libero version used for backend implementation.
Synthesis	No	Use third-party tool outside Libero SoC if desired.
Design Implementation: Manage Constraints, Compile Netlist, Place-and-Route (see Figure 1-1)	Yes	Create second Libero project for the backend implementation.
Timing and Power Verification	Yes	Stay in second Libero project
Programming File Generation	Yes	Stay in second Libero project
Firmware Generation (SmartFusion 2 only)	Yes	Stay in second Libero project
Firmware Debug (SmartFusion 2 only)	No	Use third-party tool outside Libero SoC if desired.

 **Important:** You must download precompiled libraries listed under the *Compiled Simulation Libraries for SmartFusion® 2, IGLOO® 2, and RTG4™* section on the [Libero SoC Documentation](#) page to use a third-party simulator.

In a CPLD or pure Fabric FPGA flow, enter your design using HDL or schematic entry and pass that directly to the synthesis tools. The flow is still supported. However, SmartFusion® 2, IGLOO® 2, and RTG4™ FPGAs have significant proprietary hard IP blocks requiring the use of configuration cores (SgCores) from the Libero SoC IP catalog. Special handling is required for the following blocks that comprise SoC functionality.

- SmartFusion 2 and IGLOO 2
 - SmartFusion 2 MSS (includes MDDR and eNVM)
 - IGLOO 2 HPMS (includes MDDR and eNVM)
 - System Builder
 - FDDR

- SerDes
- Oscillator
- CCC
- RAMs (TPSRAM, DPSRAM, URAM)
- TAMPER, and so on.
- RTG4
 - uPROM
 - SerDes
 - SerDes INIT
 - FDDR
 - FDDR INIT
 - RCOSC macro
 - CCC
 - RAMs (TPSRAM, DPSRAM, URAM), and so on.

In addition to the preceding listed SgCores, there are many DirectCore soft IPs available for various device families in the Libero SoC Catalog that use the FPGA fabric resources.

For design entry, if you use any one of the preceding components, you must use Libero SoC for part of the design entry ([Component Configuration](#)), but you can continue the rest of your Design Entry (HDL entry, and so on) outside of Libero. To manage the FPGA design flow outside of Libero, follow the steps provided in the rest of this guide.

1.1. Component Life Cycle [\(Ask a Question\)](#)

The following steps describe the life cycle of an SoC component and provide instructions on handle data:

1. Generate the component using its configurator in Libero SoC. This generates the following types of data:
 - a. HDL files
 - b. Memory files
 - c. Stimulus and Simulation files
 - d. Component metadata such as component file manifest text file, register configuration files (*`.reg`) for MDDR/FDDR/SerDes and *`cfg` file for eNVM (SmartFusion 2 and IGLOO 2), uPROM (RTG4)
 - e. Firmware drivers (*`.h`) files
 - f. Component SDC file
2. For HDL files, instantiate and integrate them in the rest of the HDL design using the external design entry tool or process.
3. Supply memory files and stimulus files to your simulation tool.
4. Supply firmware drivers to your firmware project.
5. Supply Component SDC file to Derive Constraint tool for Constraint Generation. For more information, see [Appendix D—Derive Constraints](#) for more details.
6. You must create a second Libero project, where you import the post-Synthesis netlist and your component metadata (data files about the design components, such as register configuration files and initialization files), thus completing the connection between what you generated and what you program.

1.2. Libero SoC Project Creation [\(Ask a Question\)](#)

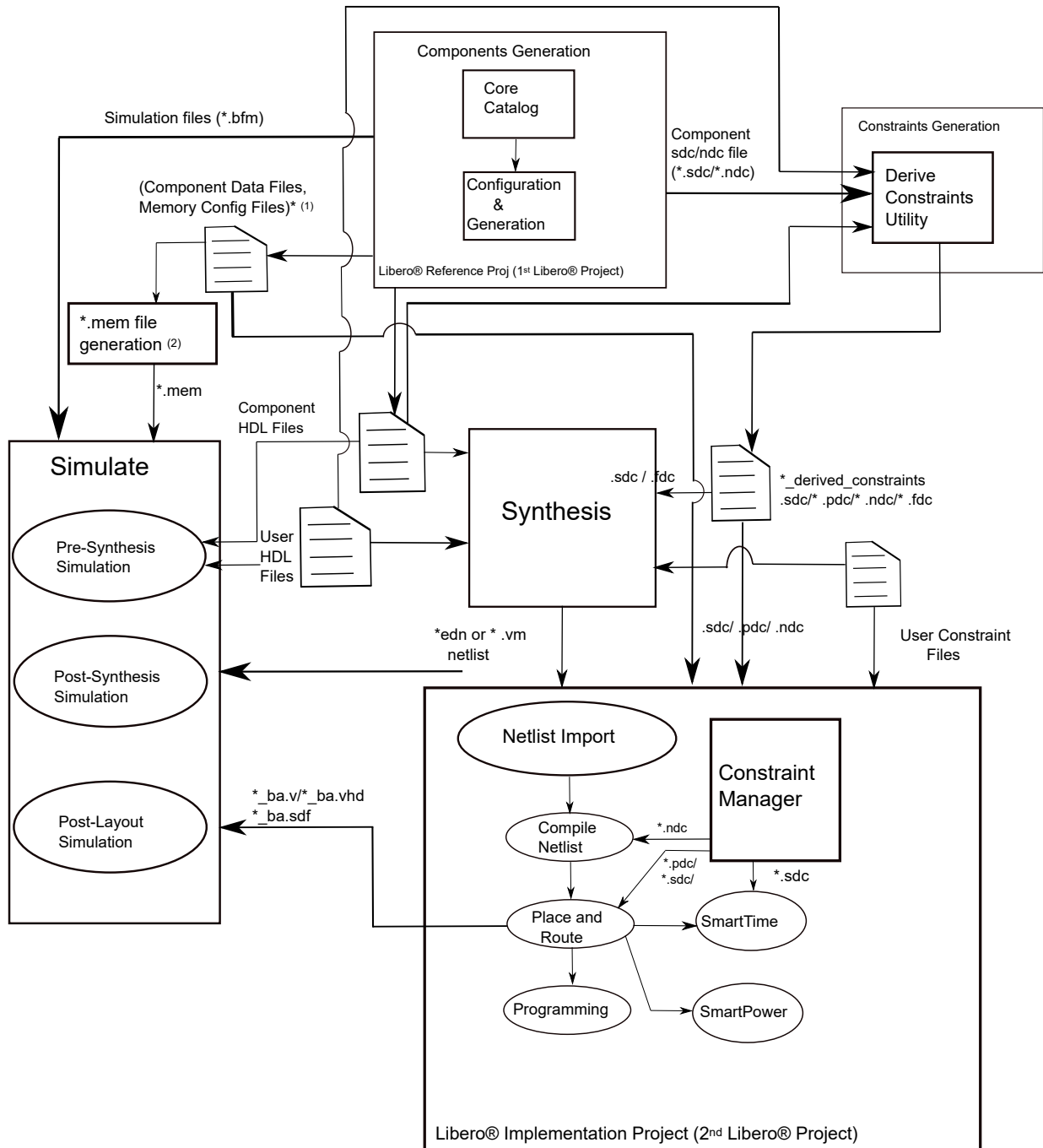
Some design steps must be run inside the Libero SoC environment ([Table 1-1](#)). For these steps to run, you must create two Libero SoC projects. The first project is used for design component configuration and generation, and the second project is for the physical implementation of the top-level design.

- The enhanced constraint flow is used for both the first project and the second (implementation) project because the enhanced constraint flow offers the constraint manager to better manage all design constraints (SDC Timing, IO PDC, Floorplanning PDC, and Synthesis NDC constraints). The creation, import, and editing of constraints and their association with individual design tools are controlled in one single management tool—the Constraint Manager.
- The enhanced constraint flow generates automatic SDC and PDC constraints for common cores such as the CCC, OSC, SerDes, and so on. For RTG4 designs using the RTG4FCCCECALIB core, the automatic constraint generation (Derive Constraints) generates netlist constraint files (NDC). In addition, SDC files are generated for association with synthesize or compile netlist design flow steps. The SDC, PDC, or NDC constraints of these cores are set from the top of the design hierarchy with the full hierarchical path given. You do not need to traverse from the top of the design hierarchy to set a constraint on these IP cores, nor do you need to worry about the syntax of the SDC, PDC, or NDC constraints such as hierarchy and pin separators, design object names, and so on.
- The automatically generated constraints, when applied, increase the chance of timing closure with less effort and fewer design iterations.

1.3. Custom Flow [\(Ask a Question\)](#)

The following figure shows how Libero SoC is integrated as a part of the larger FPGA design flow with the third-party synthesis and simulation tools outside the Libero SoC environment. The following figure shows various steps involved in the flow, starting from design creation and stitching all the way to programming the device. The figure also shows the data exchange (inputs and outputs) that must occur at each design flow step.

Figure 1-1. Custom Flow Overview





Tip:

1. Component Data Files and Memory Config Files:
 - a. Component Data Files and Memory Config Files:
 - i. SmartFusion 2/IGLOO 2: *.reg files for MDDR/FDDR/SerDes blocks.
 - b. Memory Config Files:
 - i. SmartFusion 2/IGLOO 2: ENV.M.cfg.
 - ii. RTG4: UPROM.cfg.
2. *.mem file generation for Simulation for different families:
 - a. SmartFusion 2: pa4mssenvmgen.exe takes ENV.M.cfg as input and generates ENV_init.mem.
 - b. IGLOO 2: pa4mssenvmgen.exe takes *.reg files for MDDR/FDDR/SerDes blocks and ENV.M.cfg as input and generates ENV_init.mem.
 - c. RTG4: pa4rtupromgen.exe takes UPROM.cfg as input and generates UPROM.mem.

The following are the steps in the custom flow:

1. Component configuration and generation:
 - a. Create a first Libero project (to serve as a Reference Project).
 - b. Select the Core from the Catalog. Double-click the core to give it a component name and configure the component.



Important: For SmartFusion 2 and IGLOO 2 System Builder and SmartFusion 2 MSS blocks, generate the component in the SmartDesign canvas after configuration of the SmartFusion 2 MSS block and System Builder block.

