Libero® SoC v2021.2

Custom Flow User Guide

Introduction

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 can now be 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 specific family of devices. In this case, such information is clearly identified.

Table 1. Device Families Supported by Libero SoC

Device Family	Description
PolarFire®	PolarFire FPGAs deliver the industry's lowest power at mid-range densities with exceptional security and reliability.
PolarFire SoC	PolarFire SoC is the first SoC FPGA with a deterministic, coherent RISC-V CPU cluster, and a deterministic L2 memory subsystem enabling Linux and real-time applications.
SmartFusion®2	SmartFusion2 addresses the fundamental requirements for advanced security, high reliability, and low power in critical industrial, military, aviation, communications, and medical applications.
IGLOO®2	IGLOO2 is a low-power mixed-signal programmable solution.
RTG4™	RTG4 is Microchip's family of radiation-tolerant FPGAs.

Table of Contents

1. Overview

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 (SmartFusion2 and IGLOO2 only)	Yes	Stay in first Libero project.
Automatic PDC/SDC constraint generation	No	Derived constraints need all HDL files, and a derive_constraints utility when performed outside of Libero SoC, as described in 10. 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.
Configure Design Initialization Data and Memories	Yes	Use this tool to manage different types of memories and design initialization in PolarFire. Stay in second project.
Programming File Generation	Yes	Stay in second project.
Firmware Generation (SmartFusion2 only)	Yes	Stay in second project.
Firmware Debug (SmartFusion2 only)	No	Use third-party tool outside Libero SoC if desired.

Note: You must download precompiled libraries listed under the *Compiled Simulation Libraries for SmartFusion2, IGLOO2, RTG4 and PolarFire* section on the Libero SoC Documentation page to use a third-party simulator.

In a CPLD/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, SmartFusion2, IGLOO2, RTG4, and PolarFire 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 any blocks that comprise SoC functionality. These are:

- SmartFusion2 and IGLOO2

  - SmartFusion2 MSS (includes MDDR and eNVM)
  - IGLOO2 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.
- PolarFire
  - PF_UPROM
  - PF_SYSTEM_SERVICES
  - PF_CCC
  - PF CLK DIV
  - PF_CRYPTO
  - PF_DRI
  - PF_INIT_MONITOR
  - PF_NGMUX
  - PF_OSC
  - RAMs (TPSRAM, DPSRAM, URAM)
  - PF_SRAM_AHBL_AXI
  - PF_XCVR_ERM
  - PF_XCVR_REF_CLK
  - PF_TX_PLL
  - PF_PCIE
  - PF_IO
  - PF_IOD_CDR
  - PF_IOD_CDR_CCC
  - PF_IOD_GENERIC_RX
  - PF_IOD_GENERIC_TX
  - PF_IOD_GENERIC_TX_CCC
  - PF_RGMII_TO_GMII
  - PF_IOD_OCTAL_DDR
  - PF_DDR3
  - PF_DDR4
  - PF_LPDDR3
  - PF_QDR
  - PF_CORESMARTBERT
  - PF_TAMPER
  - PF_TVS, 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 uses 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 (2. 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 Lifecycle

The following steps describe the lifecycle of an SoC component and provides instructions on handle its 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 (SmartFusion2 and IGLOO2), uPROM (RTG4), and uPROM and sNVM (PolarFire)
   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/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. See 10. 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

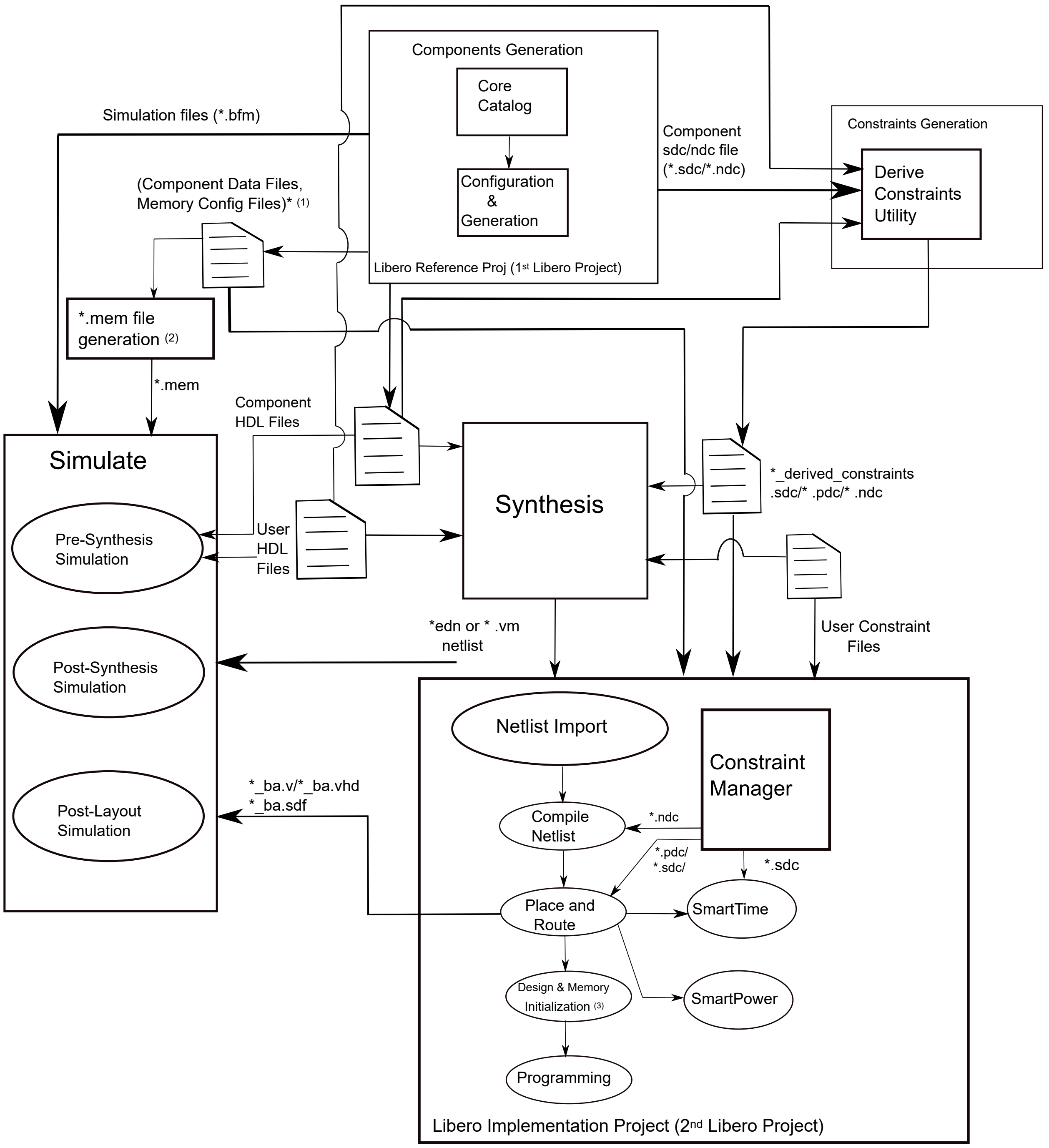
Some design steps must be run inside the Libero SoC environment (Table 1-1). For these steps to run, you must create two Libero SoC projects. The first project is used for design component configuration and generation, and the second is one for the physical implementation of the top-level design.

- 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 RTG4FCCCECALIB core, the automatic constraint generation (Derive Constraints) generates netlist constraint files (NDC) as well in addition to SDC, 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

The following custom flow figure describes how Libero SoC can be 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 depicts various steps involved in the flow, starting from design creation and stitching all the way to programming the device. The figure also lists the data exchange (inputs and outputs) that must occur at each design flow step.

Figure 1-1. Custom Flow Overview



Notes:

1. Component Data Files and Memory Config Files:
   a. Component Data Files and Memory Config Files:
      i. SmartFusion2/IGLOO2: *.reg files for MDDR/FDDR/SerDes blocks.
   b. Memory Config Files:
      i. SmartFusion2/IGLOO2: ENVM.cfg.
      ii. RTG4: UPROM.cfg.
      iii. PolarFire: SNVM.cfg, UPROM.cfg.
2. *.mem file generation for Simulation for different families:
   a. SmartFusion2: pa4mssenvmgen.exe takes ENVM.cfg as input and generates ENVM_init.mem.
   b. IGLOO2: pa4mssenvmgen.exe takes *.reg files for MDDRR/FDDR/SerDes blocks and ENVM.cfg as input and generates ENVM_init.mem.
   c. RTG4, PolarFire: pa4rtupromgen.exe takes UPROM.cfg as input and generates UPROM.mem.
3. For PolarFire only.

