SmartFusion2, IGLOO2, and RTG4 Custom Flow User Guide

Designing Using External Tools for Synthesis and Simulation

Microsemi

Power Matters.™

Table of Contents

Introduction

Microsemi's Libero SoC software provides a fully integrated FPGA design environment. However, some users may want to use third-party Synthesis and Simulation tools outside the Libero SoC environment. Until now, you were required to use the Libero end-to-end flow for SmartFusion2, IGLOO2, and RTG4 designs. Libero can now be integrated in your own FPGA design environment. While it is recommended that you use Libero SoC to manage your entire FPGA design flow whenever possible, you are no longer required to do so.

This user guide describes the Custom Flow, a process whereby you can integrate Libero as a part of your larger FPGA design flow. This document includes the following sections:

- Overview of the Custom Flow
- Component configuration
- Synthesis
- Simulation
- Firmware generation
- Implementing your final project using Libero

1 – Custom Flow Overview

While Libero SoC provides a fully-integrated end-to-end design environment to develop System-on-Chip designs, it does provide the users the flexibility to run synthesis and simulation with third-party tools outside the Libero SoC environment. Some design steps must, however, stay inside the Libero SoC environment.

Table 1-1 shows 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?**	
Design Entry: HDL	No	Use 3rd-party design-entry tool outside Libero SoC if the user so desires.
Design Entry: Configurators	Yes	Create First Libero Project for Design Generation.
Design Entry: System Builder (SmartFusion2 and IGLOO2 only)	Yes	Stay in First Libero Project. Close project after Design Generation
Automatic PDC/SDC constraint generation	Yes	Stay in First Libero Project (Available only with Libero Enhanced Constraint Flow)
Simulation*	No	Use 3rd-party tool outside Libero SoC if the user so desires.
Synthesis	No	Use 3rd-party tool outside Libero SoC if the user so desires.
One of two ways in Design Implementation • Classic Constraint Flow (Compile, Place and Route). See Figure 1-2 • Enhanced Constraint Flow (Manage Constraints, Compile Netlist, Place and Route) See Figure 1-3	Yes	Create Second Libero Project for the backend implementation.
Timing and Power Verification	Yes	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 3rd-party tool outside Libero SoC if the user so desires.

***Note**: You must download precompiled libraries from here to use a third party simulator.

In a CPLD/pure Fabric FPGA flow, you enter your design using HDL or schematic entry, and pass that directly to your Synthesis tools. This is still supported. However, SmartFusion2, IGLOO2, and RTG4 FPGAs have significant proprietary System on Chip (SoC) functionality. Special handling is required for any blocks that comprise SoC functionality; these blocks are hereinafter referred to as SoC Components. These are:

- SmartFusion2 and IGLOO2
  - SmartFusion2 MSS (includes MDDR and eNVM)
  - IGLOO2 HPMS (includes MDDR and eNVM)
  - System Builder
  - FDDR
  - SERDES
  - Oscillator
  - CCC
- RTG4
  - uPROM
  - CCC
  - SERDES
  - SERDES INIT
  - FDDR
  - FDDR INIT
  - RCOSC macro

If you are using any of the above components in your design, and you want to manage the FPGA design flow outside of Libero, follow the steps provided in the rest of this guide.

Component Lifecycle

The following steps describe the lifecycle of an SoC Component and provide instructions on how to 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 register configuration files (*.reg) for MDDR/FDDR/SERDES and *cfg file for eNVM (SmartFusion2 and IGLOO2) and uPROM (RTG4)

   e. Firmware drivers (*.h) files
2. For HDL files, instantiate and integrate them in the rest of your HDL design using your 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. 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.

For Design Entry, if you are using SmartFusion2 or IGLOO2 SoC components (i.e., oscillators, CCC, eNVM, MSS (includes MDDR)), HPMS (includes MDDR), FDDR, or SERDES), you must use Libero SoC for part of your Design Entry (Chapter 2, "Component Configuration"), but you can then continue the rest of your Design Entry (HDL entry, and so on) outside of Libero.