This automatically exports component data and files. A [Component Manifests](#) is also generated. See [Component Manifests](#) for details. For more details, see [Component Configuration](#).

2. Complete your RTL design outside of the Libero:
 - a. Instantiate the component HDL files.
 - b. The location of the HDL files is listed in the [Component Manifests](#) files.
3. Generate SDC/PDC/NDC constraints for the components. Use **Derive Constraints utility** to generate the floorplanning *.pdc (only for SmartFusion 2/IGLOO 2), the timing *.sdc, and netlist constraints *.ndc (in RTG4 designs using RTG4FCCCECALIB core) files based on:
 - a. Component HDL files
 - b. Component SDC/NDC files
 - c. User HDL files

For more details, see [Appendix D—Derive Constraints](#).
4. Synthesis tool/simulation tool:
 - a. Get HDL files, stimulus files, and component data from the specific locations as noted in the [Component Manifests](#).
 - b. Synthesize and simulate the design with third-party tools outside the Libero SoC.
5. Firmware tool (SmartFusion 2 only):

- a. Get drivers from the specific locations as noted in the manifest.
- b. Edit source code to enable run time initialization for specific components, compile firmware project.
6. Create your second (Implementation) Libero Project.
7. Remove synthesis from the design flow tool chain (**Project** > **Project Settings** > **Design Flow** > clear the **Enable Synthesis** check box) if it is a netlist file.
8. Import the design source files (post-synthesis *.edn or *.vm netlist from synthesis tool):
 - For SmartFusion 2 and IGLOO 2, import post-synthesis *.edn or *.vm netlist (**File** > **Import** > **Others**).
 - Component metadata such as *.reg files for MDDR/FDDR/SerDes (SmartFusion 2 and IGLOO 2), *.cfg file for eNVM (SmartFusion 2 and IGLOO 2), and *.cfg file for µPROM (RTG4).
9. Import any Libero SoC block component files. The block files must be in the *.cxz file format. For more information on how to create a block, see [SmartFusion2, IGLOO2, and RTG4 Block Flow User's Guide](#).
10. Import the design constraints:
 - Import I/O constraint files (**Constraints Manager** > **I/OAttributes** > **Import**).
 - Import floorplanning *.pdc files (**Constraints Manager** > **Floor Planner** > **Import**). If your design contains CoreConfigP (SmartFusion 2 and IGLOO 2), import the PDC file generated with the **Derive Constraints utility** (generate SDC and PDC constraints for the components).
 - Import *.sdc timing constraint files (**Constraints Manager** > **Timing** > **Import**). Import the SDC file generated through **Derive Constraint** tool.
 - Import *.ndc constraint files (**Constraints Manager** > **NetlistAttributes** > **Import**), if any. In RTG4 designs using the RTG4FCCCECALIB core, import the NDC file generated through **Derive Constraints utility** outside of Libero.
11. Constraint file and tool association
 - In the **Constraint Manager**, associate the *.pdc files to place and route, the *.sdc files to place and route and timing verifications, and the *.ndc files to compile netlist.
12. Complete design implementation
 - Place and route, verify timing and power.
13. Validate the design
 - Validate the design on FPGA and debug as necessary using the design tools provided with the Libero SoC design suite.

2. Component Configuration [\(Ask a Question\)](#)

The first step in the custom flow is to configure your components using a Libero reference project (also called first Libero project in [Table 1-1](#)). In subsequent steps, you will use data from this reference project.


If you are using any components listed earlier, under the [Overview](#) in your design, perform the steps described in this section.

If you are not using any of the above components, then write your RTL outside of Libero and directly import it into your Synthesis and Simulation tools. Then proceed to the post-synthesis section and only import your post-synthesis *.edn or *.vm netlist into your final Libero implementation project (also called second Libero project in [Table 1-1](#)).


2.1. Component Configuration Using Libero [\(Ask a Question\)](#)

After selecting the components that must be used from the preceding list, perform the following steps:


1. Create a new Libero project (Core Configuration and Generation):
 - a. Select the Device and Family that you target your final design to.
 - b. If you use the SmartFusion 2 MSS, SmartFusion 2, or IGLOO 2 System Builder, make the appropriate selection in the **Use Design Flow** section of the New Project window.
2. If you use Smart Fusion 2 or IGLOO 2 System Builder (for IGLOO 2, you must use the System Builder to configure the HPMS, eNVM, MDDR, and FDDR):
 - a. Use the System Builder to select your components and configure your system.
 - b. Generate your system in the System Builder and promote all its ports to the top level after instantiating in a SmartDesign canvas (select all ports, right-click and choose promote to top).

 **Tip:** You need the port names to connect the rest of your design to the generated system.


- c. Instantiate and configure any CCC or SerDes blocks in the same top-level SmartDesign or in another SmartDesign component. Again, promote any ports to top.
- d. Generate any SmartDesign instances.
- e. Double-click the **Simulate** tool (Pre-Synthesis) to invoke the simulator. Exit the simulator once it is invoked—this step generates the simulation files necessary for your project.

 **Tip:** You must perform this step if you want to simulate your design outside Libero.


- f. Save your project—this is your reference project.
3. If you do not use the System Builder (SmartFusion 2 only):
 - a. If you select the SmartFusion 2 MSS in the **Use Design Flow** subsection (step [1.b](#)), the SmartFusion 2 MSS Configurator automatically opens. Otherwise, in the **Design Flow** window, double-click **Configure MSS**.
 - i. Double-click the SmartFusion 2 MSS instance to open its configurator.
 - ii. Configure the SmartFusion 2 MSS as per your requirements.

 **Tip:** If you use the eNVM or the MDDR, you must use the SmartFusion 2 MSS to configure it.


iii. Save and generate the SmartFusion 2 MSS component.

 **Tip:** If you do not use the System Builder, and you have the SmartFusion 2 MSS (using MDDR) or FDDR or SerDes blocks in your design, you must follow the next steps.

- b. Construct the Peripheral Initialization architecture in your final design. For more details, about Peripheral Initialization, see:
 - [SmartFusion2 DDR Controller and Serial High Speed Controller Initialization Methodology](#)
 - [SmartFusion2 DDR Controller and Serial High Speed Controller Standalone Initialization Methodology](#)
- c. Instantiate and configure any FDDR, CCC, or SerDes blocks in the top-level SmartDesign. It is not necessary to connect them to anything else—just promote any ports to top.
- d. Generate all the SmartDesigns built in the preceding steps.
- e. Double-click the **Simulate** tool (Pre-Synthesis) to invoke the simulator. Exit the simulator after it is invoked; this step simply generates the simulation files necessary for your project.


 **Tip:** You must perform this step if you want to simulate your design outside Libero.
For more information, see [Simulating Your Design](#).

- f. Save your project—this is your reference project.
4. If you use SmartFusion 2, and any of the SmartFusion 2 MSS peripherals (MDDR, FDDR, or SerDes), you must export your firmware project (SoftConsole/IAR/Keil) from this Libero project. For more information, see [Building Your Firmware Project](#).
5. If you use RTG4:
 - a. If you want to use the SerDes and the FDDR blocks in your design with built-in Initialization logic, configure and generate the corresponding INIT cores (NPSS_SERDES_IF_INIT, PCIE_SERDES_IF_INIT, and RTG4FDDRC_INIT) from the Libero catalog.
 - b. If you want to use the SerDes and the FDDR blocks in your design without built-in Initialization logic, configure and generate the corresponding peripheral cores from the Libero catalog.


 **Tip:** If you are using SerDes and FDDR blocks in your design, but not their corresponding INIT cores, you must construct the Peripheral Initialization architecture in your final design to initialize the RTG4 SerDes and the FDDR blocks. For further details regarding Peripheral Initialization, see:

- [RTG4™ High Speed Serial Interface Configuration User Guide](#)
- [RTG4 DDR Memory Controller with Initialization Configuration User Guide](#)

- c. Instantiate and configure any FDDR, CCC, OSC, or SerDes blocks in the top level SmartDesign. It is not necessary to connect them to anything else — just promote any ports to top.
- d. If you want to use RTG4 μ PROM, add the μ PROM block to the top level SmartDesign.
- e. Generate all the SmartDesigns built in the above steps.
- f. Double-click **Simulate** (Pre-Synthesis) to invoke the simulator. Exit the simulator after it is invoked; this step only generates the simulation files that are necessary for your project.


 **Tip:** To generate `μ PROM.mem` file (used for simulating the μ PROM contents) from the `μ PROM.cfg` file (generated by the configurator) using a stand-alone executable outside Libero, see [Simulating Your Design](#). You must perform this step if you want to simulate your design outside of Libero.

- g. Save your project—this is your reference project.

 **Tip:** You must follow the DRCs for components that you instantiate. For example, if you have multiple SerDes instances in your design, make sure that each SerDes instance is configured to select a different physical SerDes block. Refer to the user guides for the respective component DRCs for details.