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.
      Note: For SmartFusion2 and IGLOO2 System Builder and SmartFusion2 MSS blocks, generate the component in the SmartDesign canvas after configuration of the SmartFusion2 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 2. Component Configuration.
2. Complete your RTL design outside of 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 SmartFusion2/IGLOO2), 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 10. 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 Libero SoC.
5. Firmware tool (SmartFusion2 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 C 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 SmartFusion2 and IGLOO2, import post-synthesis `*.edn netlist` (**File** > **Import** > **Others)**.
   – For PolarFire, import post-synthesis `*.vm` netlist (**File** > **Import** > **Synthesized Verilog Netlist (VM)**).
   – Component metadata such as `*.reg` files for MDDR/FDDR/SerDes (SmartFusion2 and IGLOO2), `*.cfg` file for eNVM (SmartFusion2 and IGLOO2), and `*.cfg` file for uPROM (RTG4), `*.cfg` files for uPROM and/or sNVM (PolarFire).

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 PolarFire Block Flow User Guide and SmartFusion2, RTG4, IGLOO2 Block Flow User 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 (SmartFusion2 and IGLOO2), 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, configure design initialization data and memories (PolarFire only), and programming file generation.
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

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 1. Overview in your design, perform the steps described in this section:

If you are not using any of the above components, you can write your RTL outside of Libero and directly import it into your Synthesis and Simulation tools. You can 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

After selecting the components which 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 SmartFusion2 MSS, or if you use SmartFusion2 or IGLOO2 System Builder, make the appropriate selection in the **Use Design Flow** section of the New Project window.
2. If you use Smart Fusion2 or IGLOO2 System Builder (for IGLOO2, you must use System Builder to configure the HPMS, eNVM, MDDR, and FDDR):
   a. Use System Builder to select your components and configure your system.
   b. Generate your system in 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**).
      Note: The port names—you need them 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. You can exit the simulator once it is invoked—this step generates the simulation files necessary for your project.
      Note: 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 System Builder (SmartFusion2 only):
   a. If you select the SmartFusion2 MSS in the **Use Design Flow** subsection (step 1.b), the SmartFusion2 MSS Configurator automatically opens. Otherwise, in the **Design Flow** window, double-click **Configure MSS**. Double-click the SmartFusion2 MSS instance to open its configurator.
      i. Double-click the SmartFusion2 MSS instance to open its configurator.
      ii. Configure the SmartFusion2 MSS as per your requirements.
          Note: If you use the eNVM or the MDDR, you must use the SmartFusion2 MSS to configure it.
      iii. Save and generate the SmartFusion2 MSS component.
          Note: If you do not use System Builder, and you have SmartFusion2 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 SmartDesigns built in the preceding steps.
   e. Double-click the **Simulate** tool (Pre-Synthesis) to invoke the simulator. You can exit the simulator after it is invoked; this step simply generates the simulation files necessary for your project.

Note: You must perform this step if you want to simulate your design outside Libero.
For more information, see the 4. Simulating Your Design.

f. Generate the SDC and the PDC files through Derive Constraints utility outside of Libero SoC by using Component HDL and User HDL files. The path of newly generated SDC, PDC files must be specified by user. For further information, see 10. Appendix D—Derive Constraints.
Note: You must pass this `*.sdc` file to Synthesis when you exit Libero and run Synthesis outside Libero. For the floorplanning PDC constraint file, you must pass it to Place and Route in the second Libero project you will create to implement your design.

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

4. If you use SmartFusion2, and any of the SmartFusion2 MSS peripherals (MDDR, FDDR, or SerDes), you must export your firmware project (`SoftConsole/IAR/Keil`) from this Libero project. For more information, see 6. 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.
   Note: If you are using the SerDes and the FDDR blocks in your design, but not their corresponding INIT cores, you must:
      i. 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 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 uPROM, add the uPROM block to the top level SmartDesign.
   e. Generate all SmartDesigns built in the above steps.
   f. Double-click **Simulate** (Pre-Synthesis) to invoke the simulator. You can exit the simulator after it is invoked; this step just generates the simulation files necessary for your project.
   Note: To generate `UPROM.mem` file (used for simulating the UPROM contents) from the `UPROM.cfg` file (generated by the configurator) using a stand-alone executable outside Libero, see 4. Simulating Your Design. You must perform this step if you want to simulate your design outside of Libero.
   g. Use Derive Constraint utility outside of Libero environment to generate SDC and NDC constraints. To generate SDC constraints, supply component HDL, component SDC, User HDL files to derive constraint tool, and specify the path where to generate the SDC file. For RTG4 designs using the RTG4FCCCECALIB core, supply the component NDC file of RTG4FCCCECALIB core as well, and specify the path where to write the NDC constraints. For further information, see 10. Appendix D—Derive Constraints.

   To pass the `*.sdc` file to synthesis when you exit Libero, run synthesis outside Libero. For place and route and timing verification tools in the second Libero project create to implement your design. For the netlist NDC constraint file, you must pass it to the compile netlist tool in the second Libero project, you will create, to implement your design.
   h. Save your project—this is your reference project.
   Note: You must follow 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.
6. If you use PolarFire device, use one or more of the PolarFire cores mentioned in 1.3 Custom Flow.
   a. Create a SmartDesign and Configure the desired core and instantiate it in the SmartDesign component.
   b. Promote all the pins to top level.
   c. Generate the SmartDesign.
   d. Double-click the **Simulate** tool (any of Pre-Synthesis or Post-Synthesis or Post-Layout options) to invoke the simulator. You can exit the simulator after it is invoked. This step generates the simulation files necessary for your project.

Note: You must perform this step if you want to simulate your design outside Libero. For more information, see 4. Simulating Your Design.

e. To generate SDC constraints with derive constraint tool outside of Libero SoC supply Component HDL, Component SDC, and User HDL file to the tool and specify where write the SDC constraints. For further information, see 10. Appendix D—Derive Constraints.
You must pass the generated `*.sdc` file to Synthesis when you exit Libero and run Synthesis outside Libero, and to the Place and Route and Timing Verification tools in the second Libero project you will create to implement your design.

f. Save your project—this is your reference project.

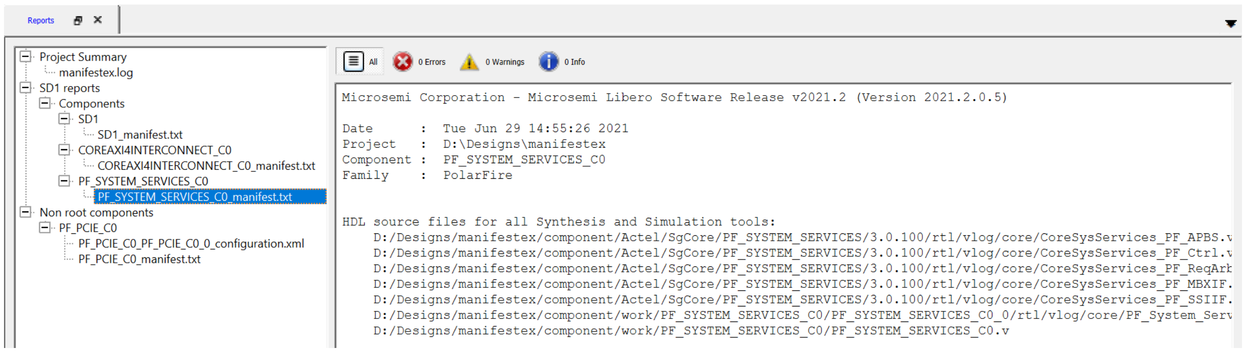
2.2 Component Manifests

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. You can access 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.