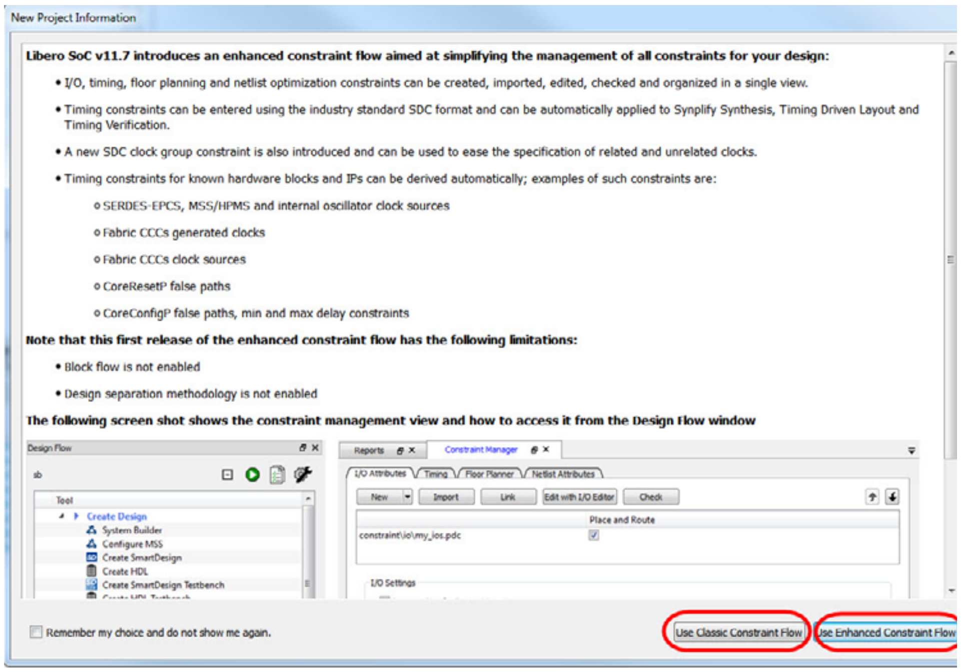
If you are using RTG4 components (i.e., uPROM, CCC, SERDES, SERDES INIT, FDDR, FDDR INIT, RCOSC macro) you must use Libero SoC for part of your Design Entry (Chapter 2, "Component Configuration"), but you can then continue the rest of your Design Entry (HDL entry, and so on) outside of Libero.

Libero SoC Project Creation

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

Classic Constraint Flow Versus Enhanced Constraint Flow

When you create a Libero SoC project for SmartFusion2, IGLOO2, and RTG4 devices, you can choose the Classic Constraint Flow or the Enhanced Constraint Flow.



New Project Information

Libero SoC v11.7 introduces an enhanced constraint flow aimed at simplifying the management of all constraints for your design:

- I/O, timing, floor planning and netlist optimization constraints can be created, imported, edited, checked and organized in a single view.
- Timing constraints can be entered using the industry standard SDC format and can be automatically applied to Synplify Synthesis, Timing Driven Layout and Timing Verification.
- A new SDC clock group constraint is also introduced and can be used to ease the specification of related and unrelated clocks.
- Timing constraints for known hardware blocks and IPs can be derived automatically; examples of such constraints are:
  - SERDES-EPCS, MSS/HPMS and internal oscillator clock sources
  - Fabric CCCs generated clocks
  - Fabric CCCs clock sources
  - CoreResetP false paths
  - CoreConfigP false paths, min and max delay constraints

Note that this first release of the enhanced constraint flow has the following limitations:

- Block flow is not enabled
- Design separation methodology is not enabled

The following screen shot shows the constraint management view and how to access it from the Design Flow window

Figure 1-1 **• Project Creation Wizard - Enhanced Versus Classic Constraint Flow**

Microsemi recommends the Enhanced Constraint Flow be 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 edit of constraints and the association of constraints 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 certain common cores such as the CCC, OSC, CoreResetP (SmartFusion2 and IGLOO2), CoreConfigP (SmartFusion2 and IGLOO2), SERDES_IF, and SmartFusion2 MSS/IGLOO2 HPMS. The SDC or PDC constraints of these cores are set from the top of the design hierarchy with the full hierarchical path given. You don't 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 or PDC constraint such as hierarchy and pin separators, and design object names, etc.
- The automatically generated constraints, when applied, increase the chance of timing closure with less effort and fewer design iterations.