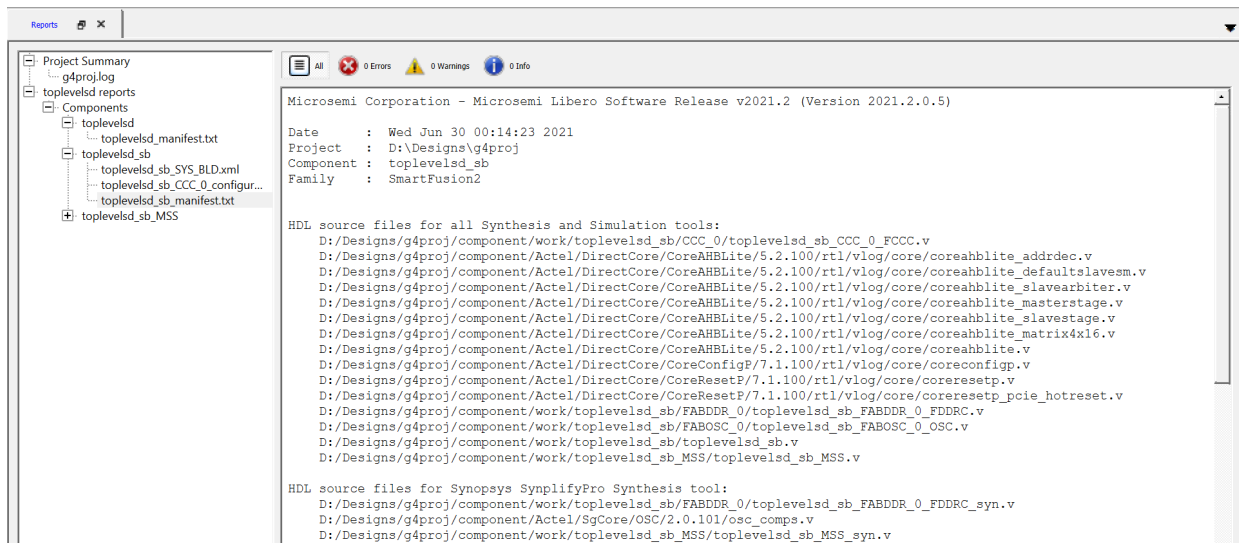
2.2. Component Manifests [\(Ask a Question\)](#)

When you generate your components, a set of files is generated for each component. The Component Manifest Report details the set of files generated and used in each subsequent step (Synthesis, Simulation, Firmware Generation, and so on). This report gives you the locations of all the generated files needed to proceed with the Custom Flow. To the component manifest in the Reports area: Click **Design > Reports** to open the reports tab. In the reports tab, you see a set of `manifest.txt` files ([Figure 1-1](#)), one for each component you generated.

 **Tip:** You must set a component or module as 'root' to see the component manifest file contents in the Reports tab.

Alternatively, access the individual manifest report files for each core component generated or SmartDesign component from `<project>/component/work/<component name>/<instance name>/<component name>_manifest.txt` or `<project>/component/work/<SmartDesign name>/<SmartDesign name>_manifest.txt`. Access the manifest file contents of each component generated from the new **Components** tab in Libero, where the file locations are mentioned with respect to the project directory.

Figure 2-1. Accessing Component Manifest Report Files from Libero Reports Tab (For SmartFusion 2 System Builder)



Focus on the following Component Manifest Reports:

- If you use SmartFusion 2 or IGLOO 2 System Builder, read the file <system builder name>_sb_manifest.txt.
- If you instantiated cores into a SmartDesign, read the file <smartdesign_name>_manifest.txt.
- If you created components for cores, read the <core_component_name>_manifest.txt.

You must use all [Component Manifests](#) Reports that apply to your design. For example, if your project has a SmartDesign with one or more core components instantiated in it and you intend to use them all in your final design, then you must select files listed in the [Component Manifests](#) Reports of all those components for use in your design flow.

2.3. Interpreting Manifest Files [\(Ask a Question\)](#)

When you open a component manifest file, you see paths to files in your Libero project and pointers on where in the design flow to use them. You might see the following types of files in a manifest file based on the device family you are targeting:

- HDL source files for all Synthesis and Simulation tools
- HDL source files for Synopsys SynplifyPro Synthesis tool
- Stimulus files for all Simulation tools
- Configuration files to be used for all Simulation tools
- Firmware files for all Software IDE tools
- Configuration files to be used for Programming
- Configuration files to be used for Power Analysis

The following is the Component Manifest (example of a SmartFusion 2 System Builder component).

```
HDL source files for all Synthesis and Simulation tools:
D:/Designs/g4proj/component/work/toplevelsd_sb/CCC_0/toplevelsd_sb_CCC_0_FCCC.v
D:/Designs/g4proj/component/Actel/DirectCore/CoreAHBLite/5.2.100/rtl/vlog/core/
coreahblite_addrdec.v
D:/Designs/g4proj/component/Actel/DirectCore/CoreAHBLite/5.2.100/rtl/vlog/core/
coreahblite_defaultsavesm.v
D:/Designs/g4proj/component/Actel/DirectCore/CoreAHBLite/5.2.100/rtl/vlog/core/
coreahblite_slavearbitr.v
```

```

D:/Designs/g4proj/component/Actel/DirectCore/CoreAHLite/5.2.100/rtl/vlog/core/
coreahblite_masterstage.v
D:/Designs/g4proj/component/Actel/DirectCore/CoreAHLite/5.2.100/rtl/vlog/core/
coreahblite_slavestage.v
D:/Designs/g4proj/component/Actel/DirectCore/CoreAHLite/5.2.100/rtl/vlog/core/
coreahblite_matrix4x16.v
D:/Designs/g4proj/component/Actel/DirectCore/CoreAHLite/5.2.100/rtl/vlog/core/
coreahblite.v
D:/Designs/g4proj/component/Actel/DirectCore/CoreConfigP/7.1.100/rtl/vlog/core/
coreconfigp.v
D:/Designs/g4proj/component/Actel/DirectCore/CoreResetP/7.1.100/rtl/vlog/core/coreresetp.v
D:/Designs/g4proj/component/Actel/DirectCore/CoreResetP/7.1.100/rtl/vlog/core/
coreresetp_pcie_hotreset.v
D:/Designs/g4proj/component/work/toplevelsd_sb/FABDDR_0/toplevelsd_sb_FABDDR_0_FDDRC.v
D:/Designs/g4proj/component/work/toplevelsd_sb/FABOSC_0/toplevelsd_sb_FABOSC_0_OSC.v
D:/Designs/g4proj/component/work/toplevelsd_sb/toplevelsd_sb.v
D:/Designs/g4proj/component/work/toplevelsd_sb_MSS/toplevelsd_sb_MSS.v

HDL source files for Synopsys SynplifyPro Synthesis tool:
D:/Designs/g4proj/component/work/toplevelsd_sb/FABDDR_0/toplevelsd_sb_FABDDR_0_FDDRC_syn.v
D:/Designs/g4proj/component/Actel/SgCore/OSC/2.0.101/osc_comps.v
D:/Designs/g4proj/component/work/toplevelsd_sb_MSS/toplevelsd_sb_MSS_syn.v

Stimulus files for all Simulation tools:
D:/Designs/g4proj/component/work/toplevelsd_sb/subsystem.bfm
D:/Designs/g4proj/component/work/toplevelsd_sb_MSS/CM3_compile_bfm.tcl
D:/Designs/g4proj/component/work/toplevelsd_sb_MSS/user.bfm
D:/Designs/g4proj/component/work/toplevelsd_sb_MSS/test.bfm
D:/Designs/g4proj/component/Actel/SmartFusion 2MSS/
MSS/1.1.500/peripheral_init.bfm

Firmware files for all Software IDE tools:
D:/Designs/g4proj/component/work/toplevelsd_sb/FABDDR_0/sys_config_fddr_define.h
D:/Designs/g4proj/component/work/toplevelsd_sb_MSS/sys_config_mss_clocks.h
D:/Designs/g4proj/component/work/toplevelsd_sb_MSS/sys_config_mddr_define.h

Configuration files to be used for all Simulation tools:
D:/Designs/g4proj/component/work/toplevelsd_sb/FABDDR_0/FDDR_init.bfm
D:/Designs/g4proj/component/work/toplevelsd_sb_MSS/MDDR_init.bfm

Configuration files to be used for Power Analysis:
D:/Designs/g4proj/component/work/toplevelsd_sb_MSS/MDDR_init.reg
D:/Designs/g4proj/component/work/toplevelsd_sb/FABDDR_0/FDDR_init.reg

```

Each type of file is a necessary downstream in your design flow. The following sections describe integration of the files from the manifest into your design flow.

3. Constraint Generation [\(Ask a Question\)](#)

When performing configuration and generation, make sure to write/generate the SDC/PDC/NDC constraint files for the design to pass them to Synthesis, Place-and-Route, and Verify Timing tools.

Use the Derive Constraints utility outside of the Libero environment to generate constraints instead of writing manually. To use the Derive Constraint utility outside of the Libero environment, you must:

- Supply user HDL, component HDL and component SDC/NDC constraint files
- Specify the top level module
- Specify the location where to generate the derived constraint files

The SDC/NDC component constraints are available under `<project>/component/work/<component name>/<instance_name>/` directory after component configuration and generation.

**Important:**


- If using RTG4FCCCECALIB core for RTG4 designs, you must supply the component NDC constraint file and specify the location to generate NDC constraints for your design.
 - If using CoreConfigP and CoreResetP for SmartFusion 2 MSS, you must specify the location to generate PDC constraints.
-

For more details on how to generate constraints for your design, see [Appendix D—Derive Constraints](#).

4. Synthesizing Your Design [\(Ask a Question\)](#)

One of the primary features of the Custom Flow is to allow you to use a third-party synthesis tool outside Libero. The custom flow supports the use of Synopsys SynplifyPro. To synthesize your project, use the following procedure:

1. Create a new project in your Synthesis tool, targeting the same device family, die, and package as the Libero project you created.
 - a. Import your own RTL files as you normally do.
 - b. Set the Synthesis output to be Structural Verilog (.vm) or edif netlist (.edn).
2. Import Component HDL files into your Synthesis project:
 - a. For each [Component Manifests](#) Report:
 - i. For each file under **HDL source files for all Synthesis and Simulation tools**, import the file into your Synthesis Project.
 - ii. Also, import all files under HDL source files for Synopsys SynplifyPro Synthesis tool.
3. Import the previously generated SDC file through the Derived Constraint tool (see [Appendix B—Sample SDC and PDC Constraints](#)) into the Synthesis tool. This constraint file constrains the synthesis tool to achieve timing closure with less effort and fewer design iterations.

 **Important:** If you plan to use the same *.sdc file to constrain Place-and-Route during the design implementation phase, you must import this *.sdc into the synthesis project. This is to make sure that there are no design object name mismatches in the synthesized netlist and the Place-and-Route constraints during the implementation phase of the design process. If you do not include this *.sdc file in the Synthesis step, the netlist generated from Synthesis may fail the Place and Route step because of design object name mismatches.

4. Import Netlist Attributes *.fdc, if any, into the Synthesis tool.
5. Run Synthesis.
6. The location of your Synthesis tool output has the *.edn or *.vm netlist file generated post Synthesis. You must import the netlist into the Libero Implementation Project to continue with the design process.