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

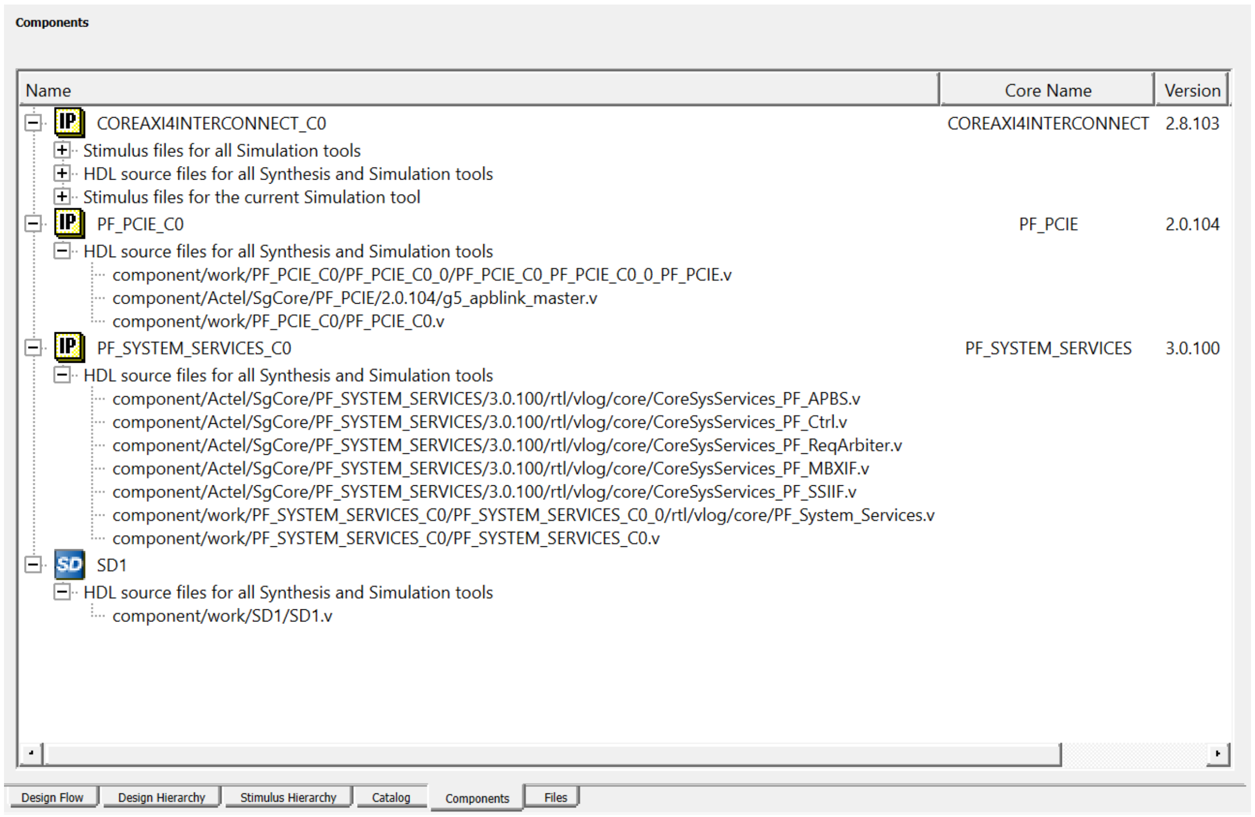
Alternatively, you can access the individual manifest report files for each core component generated or SmartDesign component from `<project>/component/work/<component name>/<instance name>/<component name>_manifest.txt` or `<project>/component/work/<SmartDesign name>/<SmartDesign name>_manifest.txt`. You can also 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 PolarFire)



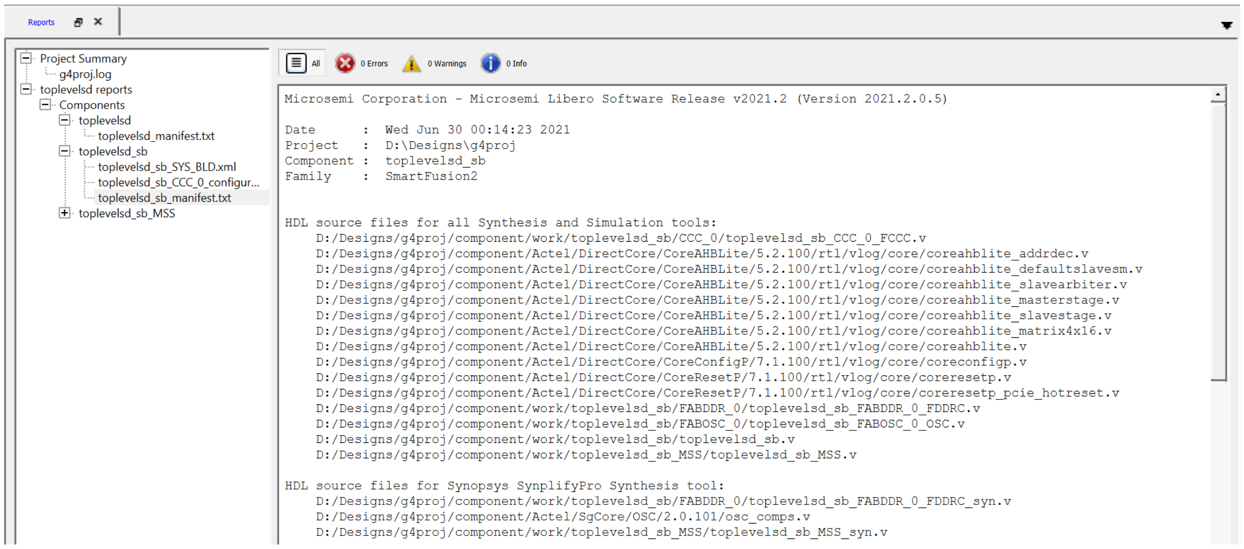
```
Microsemi Corporation - Microsemi Libero Software Release v2021.2 (Version 2021.2.0.5)

Date      :  Tue Jun 29 14:55:26 2021
Project   :  D:\Designs\manifestex
Component :  PF_SYSTEM_SERVICES_C0
Family    :  PolarFire


HDL source files for all Synthesis and Simulation tools:
    D:/Designs/manifestex/component/Actel/SgCore/PF_SYSTEM_SERVICES/3.0.100/rtl/vlog/core/CoreSysServices_PF_APBS.v
    D:/Designs/manifestex/component/Actel/SgCore/PF_SYSTEM_SERVICES/3.0.100/rtl/vlog/core/CoreSysServices_PF_Ctrl.v
    D:/Designs/manifestex/component/Actel/SgCore/PF_SYSTEM_SERVICES/3.0.100/rtl/vlog/core/CoreSysServices_PF_ReqArb
    D:/Designs/manifestex/component/Actel/SgCore/PF_SYSTEM_SERVICES/3.0.100/rtl/vlog/core/CoreSysServices_PF_MBXIF.
    D:/Designs/manifestex/component/Actel/SgCore/PF_SYSTEM_SERVICES/3.0.100/rtl/vlog/core/CoreSysServices_PF_SSIIF.
    D:/Designs/manifestex/component/work/PF_SYSTEM_SERVICES_C0/PF_SYSTEM_SERVICES_C0_0/rtl/vlog/core/PF_System_Serv
    D:/Designs/manifestex/component/work/PF_SYSTEM_SERVICES_C0/PF_SYSTEM_SERVICES_C0.v
```

Figure 2-2. Accessing Component Manifest Report Files from Libero Components Tab



Name	Core Name	Version
COREAXI4INTERCONNECT_C0	COREAXI4INTERCONNECT	2.8.103
Stimulus files for all Simulation tools		
HDL source files for all Synthesis and Simulation tools		
Stimulus files for the current Simulation tool		
PF_PCIE_C0	PF_PCIE	2.0.104
HDL source files for all Synthesis and Simulation tools		
component/work/PF_PCIE_C0/PF_PCIE_C0_0/PF_PCIE_C0_PF_PCIE_C0_0_PF_PCIE.v		
component/Actel/SgCore/PF_PCIE/2.0.104/g5_apblink_master.v		
component/work/PF_PCIE_C0/PF_PCIE_C0.v		
PF_SYSTEM_SERVICES_C0	PF_SYSTEM_SERVICES	3.0.100
HDL source files for all Synthesis and Simulation tools		
component/Actel/SgCore/PF_SYSTEM_SERVICES/3.0.100/rtl/vlog/core/CoreSysServices_PF_APBS.v		
component/Actel/SgCore/PF_SYSTEM_SERVICES/3.0.100/rtl/vlog/core/CoreSysServices_PF_Ctrl.v		
component/Actel/SgCore/PF_SYSTEM_SERVICES/3.0.100/rtl/vlog/core/CoreSysServices_PF_ReqArbiter.v		
component/Actel/SgCore/PF_SYSTEM_SERVICES/3.0.100/rtl/vlog/core/CoreSysServices_PF_MBXIF.v		
component/Actel/SgCore/PF_SYSTEM_SERVICES/3.0.100/rtl/vlog/core/CoreSysServices_PF_SSIIF.v		
component/work/PF_SYSTEM_SERVICES_C0/PF_SYSTEM_SERVICES_C0_0/rtl/vlog/core/PF_System_Services.v		
component/work/PF_SYSTEM_SERVICES_C0/PF_SYSTEM_SERVICES_C0.v		
SD1		
HDL source files for all Synthesis and Simulation tools		
component/work/SD1/SD1.v		

Figure 2-3. Accessing Component Manifest Report Files from Libero Reports Tab (For SmartFusion2 System Builder)



```
Microsemi Corporation - Microsemi Libero Software Release v2021.2 (Version 2021.2.0.5)

Date      :  Wed Jun 30 00:14:23 2021
Project   :  D:\Designs\g4proj
Component :  toplevelsd_sb
Family    :  SmartFusion2


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_defaultslavesm.v
    D:/Designs/g4proj/component/Actel/DirectCore/CoreAHBLite/5.2.100/rtl/vlog/core/coreahblite_slavearbiter.v
    D:/Designs/g4proj/component/Actel/DirectCore/CoreAHBLite/5.2.100/rtl/vlog/core/coreahblite_masterstage.v
    D:/Designs/g4proj/component/Actel/DirectCore/CoreAHBLite/5.2.100/rtl/vlog/core/coreahblite_slavestage.v
    D:/Designs/g4proj/component/Actel/DirectCore/CoreAHBLite/5.2.100/rtl/vlog/core/coreahblite_matrix4x16.v
    D:/Designs/g4proj/component/Actel/DirectCore/CoreAHBLite/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
```

Focus on the following Component Manifest Reports:

- If you use SmartFusion2 or IGLOO2 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