Custom Flow and Libero Classic Constraint Flow

Figure 1-2 shows the Custom Flow overview with Libero SoC Classic Constraint Flow projects.

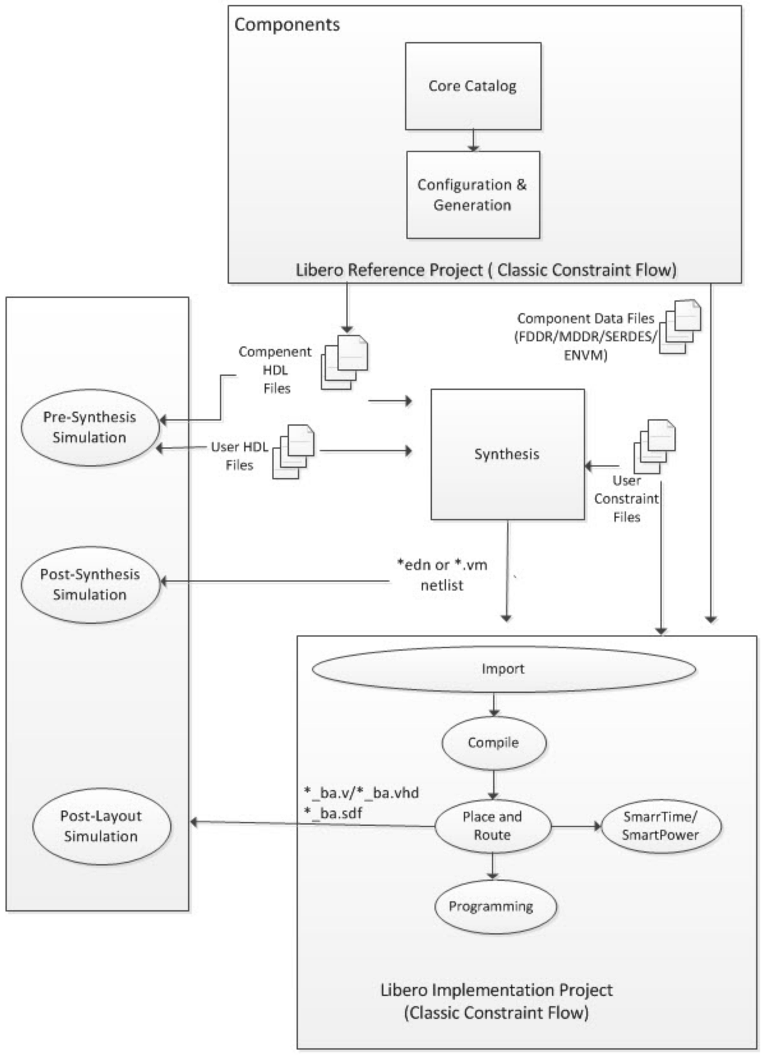


Figure 1-2 • **Custom Flow Overview (Libero Classic Constraint Flow - No automatic SDC and PDC Constraints Generation)**

The following are the steps in the Custom Flow (using Libero Classic Constraint Flow. Figure 1-2):

1. Component configuration and generation:

   a. Create a Libero SoC project (Reference Project). Choose Classic Constraint Flow.

   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 MSS blocks, *g*enerate the component in the SmartDesign canvas after configuration of the MSS block and System Builder block.

   – This automatically exports component data and files.

– A "Component Manifests" file is generated for each design component. The "Component Manifests" lists the name and location of the files to be used for running third-party design tools outside Libero, if the user so desires. See "Component Manifests" section on page 13 for details.

2. Complete your RTL design outside of Libero:

   a. Instantiate component HDL files; the location of the HDL files is noted in the manifest.