5. Simulating Your Design [\(Ask a Question\)](#)

To simulate your design outside of Libero (that is, using your own simulation environment and simulator), perform the following steps:

1. Design Files

a. Pre-Synthesis simulation:

- Import your RTL into your simulation project.
- For each [Component Manifests](#) Report.
 - Import each file under **HDL source files for all Synthesis and Simulation tools** into your simulation project.
- Compile these files as per your simulator's instructions.

b. Post-synthesis simulation:

- Import your post-synthesis *.edn or *.vm netlist (generated in [Synthesizing Your Design](#)) into your simulation project and compile it.

c. Post-layout simulation:

- First, complete implementing your design (see [Implementing Your Design](#)). Make sure that your final Libero project is in post-layout state.
- Double-click `Generate BackAnnotated Files` in the Libero Design Flow window. It generates two files:

```
<project_directory>/designer/<root>/<root>_ba.v/vhd <project_directory>/designer/  
<root>/<root>_ba.sdf
```

- Import both of these files into your simulation tool.

2. Stimulus and Configuration files:

a. For each [Component Manifests](#) Report:

- Copy all files under the `Configuration files to be used for all Simulation tools and Stimulus Files for all Simulation Tools` sections to the root directory of your Simulation project.

b. Make sure that any Tcl files in the preceding lists (in step 2.a) are executed first, before the start of simulation.

c. For SmartFusion 2 only:

- Review the `subsystem.bfm` file. Based on your usage of the MDDR, FDDR, or SerDes, make sure that the following lines are present (or absent) in the `subsystem.bfm` file — their presence indicates that the Component is used in your design. Absence indicates that the Component is not used:

```
#----- # Peripheral Initialization  
#-----  
#define USE_MDDR  
#define USE_FDDR  
#define USE_SERDESIF_0  
#define USE_SERDESIF_1  
#define USE_SERDESIF_2  
#define USE_SERDESIF_3
```

d. `eNVM_init.mem`: If you use the eNVM (SmartFusion 2 or IGLOO 2), or if you use IGLOO 2 and use MDDR, FDDR, or SerDes, you must use the `pa4mssenvmgen.exe` to generate the `ENVM_init.mem` file, regardless of whether or not you use eNVM. The `pa4mssenvmgen` executable takes all the peripheral `*init.reg` files and the `ENVM.cfg` file as inputs through a Tcl script file and outputs the `ENVM_init.mem` file required for simulations. This `ENVM_init.mem` file is required for component initialization in simulation. This file must

be copied to the simulation folder prior to the simulation run. An example showing the pa4mssenvmgen executable usage is provided in the following steps.

- e. `μPROM.mem`: If you use RTG4 `μPROM`, you must use the `pa4rtupromgen.exe` to generate the `UPROM.mem` file. The `pa4rtupromgen` executable takes the `μPROM.cfg` file as inputs through a Tcl script file and outputs the `μPROM.mem` file required for simulations. This file must be copied to the simulation folder prior to the simulation run.
3. Create a working folder and a sub-folder named `simulation` under the working folder. The `pa4mssenvmgen` and `pa4rtupromgen` executable expect the presence of the **simulation** sub folder in the working folder and the `*.tcl` script (steps 5 and 7) is placed in the **simulation** sub folder.
4. For SmartFusion 2 and IGLOO 2 devices, copy all the component `*init.reg` files and the `ENVM.cfg` file from the first Libero project (for component generation) into the working folder. Examples of component `*init.reg` files are:
 - a. `MDDR_init.reg`
 - b. `FDDR_init.reg`
 - c. `SERDESIF_0_init.reg`
 - d. `SERDESIF_1_init.reg`
 - e. `SERDESIF_2_init.reg`
 - f. `SERDESIF_3_init.reg`

For RTG4, copy the `UPROM.cfg` file from the first Libero project created for component generation (or the `UPROM.cfg` file with any modified/updated contents) into the working folder.

5. IGLOO 2 device: Paste the following commands in a `*.tcl` script and place it in the simulation folder created in step 3.

```
#Sample*.tcl for IGLOO 2 devices:

set_device -fam <family_name>
            -die <internal_die_name>
            -pkg <internal_pkg_name>
set_mDDR_reg -path <path_to_MDDR_register_file/MDDR_init.reg>
set_fDDR_reg -path <path_to_FDDR_register_file/FDDR_init.reg>
set_serdesif0_reg -path <path_to_SERDESIF_0_register_file/SERDESIF_0_init.reg>
set_serdesif1_reg -path <path_to_SERDESIF_1_register_file/SERDESIF_1_init.reg>
set_serdesif2_reg -path <path_to_SERDESIF_2_register_file/SERDESIF_2_init.reg>
set_serdesif3_reg -path <path_to_SERDESIF_3_register_file/SERDESIF_3_init.reg>
set_input_cfg -path <path_to_ENVM_configuration_file/ENVM.cfg>
set_sim_mem -path<path_to_ENVM_Initialization_File/ENVM_init.mem>
gen_sim -use_init true
```

For the proper internal name to use for the die and package, see the `*.prjx` file of the first Libero project (used for component generation).

For an IGLOO 2 device, the argument `use_init` must be set to true for the `gen_sim` command if any of the `*init.reg` files are used.

Not all `*init.reg` in the example `*.tcl` may be needed. Include the reg file paths for only those peripherals used in the design.

The `set_sim_mem` command specifies the path to the output file `ENVM_init.mem` that is generated upon execution of the script file with the `pa4mssenvmgen` executable.

6. At the command prompt or `cygwin` terminal, go to the working directory created in step 3. Execute the `pa4mssenvmgen` command with the `--script` option and pass to it the `*.tcl` script created in step 5.

For Windows®:

```
<Libero_SoC_release_installation>/designer/bin/pa4mssenvmgen.exe \
--script./simulation/<Tcl_script_name>.tcl
```

For Linux®:

```
<Libero_SoC_release_installation>/bin/pa4mssenvmgen \  
--script./simulation/<tcl_script_name>.tcl
```

7. SmartFusion 2 device: Paste the following commands in a *.tcl script and place it in the simulation folder created in step 3.

```
#Sample *.tcl for SmartFusion 2 devices  
set_device -fam <family>  
            -die <internal_die_name>  
            -pkg <internal_pkg_name>  
set_input_cfg -path <path_to_ENVM.cfg>  
set_sim_mem -path <path_to_ENVM_Initialization_File/ENVM_init.mem>  
gen_sim -use_init false
```

For the proper internal name to use for the die and package, see the *.prjx file of the first Libero project (used for component generation).

The argument use_init must be set to false for SmartFusion 2.

For SmartFusion 2 device, the set_mddr_reg, set_fddr_reg, and set_serdesif(x)_reg commands are not needed. All the peripheral register initialization information/data required to run simulations is a part of the *_init.bfm files (listed in the [Component Manifests](#) reports of each component), which must be copied from the first Libero SoC project (used for component generation) to the top level directory of your simulation project (outside of Libero SoC).

Use the set_sim_mem command to specify the path to the output file ENVM_init.mem that is generated upon execution of the script file with the pa4mssenvmgen executable.

8. At the command prompt or cygwin terminal, go to the working directory created in step 1. Execute the pa4mssenvmgen command with the --script option and pass to it the *.tcl script created in step 7.

For Windows:

```
<Libero_SoC_release_installation>/designer/bin/pa4mssenvmgen.exe \  
--script./simulation/<Tcl_script_name>.tcl
```

For Linux:

```
Libero_SoC_release_installation>/bin/pa4mssenvmgen \  
--script./simulation/<tcl_script_name>.tcl
```

9. RTG4: Paste the following commands in a *.tcl script and place it in the simulation folder created in step 3.

```
#Sample *.tcl for RTG4 devices to generate URPOM.mem file from UPROM.cfg  
set_device -fam <family>  
            -die <internal_die_name>  
            -pkg <internal_pkg_name>  
set_input_cfg -path <path_to_UPROM.cfg>  
set_sim_mem -path <path_to_UPROM_Initialization_File/UPROM.mem>  
gen_sim -use_init false
```

For the proper internal name to use for the die and package, see the *.prjx file of the first Libero project (used for component generation).

For RTG4, there is a provision to simulate the UPROM by specifying the UPROM.cfg file using the set_input_cfg command.

Use the set_sim_mem command to specify the path to the output file UPROM.mem that is generated upon execution of the script file with the pa4rtupromgen executable.

10. At the command prompt or cygwin terminal, go to the working directory created in step 3. Execute the pa4mssenvmgen command with the --script option and pass to it the *.tcl script created in step 9.

For Windows:

```
<Libero_SoC_release_installation>/designer/bin/pa4rtupromgen.exe \
--script./simulation/<Tcl_script_name>.tcl
```

For Linux:

```
<Libero_SoC_release_installation>/bin/pa4rtupromgen \
--script./simulation/<tcl_script_name>.tcl
```

11. For SmartFusion 2 and IGLOO 2 devices, after successful execution of the pa4mssenvmgen executable, check that the ENVM_init.mem file is generated in the location specified in the set_sim_mem command in the *.tcl script.
For RTG4, after successful execution of the pa4rtupromgen executable, check that the μPROM.mem file is generated in the location specified in the set_sim_mem command in the *.tcl script.
12. For SmartFusion 2 and IGLOO 2 devices, copy the generated ENVM_init.mem file into the top level simulation project to run simulation (outside of Libero SoC).
For RTG4, copy the generated μPROM.mem file into the top level simulation folder of your simulation project to run simulation (outside of Libero SoC).



Important: To simulate the functionality of SoC Components, download the pre-compiled SmartFusion 2/IGLOO 2, RTG4 simulation libraries and import them into your simulation environment as described here. For more details, see [Appendix C—Importing Simulation Libraries into Simulation Environment](#).

6. Implementing Your Design [\(Ask a Question\)](#)

After completing the Synthesis and Post-Synthesis simulation in your environment, you must use Libero again to physically implement your design, run timing and power analysis, and generate your programming file.