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

Component Manifest (example of a SmartFusion2 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_defaultslavesm.v
    D:/Designs/g4proj/component/Actel/DirectCore/CoreAHBLite/5.2.100/rtl/vlog/
core/coreahblite_slavearbiter.v
    D:/Designs/g4proj/component/Actel/DirectCore/CoreAHBLite/5.2.100/rtl/vlog/
core/coreahblite_masterstage.v
    D:/Designs/g4proj/component/Actel/DirectCore/CoreAHBLite/5.2.100/rtl/vlog/
core/coreahblite_slavestage.v
    D:/Designs/g4proj/component/Actel/DirectCore/CoreAHBLite/5.2.100/rtl/vlog/
core/coreahblite_matrix4x16.v
    D:/Designs/g4proj/component/Actel/DirectCore/CoreAHBLite/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/SmartFusion2MSS/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
```

Component Manifest (example of a PolarFire core component)

```
HDL source files for all Synthesis and Simulation tools:
    D:/Designs/manifestex/component/Actel/SgCore/PF_SYSTEM_SERVICES/
3.0.100/rtl/vlog/core/CoreSysServices_PF_APBS.v
    D:/Designs/manifestex/component/Actel/SgCore/PF_SYSTEM_SERVICES/
3.0.100/rtl/vlog/core/CoreSysServices_PF_Ctrl.v
    D:/Designs/manifestex/component/Actel/SgCore/PF_SYSTEM_SERVICES/
3.0.100/rtl/vlog/core/CoreSysServices_PF_ReqArbiter.v
    D:/Designs/manifestex/component/Actel/SgCore/PF_SYSTEM_SERVICES/
3.0.100/rtl/vlog/core/CoreSysServices_PF_MBXIF.v
    D:/Designs/manifestex/component/Actel/SgCore/PF_SYSTEM_SERVICES/
3.0.100/rtl/vlog/core/CoreSysServices_PF_SSIIF.v
    D:/Designs/manifestex/component/work/PF_SYSTEM_SERVICES_C0/
PF_SYSTEM_SERVICES_C0_0/rtl/vlog/core/PF_System_Services.v
    D:/Designs/manifestex/component/work/PF_SYSTEM_SERVICES_C0/
PF_SYSTEM_SERVICES_C0.v
```

Each type of file is necessary downstream in your design flow. The following chapters describe how to integrate files from the manifest into your design flow.

Note: The component SDC/NDC files are not included in the Component Manifest files.
The component SDC/NDC files are available under <project>/component/work/<component name>/<instance_name>/directory after component configuration and generation.

Note: Keep track of the location of all these files. These files are used downstream in the custom flow. The derived `*.sdc` and `*.pdc` files are given in SDC Timing Constraints and PDC Physical Design Constraints in Appendix B.

3. Synthesizing Your Design

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, follow the steps:

1. Create a new project in your Synthesis tool, targeting the same device family, die and package as the Libero project your first created.
   a. Import your own RTL files as you normally do.
   b. Set the Synthesis output to be Structural Verilog (`.vm`).

   Note: Structural Verilog `(.vm)` is the only supported synthesis output format in PolarFire.
2. Import Component HDL files into your Synthesis project:
   a. For each Component Manifest 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. For PolarFire only: Import the file `polarfire_syn_comps.v` (if using Synopsys Synplify) from `<Libero Installation location>/data/aPA5M` to your Synthesis project.
4. Import the previously generated sdc file through Derived Constraint tool (in Appendix B) into the Synthesis tool. This constraint file constrains the synthesis tool to achieve timing closure with less effort and fewer design iterations.

Notes: 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 ensure 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.

1. Import any Netlist Attributes `*.ndc`, if any, into the Synthesis tool.
2. Run Synthesis.

Note: 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.

4. Simulating Your Design

To simulate your design outside of Libero (that is, using your own simulation environment and simulator), follow the 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 3. Synthesizing Your Design) into your simulation project and compile it.

   c. Post-layout simulation:
      - First,complete implementing your design (see 5. Implementing Your Design). Ensure 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. Ensure that any Tcl files in the preceding lists (in step 2.a) are executed first, before the start of simulation.
   c. For SmartFusion2 only:
      - Review the `subsystem.bfm` file. Based on your usage of the MDDR, FDDR, or SerDes, ensure that the following lines are present (or absent) in the `subsystem.bfm` file — 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 #efine
USE_SERDESIF_0 #define USE_SERDESIF_1 #define USE_SERDESIF_2 # define
USE_SERDESIF_3
```

   d. ENVM_init.mem: If you use the eNVM (SmartFusion2 or IGLOO2), or if you use IGLOO2 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 subsequent steps below.
   e. UPROM.mem: If you use RTG4 uPROM, you must use the `pa4rtupromgen.exe` to generate the `UPROM.mem` file. The pa4rtupromgen executable takes the `UPROM.cfg` file as inputs through a Tcl script file and outputs the `UPROM.mem` file required for simulations. This file must be copied to the simulation folder prior to the simulation run. An example showing the pa4rtupromgen executable usage is provided in the subsequent steps below.
   f. UPROM.mem (PolarFire): If you use the PolarFire uPROM core in your design with the option **Use content for simulation** enabled for one or more data storage clients that you wish to simulate, you must use the executable pa4rtupromgen (`pa4rtupromgen.exe on Windows`) to generate the

UPROM.mem file. The pa4rtupromgen executable takes the UPROM.cfg file as inputs through a Tcl script file and outputs the UPROM.mem file required for simulations. This UPROM.mem file must be copied to the simulation folder prior to the simulation run. An example showing the pa4rtupromgen executable usage is provided in the subsequent steps below. The UPROM.cfg file will be available in the directory <Project>/component/work/<uPROM component name>/<uPROM instance name> in the Libero project you used to generate the PolarFire uPROM component.

g. snvm.mem (PolarFire): If you use the PolarFire System Services core in your design and configured the sNVM tab in the core with the option **Use content for simulation** enabled for one or more clients that you wish to simulate, then a snvm.mem file is automatically generated to the directory <Project>/component/work/<PolarFire System Services component name>/<uPROM instance name> in the Libero project you used to generate the PolarFire System Services component. This snvm.mem 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 SmartFusion2 and IGLOO2, 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.

   For PolarFire, copy the UPROM.cfg file from the first Libero project created for component generation into the working folder.
5. IGLOO2: Paste the following commands in a *.tcl script and place it in the simulation folder created in step 3.

```
Sample*.tcl for IGLOO2 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 IGLOO2, 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. SmartFusion2: Paste the following commands in a `*.tcl` script and place it in the simulation folder created in step 3.

   ```
   Sample *.tcl for SmartFusion2 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 SmartFusion2.

   For SmartFusion2, 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. PolarFire: Paste the following commands in a `*.tcl` script and place it in the simulation folder created in step 3.

    ```
    Sample *.tcl for PolarFire 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).

    The argument `use_init` must be set to false for PolarFire.

    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.
12. 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 11.
    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
    ```
13. For SmartFusion2 and IGLOO2, 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 `UPROM.mem` file is generated in the location specified in the `set_sim_mem` command in the `*.tcl` script.

    For PolarFire, after successful execution of the pa4rtupromgen executable, check that the `UPROM.mem` file is generated in the location specified in the `set_sim_mem` command in the `*.tcl` script.
14. For SmartFusion2 and IGLOO2, 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 `UPROM.mem` file into the top level simulation folder of your simulation project to run simulation (outside of Libero SoC).

    For PolarFire, to simulate the sNVM contents configured as a part of the PolarFire System Services core, copy the `snvm.mem` file from your first Libero project (used for component configuration) into the top level simulation folder of your simulation project to run simulation (outside of Libero SoC). To simulate UPROM contents, copy the generated `UPROM.mem` file into the top level simulation folder of your simulation project to run simulation (outside of Libero SoC).

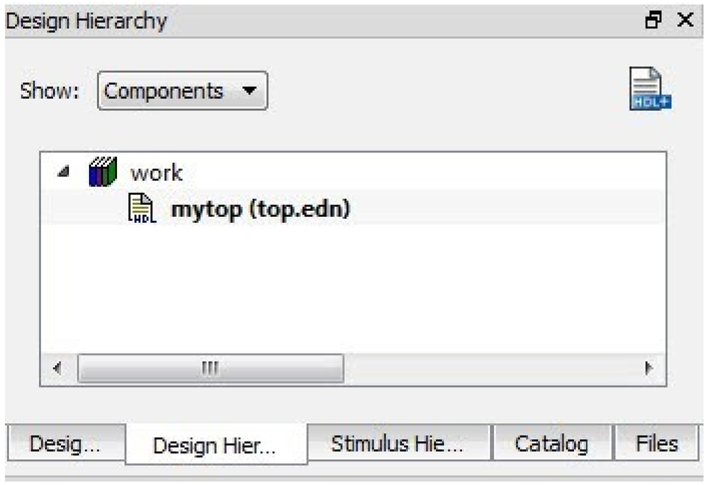
Note: To simulate the functionality of SoC Components, download the pre-compiled SmartFusion2/IGLOO2, RTG4 or PolarFire simulation libraries and import them into your simulation environment as described here. For more details, see 9. Appendix C—Importing Simulation Libraries into Simulation Environment.