3. Synthesis tool/Simulation tool:

   a. Get HDL files, stimulus files, and component data from specific locations as noted in the "Component Manifests".

   b. Synthesize and simulate outside Libero using your third-party synthesis tool and simulator.

4. Firmware tool (SmartFusion2 only):

   a. Get drivers from specific locations as noted in the "Component Manifests"

   b. Edit source code to enable runtime initialization for specific components

   c. Compile firmware project

5. Create your second (Implementation) Libero Project. Choose Classic Constraint Flow.

6. Remove Synthesis from the design flow tool chain (**Project > Project Settings > Design Flow >** clear the **Enable Synthesis** checkbox)

7. Import Design Sources (Either *edn or *.vm format)

   – Post-synthesis *.edn netlist (**File > Import > Others**) or

   – Post-synthesis *.vm netlist (**File > Import > HDL Source Files**)

   – Component metadata such as *.reg file for FDDR/MDDR, SERDES (SmartFusion2 and IGLOO2) and *.cfg file for eNVM (SmartFusion2 and IGLOO2), and *.cfg file for uPROM (RTG4).

8. Import the design constraints:

   – File > Import > I/O constraints (PDC) Files

   – File > Import > Floorplanning Constraints (PDC) Files (Refer to "<top_level>_derived_constraints.pdc (Generated in Enhanced Constraint Flow only)") if your design contains CoreConfigP (SmartFusion2 and IGLOO2)

   – File > Import > Timing Constraints (SDC) Files (Refer to "<top_level>_derived_constraints.sdc (Available in Enhanced Constraint Flow only)") if your design contains CCC, OSC, CoreConfigP (SmartFusion2 and IGLOO2) and CoreResetP (SmartFusion2 and IGLOO2), SmartFusion2 MSS/IGLOO2 HPMS, System Builder, and SERDES_IF blocks

   c. Associate the Constraints Files to Compile

   – Right-click the imported SDC/PDC files inside the Design Flow window and marked them as used for (associated with) Compile.

9. Complete Design Implementation

   – Run Compile, Place and Route, Programming File Generation, and so on.

Custom Flow and Libero Enhanced Constraint Flow

Figure 1-3 shows the Custom Flow Overview with Libero Enhanced Constraint Flow projects.