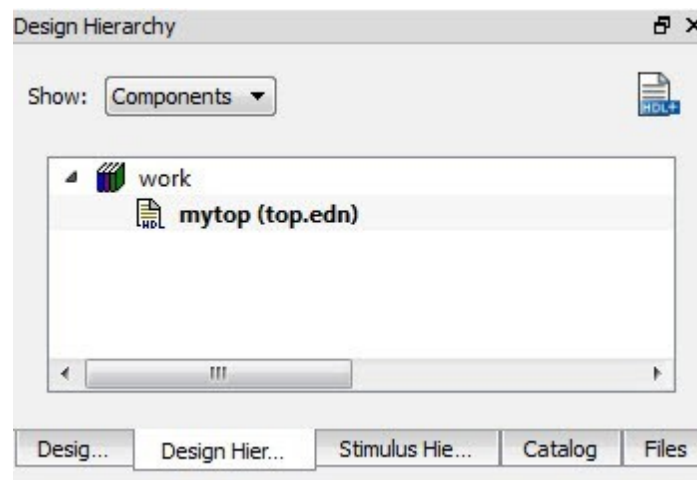
1. Create a new Libero project for the physical implementation and layout of the design. Make sure to target the same device as in the reference project you created in [Component Configuration](#).
2. After project creation, remove Synthesis from the tool chain in the **Design Flow** window (**Project > Project Settings > Design Flow > Uncheck Enable Synthesis**).
3. Import your post-synthesis *.edn or *.vm file into this project, (**File > Import > Synthesized Verilog Netlist (VM)**).



Tip: It is recommended that you create a link to this file, so that if you re-synthesize your design, Libero always uses the latest post-synthesis netlist.

- a. In the **Design Hierarchy** window, note the name of the root module (shown in the following figure).

Figure 6-1. Root Module in Design Hierarchy



4. Import the constraints into the Libero project. Use **Constraint Manager** to import *.pdc /*.sdc/* .ndc constraints.
 - a. Import I/O *.pdc constraint files (**Constraints Manager > I/O Attributes > Import**).
 - b. Import Floorplanning *.pdc constraint files (**Constraints Manager > Floorplanner > Import**). If your design contains CoreConfigP (SmartFusion 2 and IGLOO 2 only), make sure to import the PDC file generated through Derive Constraint tool.
 - c. Import *.sdc timing constraint files (**Constraints Manager > Timing > Import**). If your design has any of the cores listed in [Overview](#), make sure to import the SDC file generated through derive constraint tool.
 - d. Import *.ndc constraint files (**Constraints Manager > Netlist Attributes > Import**). If your design is an RTG4 design using RTG4FCCCECALIB core, then make sure to import the NDC file generated through derive constraint tool.
5. Associate Constraints Files to design tools

- a. Open **Constraint Manager (Manage Constraints > Open Manage Constraints View)**. Check the **Place-and-Route** and **Timing Verifications** check box next to the constraint file to establish constraint file and tool association. Associate the *.pdc constraint to Place-and-Route and the *.sdc to both **Place-and-Route** and **Timing Verifications**. Associate the *.ndc file to **Compile Netlist**.



Tip:

- If **Place and Route** fails with this *.sdc constraint file, import this same *.sdc file to synthesis and re-run synthesis.
- SmartFusion 2 and IGLOO 2 only: The derived floorplanning PDC file constrains the CoreConfigP in an optimal location for placement and improves timing performance of the design.

6. Click **Compile Netlist**, and then **Place and Route** to complete the layout step.

7. From all **Component Manifests** Reports:

- For SmartFusion 2 and IGLOO 2 devices, import all the files in the **Configuration files to be used for Programming** and **Configuration files to be used for Power Analysis** sections using the import_component_data Tcl command:

```
import_component_data -module "<name of root component>"^a \
    -fddr "<path to FDDR.reg>" \
    -mddr "<path to MDDR.reg>" \
    -serdes0 "<path to SERDESIF_0_init.reg>" \
    -serdes1 "<path to SERDESIF_1_init.reg>" \
    -serdes2 "<path to SERDESIF_2_init.reg>" \
    -serdes3 "<path to SERDESIF_3_init.reg>" \
    -envm_cfg "<path to eNVM cfg>"
```

- For RTG4, import all the files in the **Configuration files to be used for Programming** and **Configuration files to be used for Power Analysis** sections using the import_component_data Tcl command:

```
import_component_data \
    -module "<name of root component>"^a\
    -uprom_cfg "<path to uPROM cfg>"
```

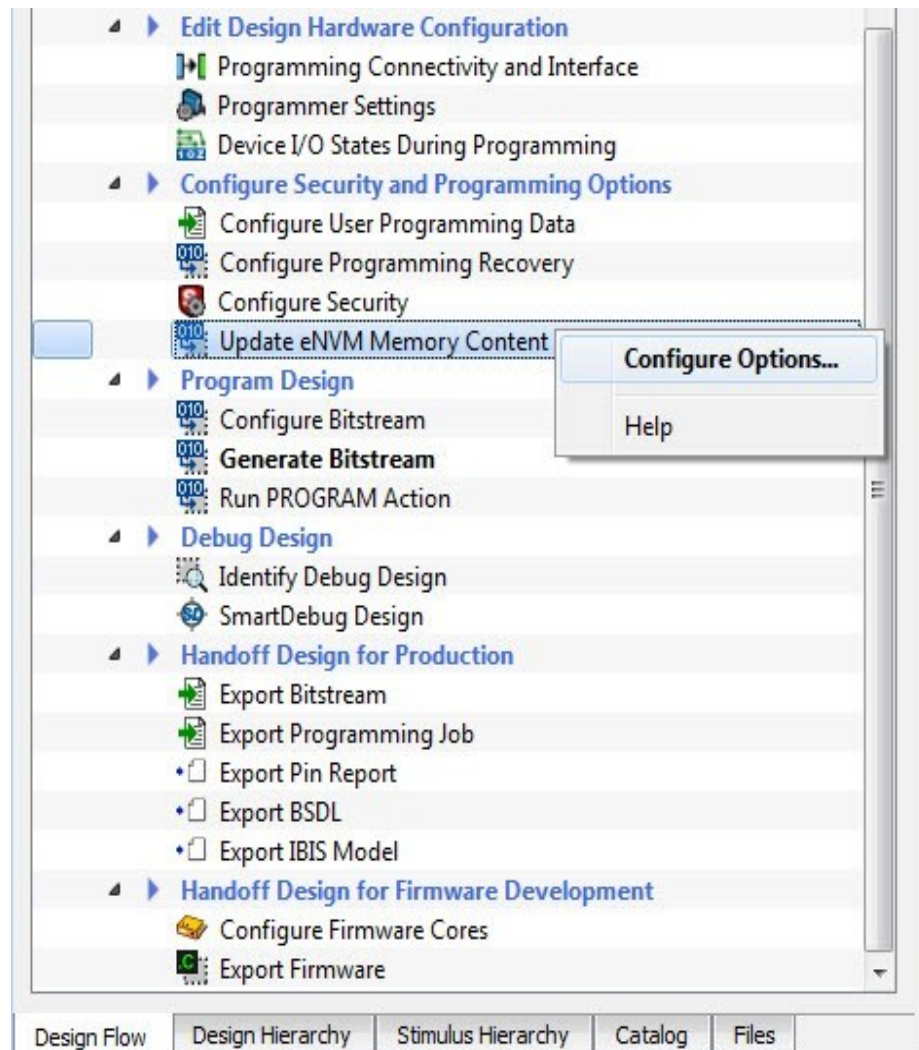


Important:

- a. "name of root component" refers to the name of the top hierarchical level of the target Libero project where files are imported. For example, see the [Figure 1-1](#), "name of root component" is "name of root component of Libero Implementation Project."
- b. All configuration files are imported with the import_component_data. Tcl commands are imported to the designer/<name of root component>/component/ folder in the Libero project directory.
- c. For SmartFusion 2 device, if you do not run SmartPower, skip importing the *.reg files, and you only need to import the eNVM.cfg file. For IGLOO 2 device, you must import all *.reg and eNVM.cfg files specified in all your relevant **Component Manifests** Reports.
- d. For RTG4, only UPROM .cfg file is imported.

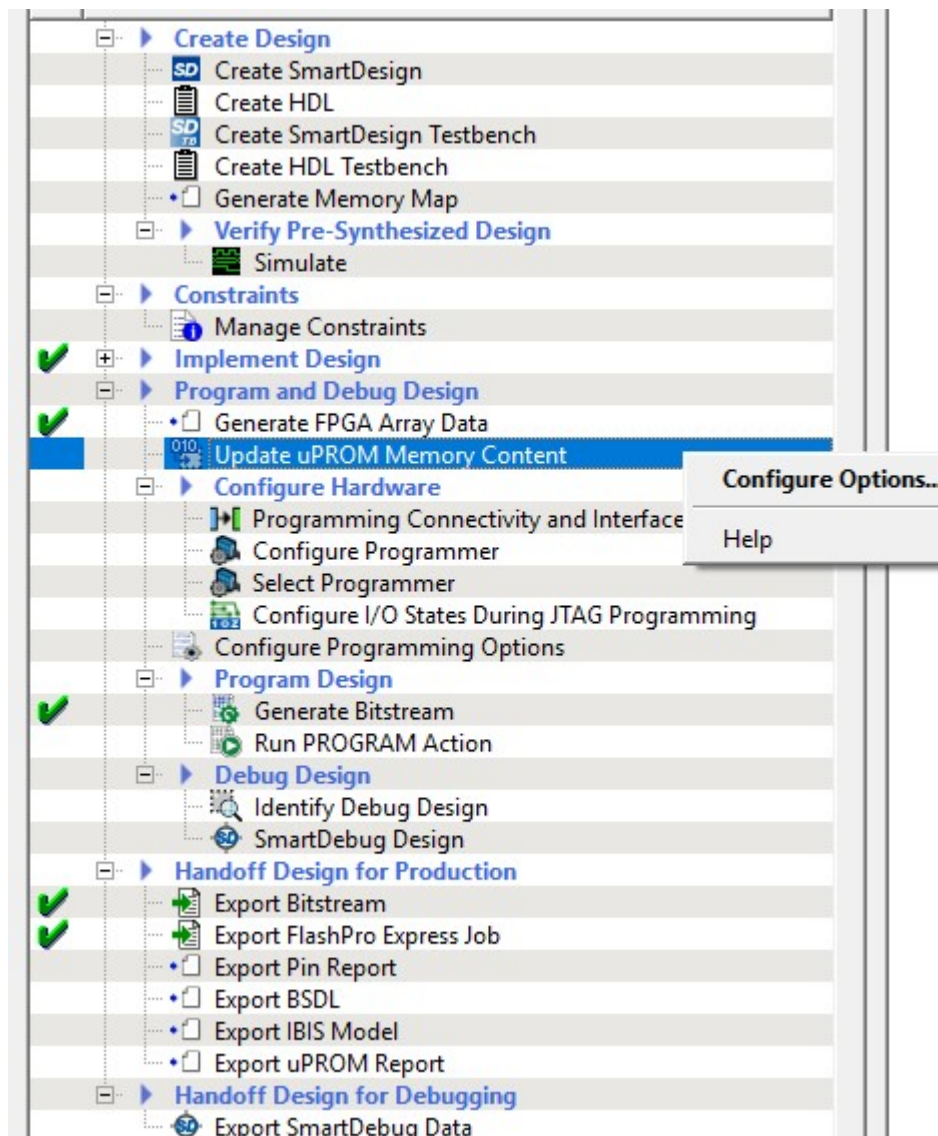
8. For SmartFusion 2/IGLOO 2 devices, if you need to change eNVM content, open the **Update eNVM Memory Content** dialog box (see the following figure). Changes you make in this dialog box are saved to the eNVM *.cfg file you imported.