5. Implementing Your Design

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. Ensure to target the same device as in the reference project you created in 2. 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)`).
   Note: 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 5-1. Root Module in Design Hierarchy



4. Import the constraints into the Libero project. Use the 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 > Floor Planner > Import)`. If your design contains CoreConfigP (SmartFusion2 and IGLOO2 only), ensure 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 1. 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.

   Note: The derived SDC timing constraint file constrains your design contains various IP cores to achieve timing closure with less effort and fewer design iterations. If Place and Route fails with this `*.sdc` constraint file, import this same `*.sdc` file to synthesis and re-run synthesis.

Note: SmartFusion2 and IGLOO2 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 SmartFusion2 and IGLOO2, 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>
            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:

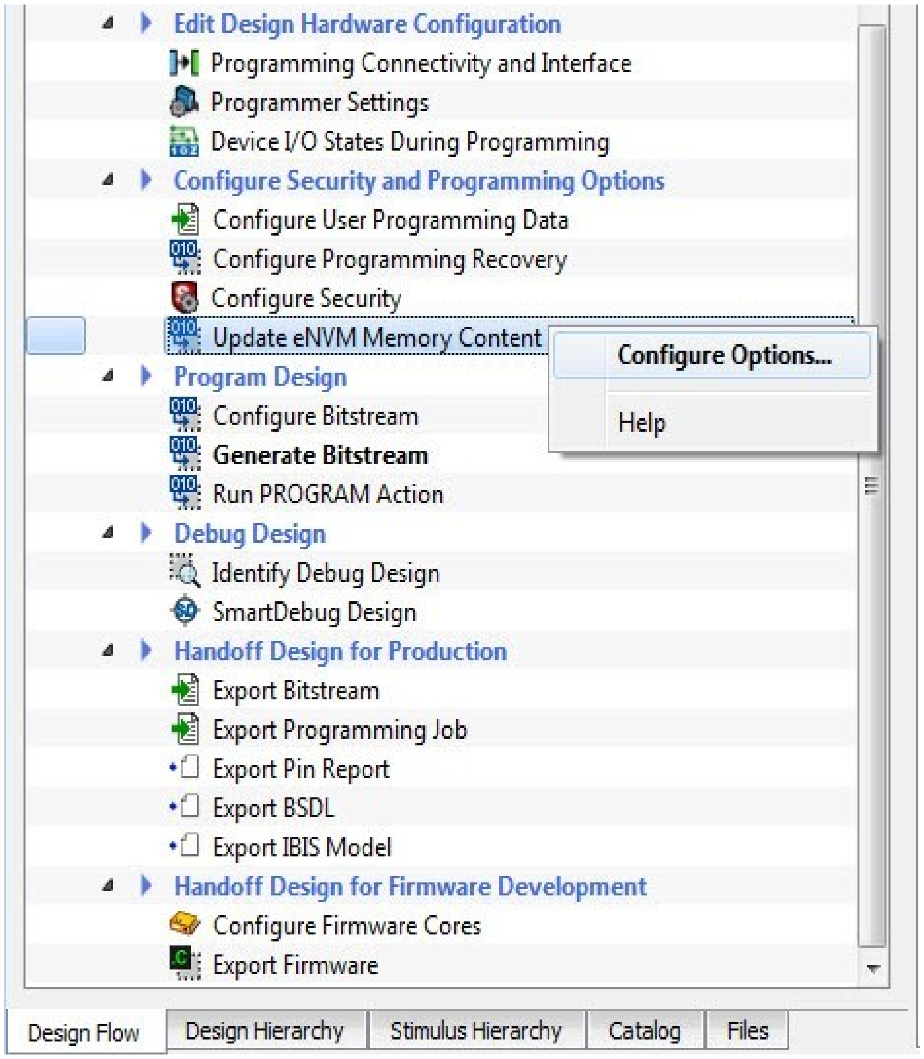
```
import_component_data
            module <name of root component>
            uprom_cfg <path to uPROM cfg>
```

   Note: All configuration files imported with the `import_component_data`. Tcl command are imported to the `designer/<root name>/component/ folder` in the Libero project directory.

   Note: For SmartFusion2, if you do not run SmartPower, you can skip importing the `*.reg files`, and you only need to import the `ENVM.cfg` file. For IGLOO2, you must import all `*.reg` and `ENVM.cfg` files specified in all your relevant Component Manifests Reports.

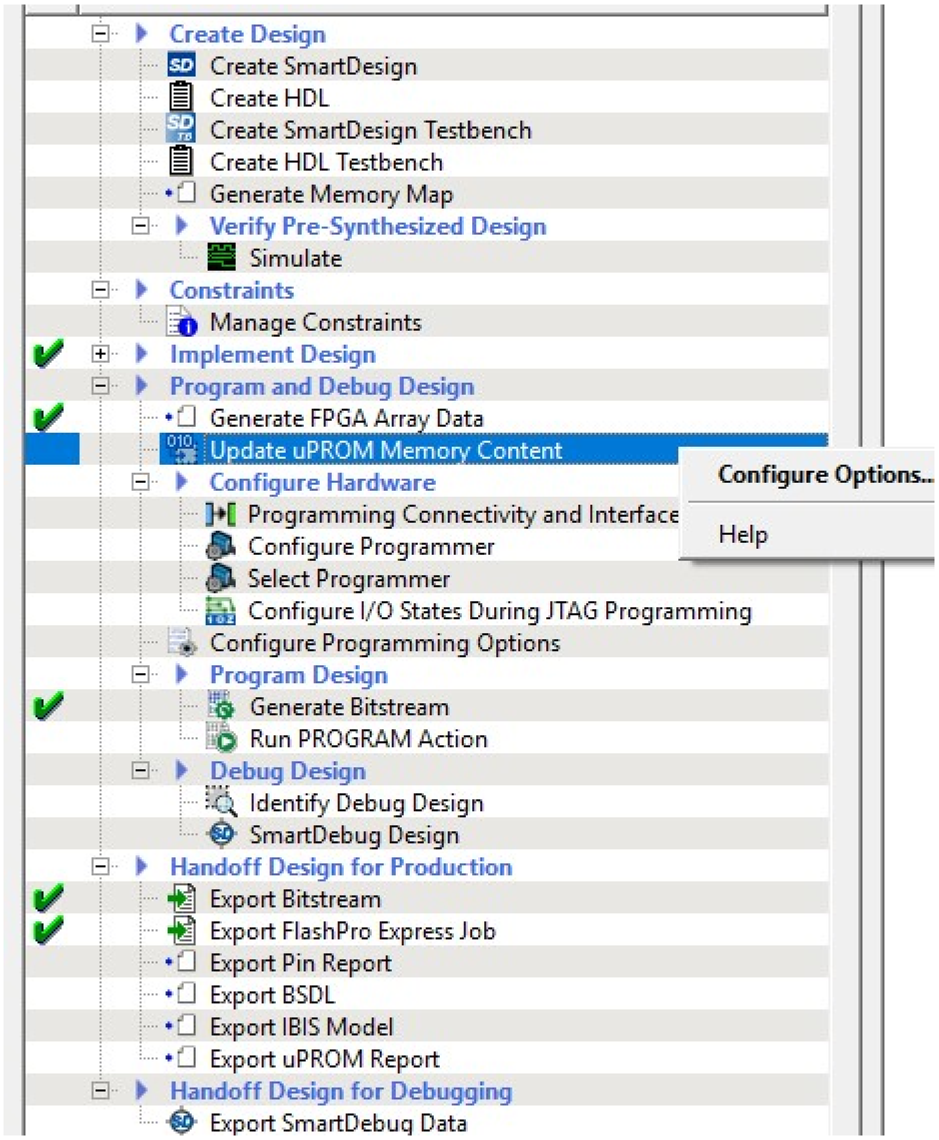
   Note: For RTG4, only the UPROM `.cfg` file can be imported.
8. For SmartFusion2/IGLOO2, if you need to change eNVM content, open the **Update eNVM Memory Content** dialog box (refer to the following figure). Changes you make in this dialog box are saved to the eNVM `*.cfg` file you imported