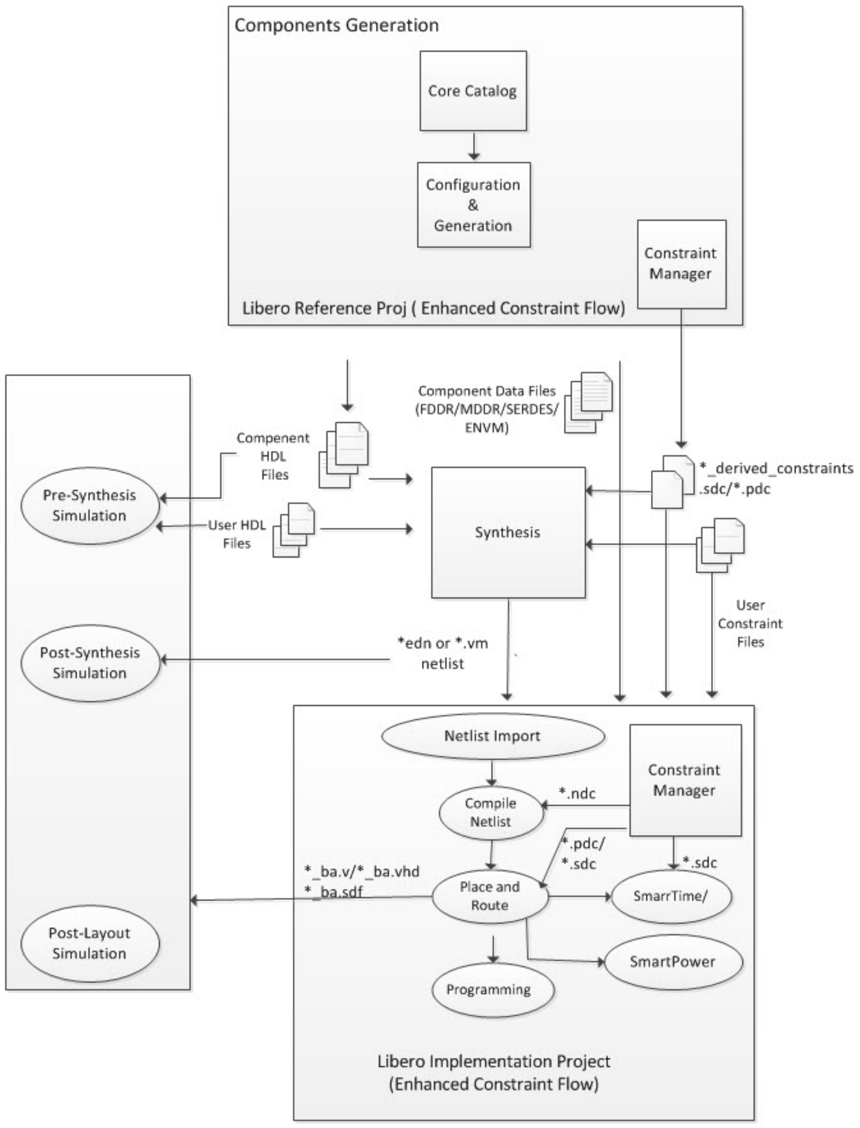


Figure 1-3 • **Custom Flow Overview (Libero Enhanced Constraint Flow - Automatic SDC and PDC Constraints Generation for CCC, OSC, CoreResetP, CoreConfigP, SERDES, and SmartFusion2 MSS/IGLOO2 HPMS IP Cores)**

The following are the steps in the Custom Flow (using Libero Enhanced Constraint Flow. Figure 1-3):

1. Component configuration and generation:

   a. Create a Libero project (Reference Project), Choose Enhanced Constraint Flow.

   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 MSS blocks, generate the component in the SmartDesign canvas after configuration of the MSS block and System Builder block.

   This automatically exports component data and files

  - A "Component Manifests" is also generated. See "Component Manifests" section on page 13 for details.
2. Generate SDC and PDC constraints for the components
   - Select the top level in the Design Hierarchy and set as Root (RMC > Set as Root)
   - Open the Constraint Manager (**Design Flow Window > Open Manage Constraints View**). Click the **Timing** Tab and then click **Derive Constraints** button to generate the floorplanning *.pdc and the timing *.sdc file.
3. Complete your RTL design outside of Libero:

   a. Instantiate component HDL files. The location of the HDL files is listed in the "Component Manifests" files.
4. Synthesis tool/Simulation tool:

   a. Get HDL files, stimulus files, and component data from 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 specific locations as noted in the manifest

   b. Edit source code to enable runtime initialization for specific components

   c. Compile firmware project
6. Create your second (Implementation) Libero Project. Choose Enhanced Constraint Flow.
7. Remove Synthesis from the design flow tool chain (**Project > Project Settings > Design Flow >** clear the **Enable Synthesis** checkbox)
8. Import the design source files (post-synthesis *.edn or *.vm netlist from Synthesis tool)
   - Post-synthesis *.edn netlist (**File > Import > Others**)
   - Post-synthesis *.vm netlist (**File > Import > HDL Source Files**)
   - Component metadata such as *.reg files for MDDR/FDDR/SERDES (SmartFusion2 and IGLOO2) and *.cfg file for eNVM (SmartFusion2 and IGLOO2), and *.cfg file for uPROM (RTG4).
9. Import the design constraints:
   - Import I/O constraint files (**Constraints Manager > I/O Attributes > Import**).
   - Import floorplanning *.pdc files (**Constraints Manager > Floor Planner > Import**). If your design contains CoreConfigP (SmartFusion2 and IGLOO2), make sure to import the "<top_level>_derived_constraints.pdc (Generated in Enhanced Constraint Flow only)" generated in the first Libero SoC project with the Derived Constraints button ("Generate SDC and PDC constraints for the components" section on page 10)
   - Import *.sdc timing constraint files (**Constraints Manager > Timing > Import**). If your design contains CCC, OSC, CoreConfigP (SmartFusion2 and IGLOO2) and CoreResetP (SmartFusion2 and IGLOO2), SmartFusion2 MSS/IGLOO2 HPMS, and SERDES_IF IP cores, make sure to import the "<top_level>_derived_constraints.sdc (Available in Enhanced Constraint Flow only)" file generated in the first Libero SoC project with the Derived Constraints button ("Generate SDC and PDC constraints for the components" section on page 10).
   - Import *.ndc constraint files (**Constraints Manager > Netlist Attributes > Import**), if any.
10. 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.
11. Complete Design Implementation
    - Run Compile Netlist, Place and Route, Programming File Generation, and so on