Figure 6-2. Update eNVM Memory Content



9. For RTG4, if you need to change μ PROM content, open the **Update μ PROM Memory Content** dialog box (see the following figure). Changes you make in this dialog box are saved to the μ PROM.cfg file you imported.

Figure 6-3. μ PROM Memory Content Dialog Box



10. Generate a **Programming File** from this project and use it to program your FPGA.

7. Building Your Firmware Project [\(Ask a Question\)](#)

This section describes how to build your firmware project when using the Custom Flow.



Important: This section applies to SmartFusion 2 devices only.

The following types of files make up a firmware project:

- Source files (your firmware application)
- Drivers: These are drivers provided to facilitate your use of SmartFusion 2 SoC Components and Soft IP blocks. They include the CMSIS Hardware Abstraction Layer, which facilitates the use of the Cortex-M3 processor, and peripheral drivers (for example, MSS SPI, MSS UART, and so on).
- Peripheral Initialization Drivers: These files are generated by Libero SoC if you use the MDDR, FDDR, or SerDes components. Libero translates configuration settings for these blocks into register values that are stored in these files. You must import these into your firmware project manually, as listed:

To build your firmware project, perform the following steps:



Important: Steps 1–4 are detailed in the SmartFusion 2 CMSIS Hardware Abstraction Layer User Guide, which you can access using the Firmware catalog.

1. Select a Software IDE Tool.
2. Use the Firmware Catalog to download driver files for SoC Components or Soft IP you use in your Libero project.
3. Create a new firmware project using your Software IDE tool of choice.
4. Import driver files and write your application code as you normally would.
5. Create a directory in your firmware project.

```
<my_project>/drivers_config/sys_config
```

6. For each [Component Manifests](#) Report (generated in [Component Configuration](#)):
 - Import each file in the **Firmware files for all Software IDE tools** section ([Figure 2-1](#)) into your firmware project's `drivers_config/sys_config` directory.
7. Navigate to your Libero installation directory (where Libero is installed), and then navigate to the following directory:

```
<Libero install dir>\data\apa4M\sysconfig
```

- There are two files here:
 - `sysconfig.c`
 - `sysconfig.h`.
- Import `sysconfig.c` (as is, do not modify the file) into your firmware project's `drivers_config/sys_config` directory.
- Edit the local copy of `sysconfig.h`:
 - If you use the MDDR, change the following line:

```
#define SYS_MDDR_CONFIG_BY_CORTEX 0
```


to

```
#define SYS_MDDR_CONFIG_BY_CORTEX    1
```

- Similarly, depending on whether you use FDDR and SerDes blocks 0 to 3 in your design, change their respective lines as preceding.
- Import `sysconfig.h` into your firmware project's `drivers_config/sys_config` directory.

8. Import `sysconfig.h` into your firmware project's `drivers_config/sys_config` directory.

```
/*===== * MDDR configuration */
#define MSS_SYS_MDDR_CONFIG_BY_CORTEX 0 /
/*===== * FDDR configuration */
#define MSS_SYS_FDDR_CONFIG_BY_CORTEX 0 /
/*===== * SERDES Interface Configuration 0 */
#define MSS_SYS_SERDES_0_CONFIG_BY_CORTEX
#if MSS_SYS_SERDES_0_CONFIG_BY_CORTEX
#include "sys_config_SERDESIF_0.h"
#endif
#define MSS_SYS_SERDES_1_CONFIG_BY_CORTEX 0
#if MSS_SYS_SERDES_1_CONFIG_BY_CORTEX #include "sys_config_SERDESIF_1.h"
#endif
#define MSS_SYS_SERDES_2_CONFIG_BY_CORTEX 0
#if MSS_SYS_SERDES_2_CONFIG_BY_CORTEX #include "sys_config_SERDESIF_2.h"
#endif
#define MSS_SYS_SERDES_3_CONFIG_BY_CORTEX 0
#if MSS_SYS_SERDES_3_CONFIG_BY_CORTEX
#include "sys_config_SERDESIF_3.h"
#endif
```

8. Appendix A—Libero-Generated Hardware Configuration Files [\(Ask a Question\)](#)

This appendix describes the hardware configuration files that Libero generates. These files are intended to be imported into a firmware project. For more information, see [Building Your Firmware Project](#). Depending on the components present in a design, not all of these files are present.

<code>sys_config.h</code>	This header file contains information about the SmartFusion 2 MSS hardware configuration. The Libero hardware design flow generates it. The content of this file is hardware design specific. This file must not be included in the application code.
<code>sys_config.c</code>	This C source file contains information about the SmartFusion 2 MSS hardware configuration. The Libero hardware design flow generates it. The content of this file is hardware design specific. This file must be part of your software project if the hardware design uses one of the DDR memory controllers or a SerDes interface.
<code>sys_config_mss_clock.s.h</code>	This header file contains information about the SmartFusion 2 MSS hardware clock configuration. The Libero hardware design flow generates it. The content of this file is hardware design specific. This file must not be included in the application code.
<code>sys_config_mddr_define.h</code>	This header file contains information about the SmartFusion 2 MSS DDR hardware configuration. The Libero hardware design flow generates them if DDR is included in the Libero design. The content of this file is hardware design specific. This file must not be included in the application code.
<code>sys_config_SERDESIF_<0-3>.c</code>	These C source files contain information about the SmartFusion 2 SerDes interface hardware configuration. The Libero hardware design flow generates them if SerDes interfaces are included in the Libero design. A separate file is generated for each SerDes interface. The content of these files is hardware design specific. These files must be part of your software project if the hardware design uses one or more SerDes interfaces.
<code>sys_config_SERDESIF_<0-3>.h</code>	These header files contain information about the SmartFusion 2 SerDes interface hardware configuration. The Libero hardware design flow generates them if SerDes interfaces are included in the Libero design. A separate file is generated for each SerDes interface. The content of these files is hardware design specific. These files must not be included in the application code.


```
<top_level_instance_name>/CORERESETP_0/SOFT_SDIF0_1_CORE_RESET}}]
set_max_delay 0 -through [get_nets {\
<top_level_instance_name>/CORECONFIGP_0/FIC_2_APB_M_PSEL\
<top_level_instance_name>/CORECONFIGP_0/FIC_2_APB_M_PENABLE}}] -to [get_cells {\
<top_level_instance_name>/CORECONFIGP_0/FIC_2_APB_M_PREADY*\
<top_level_instance_name>/CORECONFIGP_0/state[0]]}
set_min_delay -24 -through \
[get_nets {<top_level_instance_name>/CORECONFIGP_0/FIC_2_APB_M_PWRITE\
<top_level_instance_name>/CORECONFIGP_0/FIC_2_APB_M_PADDR[*]\
<top_level_instance_name>/CORECONFIGP_0/FIC_2_APB_M_PWDATA[*]\
<top_level_instance_name>/CORECONFIGP_0/FIC_2_APB_M_PSEL\
<top_level_instance_name>/CORECONFIGP_0/FIC_2_APB_M_PENABLE}}]
```

9.2. PDC Physical Design Constraints [\(Ask a Question\)](#)

In the Libero IP core reference project, this top-level PDC constraint file is available from the Constraint Manager (**Design Flow > Open Manage Constraint View > Timing > Derive Constraints**).



Important: See this file to set the PDC constraints for the CoreConfigP component. Modify the full hierarchical path, if necessary, to match your design hierarchy or use the Derive_Constraints utility and steps in [Appendix D—Derive Constraints](#) on the component level PDC file. Save the *.pdc file to a different name. Import the PDC file to your project and use it for Compile, just like any other PDC constraint files.

9.2.1. Derived PDC file [\(Ask a Question\)](#)

This PDC design constraint file creates a region specifically for the CoreConfigP IP Core and places the core in the region created. This constrains the Place and Route engine to place the core in an optimal location resulting in better timing performance of the design when routed. The full hierarchical path from the top is given for the constraint. Modify, if necessary, the hierarchical path to match the names in your design.

```
# This file was generated based on the following PDC source files:
# W:/pc/11_7_1_14_lily/Designer/data/aPA4M/cores/constraints/PA4M12000/
# coreconfigp.pdc
# define_region -name {auto_coreconfigp} -type inclusive 1104 159 1451 299 assign_region
# {auto_coreconfigp} {<top_level_instance_name>/CORECONFIGP_0}
```

10. Appendix C—Importing Simulation Libraries into Simulation Environment [\(Ask a Question\)](#)

The default simulator for RTL simulation with Libero SoC is QuestaSim® Pro ME. Pre-compiled libraries for the default simulator is available with Libero installation at directory `<install_location>/Designer/lib/questasimpro/precompiled/vlog` for supported families.