Figure 5-2. Update eNVM Memory Content



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

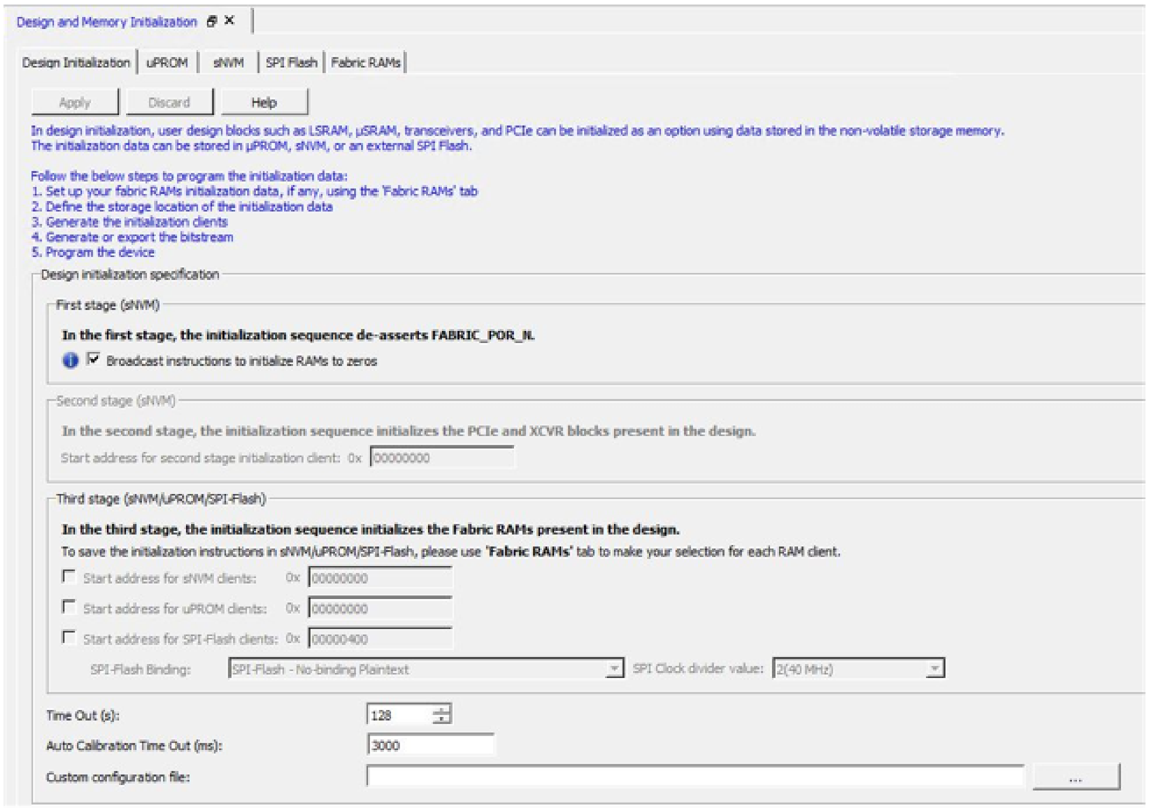
Figure 5-3. uPROM Memory Content Dialog Box


Create Design
Create SmartDesign
Create HDL
Create SmartDesign Testbench
Create HDL Testbench
Generate Memory Map
Verify Pre-Synthesized Design
Simulate
Constraints
Manage Constraints
✓ Implement Design
Program and Debug Design
✓ Generate FPGA Array Data
Update uPROM Memory Content
Configure Hardware
Programming Connectivity and Interface
Configure Programmer
Select Programmer
Configure I/O States During JTAG Programming
Configure Programming Options
Program Design
✓ Generate Bitstream
Run PROGRAM Action
Debug Design
Identify Debug Design
SmartDebug Design
Handoff Design for Production
✓ Export Bitstream
✓ Export FlashPro Express Job
Export Pin Report
Export BSDL
Export IBIS Model
Export uPROM Report
Handoff Design for Debugging
Export SmartDebug Data

10. **PolarFire**: The Configure Design Initialization Data and Memories tool allows you to initialize design blocks such as LSRAM, uSRAM, XCVR (transceivers), and PCIe using data stored in nonvolatile uPROM, sNVM, or external SPI Flash storage memory. The tool has the following tabs for defining the specification of the design initialization sequence and the specification of the initialization clients as well as user data clients.
    - Design Initialization tab
    - uPROM tab
    - sNVM tab
    - SPI Flash tab
    - Fabric RAMs tab

    Use the tabs in the tool to configure the design initialization data and memories.

Figure 5-4. Design Initialization data and memories



In design initialization, user design blocks such as LSRAM, µSRAM, transceivers, and PCIe can be initialized as an option using data stored in the non-volatile storage memory.
The initialization data can be stored in µPROM, sNVM, or an external SPI Flash.

Follow the below steps to program the initialization data:
1. Set up your fabric RAMs initialization data, if any, using the 'Fabric RAMs' tab
2. Define the storage location of the initialization data
3. Generate the initialization clients
4. Generate or export the bitstream
5. Program the device

Design initialization specification

First stage (sNVM)

In the first stage, the initialization sequence de-asserts FABRIC_POR_N.

[✓] Broadcast instructions to initialize RAMs to zeros

Second stage (sNVM)

In the second stage, the initialization sequence initializes the PCIe and XCVR blocks present in the design.

Start address for second stage initialization client: 0x 00000000

Third stage (sNVM/uPROM/SPI-Flash)

In the third stage, the initialization sequence initializes the Fabric RAMs present in the design.

To save the initialization instructions in sNVM/uPROM/SPI-Flash, please use **'Fabric RAMs'** tab to make your selection for each RAM client.

[ ] Start address for sNVM clients: 0x 00000000
[ ] Start address for uPROM clients: 0x 00000000
[ ] Start address for SPI-Flash clients: 0x 00000400

SPI-Flash Binding: SPI-Flash - No-binding Plaintext
SPI Clock divider value: 2(40 MHz)

Time Out (s): 128
Auto Calibration Time Out (ms): 3000
Custom configuration file:

After completing the configuration, perform the following steps to program the initialization data:

- Generate initialization clients
- Generate or export the bit stream
- Program the device

For detailed information on how to use this tool, see Libero SoC Design Flow User Guide. Fore more information on the Tcl commands used to configure various tabs in the tool and specify memory configuration files (`*.cfg`), see Tcl Commands Reference Guide.

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

6. Building Your Firmware Project

Note: This chapter applies to SmartFusion2 only.

This section describes how to build your firmware project when using the Custom Flow. Three types of files that make up a firmware project:

- Source files (your firmware application)
- Drivers: These are drivers provided to facilitate your use of SmartFusion2 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:

Build your firmware project as follows:

Note:
Steps 1- 4 are detailed in the SmartFusion2 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 do.
5. Create a directory in your firmware project.

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

6. For each Component Manifests Report (generated in 2. Component Configuration):
   – Import each file in the **Firmware files for all Software IDE tools** section (Figure 2-2) 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
```

7. Appendix A—Libero-Generated Hardware Configuration Files

This appendix describes the hardware configuration files that Libero generate. These files are intended to be imported into a firmware project. For more information, see 6. Building Your Firmware Project. Depending on the components present in a design, not all of these files are present.

sys_config.h	This header file contains information about the SmartFusion2 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.
sys_config.c	This C source file contains information about the SmartFusion2 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.
sys_config_mss_clocks.h	This header file contains information about the SmartFusion2 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.
sys_config_mddr_define.h	This header file contains information about the SmartFusion2 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.
sys_config_SERDESIF_<0-3>.c	These C source files contain information about the SmartFusion2 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.
sys_config_SERDESIF_<0-3>.h	These header files contain information about the SmartFusion2 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.

8. Appendix B—Sample SDC and PDC Constraints

For certain IP cores such as CCC, OSC, CoreResetP and CoreConfigP, Libero SoC generates SDC, and PDC timing constraints. Passing the SDC and/or PDC constraints to design tools increases the chance of meeting timing closure with less effort and fewer design iterations. The full hierarchical path from the top-level instance is given for all design objects referenced in the constraints.

8.1 SDC Timing Constraints

In the Libero IP core reference project, this top-level SDC constraint file is available from the Constraint Manager (`Design Flow > Open Manage Constraint View >Timing > Derive Constraints`).
Note: See this file to set the SDC constraints if your design contains CCC, OSC, CoreResetP, and CoreConfigP components. Modify the full hierarchical path, if necessary, to match your design hierarchy or use the Derive_Constraints utility and steps in 10. Appendix D—Derive Constraints on the component level SDC file. Save the file to a different name and import the SDC file to the synthesis tool, Place and Route Tool, and Timing Verifications, just like any other SDC constraint files.

8.1.1 Derived SDC file