2 – Component Configuration

The first step in the Custom Flow is to configure your Components using a Libero project. In subsequent steps, you will use data from this reference project.

If you are using any of the following components in your design, you must perform the steps described in this section:

- SmartFusion2 and IGLOO2
  - SmartFusion2 MSS
  - IGLOO2 HPMS
  - MDDR
  - FDDR
  - SERDES
  - eNVM
  - CCC
- RTG4
  - uPROM
  - CCC
  - SERDES
  - SERDES INIT
  - FDDR
  - FDDR INIT
  - RCOSC macro

If you are not using any of the above components, you can write your RTL outside of Libero, and import it into your Synthesis and Simulation tools directly. You can then proceed to the post-synthesis section and only import your post-synthesis *.edn or *.vm netlist into your final Libero project.

Component Configuration Using Libero

After you have selected the Components you are going to use from the above list, perform the following steps:

1. Create a new Libero project (Core Configuration and Generation):

   a. Select the Device and Family that you are targeting your final design to.

   b. If you are using the SmartFusion2 MSS, or if you would like to use System Builder, make the appropriate selection in the “Use Design Flow” section of the New Project Window. Microsemi recommends using System Builder (for both SmartFusion2 and IGLOO2) to configure any of the above Components.
2. If you are using 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. Refer to the SmartFusion2 System Builder User's Guide or the IGLOO2 System Builder User's Guide for details.

   b. Generate your system in System Builder, and promote all its ports to the top level (select all ports, right-click, and choose **Promote to Top**).

   - Note the port names—you will 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 (any one of Pre-Synthesis or Post-Synthesis or Post-Layout options) to invoke the Simulator. You can exit the simulator once it is invoked—this step will generate the simulation files necessary for your project.

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

f.Save your project—this is your Reference Project.

3. If you are not using System Builder (SmartFusion2 only):

a. If you select the SmartFusion2 MSS in the “Use Design Flow” subsection (step 1b above), the SmartFusion2 MSS Configurator will automatically open. Otherwise, in the Design Flow window, double-click **Configure MSS**.

– Libero automatically creates a new SmartDesign for you, in which the MSS will be instantiated. Double-click the MSS instance to open its configurator.

– Configure the SmartFusion2 MSS as per your requirements. Refer to the SmartFusion2 MSS Users Guides for details.

Note: If you are using the eNVM or the MDDR, you must use the MSS to configure it

– Save and generate the MSS component

Note: If you are not using System Builder, and you have MSS (using MDDR) or FDDR or SERDES blocks in your design, you must

a. Construct the Peripheral Initialization architecture in your final design. Refer to the SmartFusion2 Peripheral Initialization User's Guide or the IGLOO2 Peripheral Initialization User's Guide for details.

b. 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.

c. Generate all SmartDesigns built in steps a and b above.

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 simply generates the simulation files necessary for your project.

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

e. Skip to Step 4 for Classic Constraint Flow. For Enhanced Constraint Flow, Open the Constraint Manager (**Design Flow** window > **Manage Constraints > Open Manage Constraints View).** Click the Timing Tab and click Derived Constraints. Click **Yes** when asked whether you want to associate the *.sdc constraint file to Synthesis, Place and Route, and Timing Verifications and associate the *.pdc file to Place and Route. A <top_level>_derived_constraints.sdc file is generated for timing constraints. This file is located in <proj_location/constraints/<top_level>/derived_constraints.sdc.