Libero SoC also supports other third-party simulators editions of ModelSim, VCS, Xcelium, Active HDL, and Riviera Pro. Download respective pre-compiled libraries from [Libero SoC v12.0 and later](#) based on the simulator and its version.

Similar to the Libero environment, `run.do` file must be created to run simulation outside Libero.

Create a simple `run.do` file that has commands to establish library for compilation results, library mapping, compilation, and simulation. Follow the steps to create a basic `run.do` file.

1. Create a logical library to store compilation results using the `vlib` command, `vlib presynth`.
2. Map the logical library name to pre-compiled library directory using the `vmap` command, `vmap <logical_name> <pre-compiled directory path>`.
3. Compile source files—use language-specific compiler commands to compile design files into working directory.
 - `vlog` for `.v/.sv`
 - `vcom` for `.vhd`
4. Load the design for simulation using the `vsim` command by specifying name of any top-level module.
5. Simulate the design using `run` command.
6. After loading the design, simulation time is set to zero, and enter the `run` command to begin simulation.

In the simulator transcript window, execute `run.do` to run the simulation.

```
Sample run.do for SmartFusion2:
quietly set ACTELLIBNAME SmartFusion2
quietly set PROJECT_DIR "W:/Test/basic_test"

if {[file exists presynth/_info]} {
    echo "INFO: Simulation library presynth already exists"
} else {
    file delete -force presynth
    vlib presynth
}
vmap presynth presynth
vmap SmartFusion2 "X:/Libero/Designer/lib/questasimpro/precompiled/vlog/SmartFusion2"

vlog -sv -work presynth "${PROJECT_DIR}/hdl/top.v"
vlog "+incdir+${PROJECT_DIR}/stimulus" -sv -work presynth "${PROJECT_DIR}/stimulus/tb.v"

vsim -L PolarFire -L presynth -t lps presynth.tb
add wave /tb/*
run 1000ns
log /tb/*
exit
```

11. Appendix D—Derive Constraints [\(Ask a Question\)](#)

This appendix describes the Derive Constraints Tcl commands.

11.1. Derive Constraints Tcl Commands [\(Ask a Question\)](#)

The `derive_constraints` utility helps you derive constraints from the RTL or the configurator outside the Libero SoC design environment. To generate constraints for your design, you need the User HDL, Component HDL, and Component Constraints files. The SDC/NDC component constraints files are available under `<project>/component/work/<component_name>/<instance_name>/` directory after component configuration and generation.

Each component constraint file consists of the `set_component tcl` command (specifies the component name) and the list of constraints generated after configuration. The constraints are generated based on the configuration and are specific to each component.

11.1.1. Working with the `derive_constraints` Utility [\(Ask a Question\)](#)

Derive constraints traverse through the design and allocate new constraints for each instance of component based on previously provided component SDC/NDC/PDC files. For the CCC reference clocks, it propagates back through the design to find the source of the reference clock. If the source is an I/O, the reference clock constraint will be set on the I/O. If it is a CCC output or another clock source (for example, SerDes, oscillator), it uses the clock from the other component and reports a warning if the intervals do not match. Derive constraints will also allocate constraints for some macros like on-chip oscillators if you have them in your RTL.

To execute the `derive_constraints` utility, you must supply a `.tcl` file command-line argument with the following information in the specified order.

1. The device information is specified in [set_device](#).
2. The RTL path is specified in [read_verilog](#) or [read_vhdl](#).
3. Set top level module as per the [set_top_level](#).
4. Path to the component SDC/NDC files is specified in [read_sdc](#) or [read_ndc](#).
5. Execute the files as per the [derive_constraints](#).
6. The SDC/PDC/NDC derived constraints file is specified in [write_sdc](#) or [write_pdc](#) or [write_ndc](#).

The following is an example command-line argument to execute the `derive_constraints` utility.

```
$ <libero_installation_path>/bin{64}/derive_constraints derive.tcl
```

11.1.1.1. `set_device` [\(Ask a Question\)](#)

Description

Specify family name, die name, and speed grade.

```
set_device -family <family_name> -die <die_name> -speed <speed>
```

Arguments

Parameter	Type	Description
-family <family_name>	String	Specify the family name. Possible values are IGLOO® 2, SmartFusion® 2, and RTG4™.
-die <die_name>	String	Specify the die name.
-speed <speed>	String	Specify the device speed grade. Possible values are STD or -1.

Return Type	Description
0	Command succeeded.

set_device (continued)	
Return Type	Description
1	Command failed. There is an error. The console displays the error message.

List of Errors

Error Code	Error Message	Description
ERR0023	Required parameter—die is missing	The die option is mandatory and must be specified.
ERR0005	Unknown die MPF30	The value of -die option is not correct. See the possible list of values in option's description.
ERR0023	Parameter—die is missing value	The die option is specified without value.
ERR0023	Required parameter—family is missing	The family option is mandatory and must be specified.
ERR0004	Unknown family	The family option is not correct. See the possible list of values in option's description.
ERR0023	Parameter—family is missing value	The family option is specified without value.
ERR0023	Required parameter—speed is missing	The speed option is mandatory and must be specified.
ERR0007	Unknown speed <speed>	The speed option is not correct. See the possible list of values in option's description.
ERR0023	Parameter—speed is missing value	The speed option is specified without value.

Supported Families

RTG4™
SmartFusion® 2
IGLOO® 2

Example

```
set_device -family SmartFusion 2 -die M2S090T -speed -1
```

11.1.1.2. read_verilog [\(Ask a Question\)](#)

Description

Read a Verilog file using Verific.

```
read_verilog [-lib <libname>] [-mode <mode>] <filename>
```

Arguments

Parameter	Type	Description
-lib <libname>	String	Specify the library that contains the modules to be added into the library.
-mode <mode>	String	Specify the Verilog standard. Possible values are verilog_95, verilog_2k, system_verilog_2005, system_verilog_2009, system_verilog, verilog_ams, verilog_psl, system_verilog_mfcu. Values are case insensitive. Default is verilog_2k.
filename	String	Verilog file name

Return Type	Description
0	Command succeeded
1	Command failed. There is an error. The console displays the error message.

List of Errors

Error Code	Error Message	Description
ERR0023	Parameter—lib is missing value	The lib option is specified without value.
ERR0023	Parameter—mode is missing value	The mode option is specified without value.
ERR0015	Unknown mode <mode>	The specified verilog mode is unknown. See the list of possible verilog mode in—mode option description.
ERR0023	Required parameter file name is missing	No verilog file path is provided.
ERR0016	Failed due to Verific's parser	Syntax error in verilog file. The console displays the Verific's parser above the error message.
ERR0012	set_device is not called	The device information is not specified. Use set_device command to describe the device.

Supported Families

RTG4™
SmartFusion® 2
IGLOO® 2

Example

```
read_verilog -mode system_verilog {component/work/top/top.v}
```

```
read_verilog -mode system_verilog_mfcu design.v
```

11.1.1.3. read_vhdl [\(Ask a Question\)](#)

Description

Add a VHDL file into the list of VHDL files.

```
read_vhdl [-lib <libname>] [-mode <mode>] <filename>
```

Arguments

Parameter	Type	Description
-lib <libname>	—	Specify the library in which the content needs to be added.
-mode <mode>	—	Specifies the VHDL standard. Default is VHDL_93. Possible values are vhdl_93, vhdl_87, vhdl_2k, vhdl_2008, vhdl_psl. Values are case insensitive.
filename	—	VHDL file name

Return Type	Description
0	Command succeeded
1	Command failed. There is an error. The console displays the error message.

List of Errors

Error Code	Error Message	Description
ERR0023	Parameter—lib is missing value	The lib option is specified without value.
ERR0023	Parameter—mode is missing value	The mode option is specified without value.
ERR0018	Unknown mode <mode>	The specified VHDL mode is unknown. See the list of possible VHDL mode in—mode option description.
ERR0023	Required parameter file name is missing	No VHDL file path is provided.
ERR0019	Unable to register invalid_path.v file	The specified VHDL file does not exist or does not have read permissions.

read_vhdl (continued)

Error Code	Error Message	Description
ERR0012	set_device is not called	The device information is not specified. Use set_device command to describe the device.

Supported Families

RTG4™
SmartFusion® 2
IGLOO® 2

Example

```
read_vhdl -mode vhdl_2008 osc2dfn.vhd
```

```
read_vhdl {hdl/top.vhd}
```

11.1.1.4. set_top_level ([Ask a Question](#))**Description**

Specify the name of the top-level module in RTL.

```
set_top_level [-lib <libname>] <name>
```

Arguments

Parameter	Type	Description
-lib <libname>	String	The library to search for the top-level module or entity (Optional).
name	String	The top-level module or entity name.

Return Type	Description
0	Command succeeded
1	Command failed. There is an error. The console displays the error message.

List of Errors

Error Code	Error Message	Description
ERR0023	Required parameter top level is missing	The top level option is mandatory and must be specified.
ERR0023	Parameter—lib is missing value	The lib option is specified without values.
ERR0014	Unable to find top level <top> in library <lib>	The specified top-level module is not defined in the provided library. To fix this error, the top module or library name must be corrected.
ERR0017	Elaborate failed	Error in the RTL elaboration process. The console displays the error message.

Supported Families

RTG4™
SmartFusion® 2
IGLOO® 2

Example

```
set_top_level {top}
```

```
set_top_level -lib hdl top
```

11.1.1.5. read_sdc ([Ask a Question](#))**Description**

Read a SDC file into the component database.

```
read_sdc -component <filename>
```

Arguments

Parameter	Type	Description
-component	—	This is a mandatory flag for <code>read_sdc</code> command when we derive constraints.
filename	String	Path to the SDC file.

Return Type	Description
0	Command succeeded
1	Command failed. There is an error. The console displays the error message.