```
#Libero SoC uses "/" as the hierarchy separator and pin separators in the *.sdc file
create_clock -name {<top_level_instance_name>/FABOSC_0/I_RCOSC_25_50MHZ/CLKOUT}\
-period 20 \
[get_pins \{<top_level_instance_name>/FABOSC_0/I_RCOSC_25_50MHZ/CLKOUT}]
create_clock -name {<top_level_instance_name>/mddr_top_sb_MSS_0/CLK_CONFIG_APB} \
-period 40 \ [get_pins {\
<top_level_instance_name>/mddr_top_sb_MSS_0/MSS_ADLIB_INST/CLK_CONFIG_APB}]
create_generated_clock -name {<top_level_instance_name>/CCC_0/GL0}\
-multiply_by 4 -divide_by 2 \
-source [get_pins {<top_level_instance_name>/CCC_0/CCC_INST/RCOSC_25_50MHZ}]\
-phase 0 \
[get_pins {<top_level_instance_name>/CCC_0/CCC_INST/GL0}]
set_false_path -ignore_errors -through [get_nets {\
<top_level_instance_name>/CORECONFIGP_0/INIT_DONE\
<top_level_instance_name>/CORECONFIGP_0/SDIF_RELEASED}]
set_false_path -ignore_errors -through [get_nets {\
<top_level_instance_name>/CORERESETP_0/ddr_settled \
<top_level_instance_name>/CORERESETP_0/count_ddr_enable\
<top_level_instance_name>/CORERESETP_0/release_sdif*_core\
<top_level_instance_name>/CORERESETP_0/count_sdif*_enable}]
set_false_path -ignore_errors -from [get_cells {\
<top_level_instance_name>/CORERESETP_0/MSS_HPMS_READY_int}] -to [get_cells {
<top_level_instance_name>/CORERESETP_0/sm0_areset_n_rcosc\
<top_level_instance_name>/CORERESETP_0/sm0_areset_n_rcosc_q1}]
set_false_path -ignore_errors -from [get_cells {\
<top_level_instance_name>/CORERESETP_0/MSS_HPMS_READY_int\
<top_level_instance_name>/CORERESETP_0/SDIF*_PERST_N_re}] -to [get_cells {\
<top_level_instance_name>/CORERESETP_0/sdif*_areset_n_rcosc*}]
set_false_path -ignore_errors -through [get_nets {\
<top_level_instance_name>/CORERESETP_0 CONFIG1_DONE\
<top_level_instance_name>/CORERESETP_0/CONFIG2_DONE\
<top_level_instance_name>/CORERESETP_0/SDIF*_PERST_N \
<top_level_instance_name>/CORERESETP_0/SDIF*_PSEL\
<top_level_instance_name>/CORERESETP_0/SDIF*_PWRITE\
<top_level_instance_name>/CORERESETP_0/SDIF*_PRDATA[*]\
<top_level_instance_name>/CORERESETP_0/SOFT_EXT_RESET_OUT \
<top_level_instance_name>/CORERESETP_0/SOFT_RESET_F2M\
<top_level_instance_name>/CORERESETP_0/SOFT_M3_RESET \
<top_level_instance_name>/CORERESETP_0/SOFT_MDDR_DDR_AXI_S_CORE_RESET \
<top_level_instance_name>/CORERESETP_0/SOFT_FDDR_CORE_RESET\
<top_level_instance_name>/CORERESETP_0/SOFT_SDIF*_PHY_RESET \
<top_level_instance_name>/CORERESETP_0/SOFT_SDIF*_CORE_RESET \
<top_level_instance_name>/CORERESETP_0/SOFT_SDIF0_0_CORE_RESET\
<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}]
```

8.2 PDC Physical Design Constraints

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`).

Note: 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 10. 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.

8.2.1 Derived PDC file

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}
```

9. Appendix C—Importing Simulation Libraries into Simulation Environment

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

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

Similar to 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 `vlib` command
   `vlib presynth.`
2. Map the logical library name to pre-compiled library directory using `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 `vsim` command by specifying name of any top-level module.
5. Simulate the design using run command.

After loading the design, simulation time is set to zero, and enter run command to begin simulation.

In the simulator transcript window, execute `run.do` file as do `run.do` to run the simulation. Sample `run.do` file as follows.

```
quietly set ACTELLIBNAME PolarFire
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 PolarFire "X:/Libero/Designer/lib/modelsimpro/precompiled/vlog/PolarFire"

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 1ps presynth.tb
add wave /tb/*
run 1000ns
log /tb/*
exit
```

10. Appendix D—Derive Constraints

This appendix describes the Derive Constraints Tcl commands.

10.1 Derive Constraints Tcl Commands

The `derive_constraints` utility helps you `Derive Constraints` from the RTL or the Configurator outside of the Libero SoC design environment. To run the `derive_constraints` utility supply path to the Tcl file as a command-line argument. For example:

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

The Tcl file supplied to the `derive_constraints` utility should specify the following information in specified order.

1. Device information (10.1.7 set_device).
2. RTL files (10.1.5 read_verilog/10.1.6 read_vhdl).
3. Top level module
   (10.1.8 set_top_level).
4. Component SDC/NDC files
   (10.1.3 read_sdc/10.1.4 read_ndc).
5. derive_constraint command
   (10.1.2 derive_constraints).
6. Where to generate SDC/PDC/NDC derived constraints file (10.1.11 write_sdc/10.1.9 write_pdc/ 10.1.10 write_ndc).

The following is a Tcl file example using the Derive Constraints Tcl commands.

```
# Device information
set_device -family PolarFire -die MPF100T -speed -1

# RTL files
read_verilog -mode system_verilog project/component/work/txpll0/txpll0_txpll0_0_PF_TX_PLL.v
read_verilog -mode system_verilog {project/component/work/txpll0/txpll0.v}
read_verilog -mode system_verilog {project/component/work/xcvr0/I_XCVR/xcvr0_I_XCVR_PF_XCVR.v}
read_verilog -mode system_verilog {project/component/work/xcvr0/xcvr0.v}
read_vhdl -mode vhdl_2008 {project/hdl/xcvr1.vhd}

#Component SDC files
set_top_level {xcvr1}
read_sdc -component {project/component/work/txpll0/txpll0_0/txpll0_txpll0_0_PF_TX_PLL.sdc}
read_sdc -component {project/component/work/xcvr0/I_XCVR/xcvr0_I_XCVR_PF_XCVR.sdc}

#Use derive_constraint command
derive_constraints

#SDC/PDC/NDC result files
write_sdc {project/constraint/xcvr1_derived_constraints.sdc}
write_pdc {project/constraint/fp/xcvr1_derived_constraints.pdc}
```

10.1.1 add_include_path

Description

Specifies a path to search for includes when reading RTL files.

```
add_include_path <directory>
```

Arguments

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

Return Type	Description
0	Command succeeded.
1	Command does not succeeded. There is an error and user can observer the error message in console.

List of Errors

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

Note: 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 errors.

Supported Families

PolarFire
PolarFire SoC
RTG4
SmartFusion2
IGLOO2

Example

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

10.1.2 derive_constraints

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

```
derive_constraints
```

Arguments

Return Type	Description
0	Command succeeded.
1	Command does not succeeded. There is an error and user can observer the error message in console.

List of Errors

Error Message	Description
Top-level is not defined	This means that the top-level module or entity is not specified. To fix this call `set_top_level` command before `derive_constraints` command call.

Supported Families

PolarFire
PolarFire SoC
RTG4
SmartFusion2
IGLOO2

Example

```
derive_constraints
```

10.1.3 read_sdc

Description
Read a SDC file into the component database.

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

Arguments

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

Return Type	Description
0	Command succeeded.
1	Command does not succeeded. There is an error and user can observer the error message in console.

List of Errors

Error Message	Description
Required parameter file name is missing.	The mandatory option file name is not specified.
SDC file <file_path> is not readable.	The specified SDC file does not have read permissions.
Unable to open <file_path> file.	The SDC file does not exist. The path must be corrected.
Missing `set_component` command in <file_path> file	The specified component of SDC file does not specify the component.

...........continued

Error Message	Description
<List of errors from sdc file>	The SDC file contains incorrect sdc commands. Example when there is an error in `set_multicycle_path` constraint: Error while executing command `read_sdc: in <sdc_file_path> file:` Error in command `set_multicycle_path: Unknown parameter [get_cells {reg_a}]`

Supported Families

PolarFire
PolarFire SoC
RTG4
SmartFusion2
IGLOO2

Example

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

10.1.4 read_ndc

Description

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

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

Arguments

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

Return Type	Description
0	Command succeeded.
1	Command does not succeeded. There is an error and user can observer the error message in console.

List of Errors

Error Message	Description
Unable to open <file_path> file	The NDC file does not exist. The path must be corrected.
Required parameter—AtclParam0_ is missing.	The mandatory option filename is not specified.
Required parameter—component is missing	Component option is mandatory and must be specified.

...........continued

Error Message	Description
NDC file '<file_path>' is not readable.	The specified NDC file does not have read permissions.

Supported Families

PolarFire
PolarFire SoC
RTG4
SmartFusion2
IGLOO2

Example

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

10.1.5 read_verilog

Description
Read a Verilog file using Verific.

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

Arguments

Parameter	Type	Description
-lib <libname>	String	Specify the library to add the modules into 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 does not succeeded. There is an error and user can observer the error message in console.

List of Errors

Error Message	Description
Parameter—lib is missing value	The lib option is specified without value.
Parameter—mode is missing value	The mode option is specified without value.
Unknown mode '<mode>'	The specified verilog mode is unknown. See the list of possible verilog mode in—mode option description.

...........continued

Error Message	Description
Required parameter file name is missing	No verilog file path is provided.
Failed due to Verific errors	Syntax error in verilog file. Verific errors can be observed in the console above the error message.
set_device is not called	The device information is not specified. Use `set_device` command to describe the device.

Supported Families

PolarFire
PolarFire SoC
RTG4
SmartFusion2
IGLOO2

Example

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

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

10.1.6 read_vhdl

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 does not succeeded. There is an error and user can observer the error message in console.