List of Errors

Error Code	Error Message	Description
ERR0023	Required parameter file name is missing.	The mandatory option file name is not specified.
ERR0000	SDC file <file_path> is not readable.	The specified SDC file does not have read permissions.
ERR0001	Unable to open <file_path> file.	The SDC file does not exist. The path must be corrected.
ERR0008	Missing <code>set_component</code> command in <file_path> file	The specified component of SDC file does not specify the component.
ERR0009	List of errors from sdc file	The SDC file contains incorrect sdc commands. Example when there is an error in <code>set_multicycle_path</code> constraint: Error while executing command <code>read_sdc: in <sdc_file_path> file: Error in command set_multicycle_path: Unknown parameter [get_cells {reg_a}]</code> .

Supported Families

RTG4™
SmartFusion® 2
IGLOO® 2

Example

```
read_sdc -component {./component/work/ccc0/ccc0_0/ccc0_ccc0_0_PF_CCC.sdc}
```

11.1.1.6. read_ndc ([Ask a Question](#))**Description**

Read a NDC file into the component database. The command is used for RTG4 designs using RTG4FCCCECALIB core.

```
read_ndc -component <filename>
```

Arguments

Parameter	Type	Description
-component	—	This is a mandatory flag for <code>read_ndc</code> command when we derive constraints.
filename	String	Path to the NDC file.

Return Type	Description
0	Command succeeded
1	Command failed. There is an error. The console displays the error message.

List of Errors

Error Code	Error Message	Description
ERR0001	Unable to open <file_path> file	The NDC file does not exist. The path must be corrected.
ERR0023	Required parameter—AtclParamO_ is missing.	The mandatory option filename is not specified.
ERR0023	Required parameter—component is missing	Component option is mandatory and must be specified.
ERR0000	NDC file <file_path> is not readable.	The specified NDC file does not have read permissions.

Supported Families

RTG4™
SmartFusion® 2
IGLOO® 2

Example

```
read_ndc -component {component/work/ccc1/ccc1_0/ccc1_ccc1_0_RTG4FCCCECALIB.ndc}
```

11.1.1.7. derive_constraints [\(Ask a Question\)](#)

Description

Instantiate component SDC/PDC/NDC files into the design-level database.

```
derive_constraints
```

Arguments

Return Type	Description
0	Command succeeded
1	Command failed. There is an error. The console displays the error message.

List of Errors

Error Code	Error Message	Description
ERR0013	Top-level is not defined	This means that the top-level module or entity is not specified. To fix this call, issue the <code>set_top_level</code> command before the <code>derive_constraints</code> command.

Supported Families

RTG4™
SmartFusion® 2
IGLOO® 2

Example

```
derive_constraints
```

11.1.1.8. write_sdc [\(Ask a Question\)](#)

Description

Writes a constraint file in SDC format.

```
write_sdc <filename>
```

Arguments

Parameter	Type	Description
<filename>	String	Path to the SDC file will be generated. This is a mandatory option. If the file exists, it will be overwritten.

Return Type	Description
0	Command succeeded
1	Command failed. There is an error. The console displays the error message.

List of Errors

Error Code	Error Message	Description
ERR0003	Unable to open <file path> file.	File path is not correct. Check whether the parent directories exist.
ERR0002	SDC file <file path> is not writable.	The specified SDC file does not have write permission.
ERR0023	Required parameter file name is missing.	The SDC file path is a mandatory option and needs to be specified.

Supported Families

RTG4™
SmartFusion® 2
IGLOO® 2

Example

```
write_sdc "derived.sdc"
```

11.1.1.9. write_pdc [\(Ask a Question\)](#)

Description

Writes physical constraints (Derive Constraints only).

```
write_pdc <filename>
```

Arguments

Parameter	Type	Description
<filename>	String	Path to the PDC file will be generated. This is a mandatory option. If the file path exists, it will be overwritten.

Return Type	Description
0	Command succeeded
1	Command failed. There is an error. The console displays the error message.

List of Errors

Error Code	Error Messages	Description
ERR0003	Unable to open <file path> file	The file path is not correct. Check whether the parent directories exist.
ERR0002	PDC file <file path> is not writeable.	The specified PDC file does not have write permission.
ERR0023	Required parameter file name is missing	The PDC file path is a mandatory option and needs to be specified.

Supported Families

RTG4™
SmartFusion® 2
IGLOO® 2

Example

```
write_pdc "derived.pdc"
```

11.1.1.10. write_ndc [\(Ask a Question\)](#)

Description

Writes NDC constraints into a file. The command is used for RTG4 designs using RTG4FCCCECALIB core.

```
write_ndc <filename>
```

Arguments

Parameter	Type	Description
filename	String	Path to the NDC file will be generated. This is a mandatory option. If the file exists, it will be overwritten.

Return Type	Description
0	Command succeeded
1	Command failed. There is an error. The console displays the error message.

List of Errors

Error Code	Error Messages	Description
ERR0003	Unable to open <file_path> file.	File path is not correct. The parent directories do not exist.
ERR0002	NDC file <file_path> is not writable.	The specified NDC file does not have write permission.
ERR0023	Required parameter _AtclParam0_ is missing.	The NDC file path is a mandatory option and needs to be specified.

Supported Families

RTG4™
SmartFusion® 2
IGLOO® 2

Example

```
write_ndc "derived.ndc"
```

11.1.1.11. add_include_path [\(Ask a Question\)](#)

Description

Specifies a path to search include files when reading RTL files.

```
add_include_path <directory>
```

Arguments

Parameter	Type	Description
directory	String	Specifies a path to search include files when reading RTL files. This option is mandatory.

Return Type	Description
0	Command succeeded
1	Command failed. There is an error. The console displays the error message.

List of Errors

Error Code	Error Message	Description
ERR0023	Required parameter include path is missing.	The directory option is mandatory and must be provided.



Important: If the directory path is not correct then `add_include_path` will be passed without an error. However, `read_verilog/read_vhd` commands will be fail due to Verific's parser.

Supported Families

RTG4™
SmartFusion® 2
IGLOO® 2

Example

```
add_include_path component/work/COREABC0/COREABC0_0/rtl/vlog/core
```

12. Revision History [\(Ask a Question\)](#)

The revision history describes the changes that were implemented in the document. The changes are listed by revision, starting with the most current publication.

Revision	Date	Description
P	12/2025	This document is released with Libero SoC Design Suite v2025.2 without changes from v2025.1.
N	05/2025	This document is released with Libero SoC Design Suite v2025.1 without changes from v2024.2.
M	09/2024	<ul style="list-style-type: none"> Updated Appendix C—Importing Simulation Libraries into Simulation Environment.
L	08/2024	<ul style="list-style-type: none"> Updated Figure 1-1. Updated step 4 in Synthesizing Your Design. Updated Appendix C—Importing Simulation Libraries into Simulation Environment.
K	02/2024	This document is released with Libero SoC Design Suite v2024.1 without any updates from v2023.2.
J	08/2023	This document is released with Libero SoC Design Suite v2023.2 without any technical content updates from v2023.1.
H	05/2023	Updated step 7 in Implementing Your Design .
G	04/2023	<ul style="list-style-type: none"> The information related to Polarfire family is moved to a separate document. For details, see PolarFire Custom Flow User Guide. Editorial updates are done throughout the document.
F	12/2022	Updated step 7 in Implementing Your Design .
E	08/2022	Added Error Codes for the following sections: <ul style="list-style-type: none"> set_device read_verilog read_vhdl set_top_level read_sdc read_ndc derive_constraints write_sdc write_pdc write_ndc add_include_path
D	04/2022	This document is released with Libero SoC Design Suite v2022.1 without any technical content updates from v2021.3. Editorial updates have been made throughout the document.
C	12/2021	<ul style="list-style-type: none"> Added Constraint Generation. Updated Appendix D—Derive Constraints.
B	08/2021	<ul style="list-style-type: none"> Updated Custom Flow. Added Appendix D—Derive Constraints.
A	08/2021	Initial Revision.

Microchip FPGA Support

Microchip FPGA products group backs its products with various support services, including Customer Service, Customer Technical Support Center, a website, and worldwide sales offices. Customers are suggested to visit Microchip online resources prior to contacting support as it is very likely that their queries have been already answered.

Contact Technical Support Center through the website at www.microchip.com/support. Mention the FPGA Device Part number, select appropriate case category, and upload design files while creating a technical support case.

Contact Customer Service for non-technical product support, such as product pricing, product upgrades, update information, order status, and authorization.

- From North America, call **800.262.1060**
- From the rest of the world, call **650.318.4460**
- Fax, from anywhere in the world, **650.318.8044**

Microchip Information

Trademarks

The “Microchip” name and logo, the “M” logo, and other names, logos, and brands are registered and unregistered trademarks of Microchip Technology Incorporated or its affiliates and/or subsidiaries in the United States and/or other countries (“Microchip Trademarks”). Information regarding Microchip Trademarks can be found at <https://www.microchip.com/en-us/about/legal-information/microchip-trademarks>.

ISBN: 979-8-3371-2480-3

Legal Notice

This publication and the information herein may be used only with Microchip products, including to design, test, and integrate Microchip products with your application. Use of this information in any other manner violates these terms. Information regarding device applications is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. Contact your local Microchip sales office for additional support or, obtain additional support at www.microchip.com/en-us/support/design-help/client-support-services.

THIS INFORMATION IS PROVIDED BY MICROCHIP “AS IS”. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE, OR WARRANTIES RELATED TO ITS CONDITION, QUALITY, OR PERFORMANCE.

IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE, INCIDENTAL, OR CONSEQUENTIAL LOSS, DAMAGE, COST, OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE INFORMATION OR ITS USE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE FULLEST EXTENT ALLOWED BY LAW, MICROCHIP’S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THE INFORMATION OR ITS USE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THE INFORMATION.

Use of Microchip devices in life support and/or safety applications is entirely at the buyer’s risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights unless otherwise stated.

Microchip Devices Code Protection Feature

Note the following details of the code protection feature on Microchip products:

- Microchip products meet the specifications contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is secure when used in the intended manner, within operating specifications, and under normal conditions.
- Microchip values and aggressively protects its intellectual property rights. Attempts to breach the code protection features of Microchip products are strictly prohibited and may violate the Digital Millennium Copyright Act.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of its code. Code protection does not mean that we are guaranteeing the product is “unbreakable”. Code protection is constantly evolving. Microchip is committed to continuously improving the code protection features of our products.