List of Errors

Error Message	Description
Parameter—lib is missing value	The lib option is specified without value.
Parameter—mode is missing value	The mode option is specified without value.
Unknown mode '<mode>'	The specified VHDL mode is unknown. See the list of possible VHDL mode in—mode option description.
Required parameter file name is missing	No VHDL file path is provided.
Unable to register `invalid_path.v` file	The specified VHDL file does not exist or does not have read permissions.
`set_device` is not called	The device information is not specified. Use `set_device` command to describe the device.

Supported Families

PolarFire
PolarFire SoC
RTG4
SmartFusion2
IGLOO2

Example

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

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

10.1.7 set_device

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 PolarFire, PolarFire SoC, IGLOO2, SmartFusion2, 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.
1	Command does not succeeded. There is an error and user can observer the error message in console.

List of Errors

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

Supported Families

PolarFire
PolarFire SoC
RTG4
SmartFusion2
IGLOO2

Example

```
set_device -family {PolarFire} -die {MPF300T_ES} -speed -1
```

```
set_device -family SmartFusion2 -die M2S090T -speed -1
```

10.1.8 set_top_level

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 does not succeeded. There is an error and user can observer the error message in console.

List of Errors

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

Supported Families

PolarFire
PolarFire SoC
RTG4
SmartFusion2
IGLOO2

Example

```
set_top_level {top}
```

```
set_top_level -lib hdl top
```

10.1.9 write_pdc

Description

Write physical constraints (Derive Constraints only).

```
write_pdc <filename>
```

Arguments

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

Return Type	Description
0	Command succeeded.
1	Command does not succeeded. There is an error and user can observer the error message in console.

List of Errors

Error Messages	Description
Unable to open <file path> file	The file path is not correct. May be the parent directories do not exist.
PDC file '<file path>' is not writeable.	The specified PDC file does not have write permission.
Required parameter file name is missing	The PDC file path is mandatory option and needs to be specified.

Supported Families

PolarFire
PolarFire SoC
RTG4
SmartFusion2
IGLOO2

Example

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

10.1.10 write_ndc

Description
Write NDC constraints into a file. The command can be 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 mandatory option. If the file already exists it will be overwritten.

Return Type	Description
0	Command succeeded.
1	Command does not succeeded. There is an error and user can observer the error message in console.

List of Errors

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

Supported Families

PolarFire
PolarFire SoC
RTG4
SmartFusion2
IGLOO2

Example

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

10.1.11 write_sdc

Description
Write 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 mandatory option. If the file already exists it will be overwritten.

Return Type	Description
0	Command succeeded.
1	Command does not succeeded. There is an error and user can observer the error message in console.

List of Errors

Error Message	Description
Unable to open <file path> file.	File path is not correct. May be the parent directories do not exist.
SDC file '<file path>' is not writable.	The specified SDC file does not have write permission.
Required parameter file name is missing.	The SDC file path is mandatory option and needs to be specified.

Supported Families

PolarFire
PolarFire SoC
RTG4
SmartFusion2
IGLOO2

Example

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

11. Revision History

Revision	Date	Description
B	08/2021	• Updated Figure 1-1. • Added 10. 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**

The Microchip Website

Microchip provides online support via our website at www.microchip.com/. This website is used to make files and information easily available to customers. Some of the content available includes:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQs), technical support requests, online discussion groups, Microchip design partner program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

Product Change Notification Service

Microchip's product change notification service helps keep customers current on Microchip products. Subscribers will receive email notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, go to www.microchip.com/pcn and follow the registration instructions.

Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Embedded Solutions Engineer (ESE)
- Technical Support

Customers should contact their distributor, representative or ESE for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in this document.

Technical support is available through the website at: www.microchip.com/support

Microchip Devices Code Protection Feature

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

- 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 and under normal conditions.

Legal Notice

Trademarks

ISBN: 978-1-5224-8795-1

Quality Management System

For information regarding Microchip's Quality Management Systems, please visit www.microchip.com/quality.

Worldwide Sales and Service

AMERICAS	ASIA/PACIFIC	ASIA/PACIFIC	EUROPE
Corporate Office	**Australia - Sydney**	**India - Bangalore**	**Austria - Wels**
2355 West Chandler Blvd.	Tel: 61-2-9868-6733	Tel: 91-80-3090-4444	Tel: 43-7242-2244-39
Chandler, AZ 85224-6199	**China - Beijing**	**India - New Delhi**	Fax: 43-7242-2244-393
Tel: 480-792-7200	Tel: 86-10-8569-7000	Tel: 91-11-4160-8631	**Denmark - Copenhagen**
Fax: 480-792-7277	**China - Chengdu**	**India - Pune**	Tel: 45-4485-5910
Technical Support:	Tel: 86-28-8665-5511	Tel: 91-20-4121-0141	Fax: 45-4485-2829
www.microchip.com/support	**China - Chongqing**	**Japan - Osaka**	**Finland - Espoo**
Web Address:	Tel: 86-23-8980-9588	Tel: 81-6-6152-7160	Tel: 358-9-4520-820
www.microchip.com	**China - Dongguan**	**Japan - Tokyo**	**France - Paris**
Atlanta	Tel: 86-769-8702-9880	Tel: 81-3-6880- 3770	Tel: 33-1-69-53-63-20
Duluth, GA	**China - Guangzhou**	**Korea - Daegu**	Fax: 33-1-69-30-90-79
Tel: 678-957-9614	Tel: 86-20-8755-8029	Tel: 82-53-744-4301	**Germany - Garching**
Fax: 678-957-1455	**China - Hangzhou**	**Korea - Seoul**	Tel: 49-8931-9700
Austin, TX	Tel: 86-571-8792-8115	Tel: 82-2-554-7200	**Germany - Haan**
Tel: 512-257-3370	**China - Hong Kong SAR**	**Malaysia - Kuala Lumpur**	Tel: 49-2129-3766400
Boston	Tel: 852-2943-5100	Tel: 60-3-7651-7906	**Germany - Heilbronn**
Westborough, MA	**China - Nanjing**	**Malaysia - Penang**	Tel: 49-7131-72400
Tel: 774-760-0087	Tel: 86-25-8473-2460	Tel: 60-4-227-8870	**Germany - Karlsruhe**
Fax: 774-760-0088	**China - Qingdao**	**Philippines - Manila**	Tel: 49-721-625370
Chicago	Tel: 86-532-8502-7355	Tel: 63-2-634-9065	**Germany - Munich**
Itasca, IL	**China - Shanghai**	**Singapore**	Tel: 49-89-627-144-0
Tel: 630-285-0071	Tel: 86-21-3326-8000	Tel: 65-6334-8870	Fax: 49-89-627-144-44
Fax: 630-285-0075	**China - Shenyang**	**Taiwan - Hsin Chu**	**Germany - Rosenheim**
Dallas	Tel: 86-24-2334-2829	Tel: 886-3-577-8366	Tel: 49-8031-354-560
Addison, TX	**China - Shenzhen**	**Taiwan - Kaohsiung**	**Israel - Ra'anana**
Tel: 972-818-7423	Tel: 86-755-8864-2200	Tel: 886-7-213-7830	Tel: 972-9-744-7705
Fax: 972-818-2924	**China - Suzhou**	**Taiwan - Taipei**	**Italy - Milan**
Detroit	Tel: 86-186-6233-1526	Tel: 886-2-2508-8600	Tel: 39-0331-742611
Novi, MI	**China - Wuhan**	**Thailand - Bangkok**	Fax: 39-0331-466781
Tel: 248-848-4000	Tel: 86-27-5980-5300	Tel: 66-2-694-1351	**Italy - Padova**
Houston, TX	**China - Xian**	**Vietnam - Ho Chi Minh**	Tel: 39-049-7625286
Tel: 281-894-5983	Tel: 86-29-8833-7252	Tel: 84-28-5448-2100	**Netherlands - Drunen**
Indianapolis	**China - Xiamen**		Tel: 31-416-690399
Noblesville, IN	Tel: 86-592-2388138		Fax: 31-416-690340
Tel: 317-773-8323	**China - Zhuhai**		**Norway - Trondheim**
Fax: 317-773-5453	Tel: 86-756-3210040		Tel: 47-72884388
Tel: 317-536-2380			**Poland - Warsaw**
Los Angeles			Tel: 48-22-3325737
Mission Viejo, CA			**Romania - Bucharest**
Tel: 949-462-9523			Tel: 40-21-407-87-50
Fax: 949-462-9608			**Spain - Madrid**
Tel: 951-273-7800			Tel: 34-91-708-08-90
Raleigh, NC			Fax: 34-91-708-08-91
Tel: 919-844-7510			**Sweden - Gothenberg**
New York, NY			Tel: 46-31-704-60-40
Tel: 631-435-6000			**Sweden - Stockholm**
San Jose, CA			Tel: 46-8-5090-4654
Tel: 408-735-9110			**UK - Wokingham**
Tel: 408-436-4270			Tel: 44-118-921-5800
Canada - Toronto			Fax: 44-118-921-5820
Tel: 905-695-1980			
Fax: 905-695-2078			