A <top_level>_derived_constraints.pdc file is generated for floorplanning constraints. This file is located in <proj_location/constraints/fp/<top_level>_derived_constraints.pdc. Note the location of this file. You need to pass this *.sdc file to Synthesis when you exit Libero and run Synthesis outside Libero. For the floorplanning PDC constraint file, you need to pass it to Place and Route in the second Libero Project you will create to implement your design.

f.Save your project—this is your Reference Project.

4. If you are using SmartFusion2, and using any of the MSS peripherals (MDDR, FDDR, or SERDES), you must export your firmware project (SoftConsole/IAR/Keil) from this Libero project. Refer to Chapter 6, "Building Your Firmware Project" for more details.

5. If you are using RTG4:

a. If you want to use SERDES and 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 SERDES and 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 SERDES and FDDR blocks in your design, but not their corresponding INIT cores, you must:

c. Construct the Peripheral Initialization architecture in your final design to initialize RTG4 SERDES and FDDR blocks.

d. 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.

e. If you want to use RTG4 uPROM, add the uPROM block to the top level SmartDesign.

f. Generate all SmartDesigns built in the above steps.

g. Double-click **Simulate** (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 just generates the simulation files necessary for your project.

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

h. For Enhanced Constraint Flow, open the Constraint Manager (**Design Flow window > Manage Constraints > Open Manage Constraints View**). Click the **Timing** tab and click **Derived Constraints**. Click **Yes** when asked whether you want to associate the *.sdc constraint file to Synthesis, Place and Route, and Timing Verification. A <top_level>_derived_constraints.sdc file is generated for timing constraints. This file is located in <proj_location>/constraints/<top_level>/derived_constraints.sdc.

You need to pass this *.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.

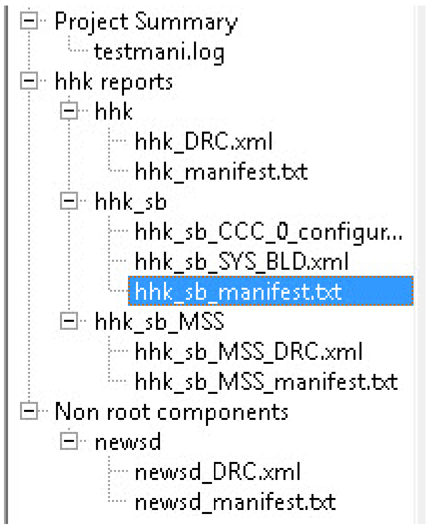
i. Save your project — this is your Reference Project.

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.

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 will see a set of manifest.txt files (Figure 2-1), one for each component you generated. Focus on the following Component Manifest Reports:

- If you are using System Builder, read the file <system builder name>_sb_manifest.txt.
- If you instantiated components into a SmartDesign, read the file <smartdesign_name>_manifest.txt.
- If you created components for cores (normal or RTG4 INIT cores), read the <core_component_name>_manifest.txt.



```
Project Summary
    testmani.log
hhk reports
    hhk
        hhk_DRC.xml
        hhk_manifest.txt
    hhk_sb
        hhk_sb_CCC_0_configur...
        hhk_sb_SYS_BLD.xml
        hhk_sb_manifest.txt
    hhk_sb_MSS
        hhk_sb_MSS_DRC.xml
        hhk_sb_MSS_manifest.txt
Non root components
    newsd
        newsd_DRC.xml
        newsd_manifest.txt
```

***Figure 2-1* • <smartdesign_name>_manifest.txt File**

In this project (Figure 2-1), a System Builder component (hhk_sb) instantiates MSS and FDDR, and a separate SmartDesign (newsd) instantiates two SERDES blocks. The relevant files to focus on for this design are hhk_sb_manifest.txt and newsd_manifest.txt.

You must use all "Component Manifests" Reports that apply to your design. For example, if you instantiate two SmartDesigns, one with a SERDES block and one with an MSS, you must select files from both "Component Manifests" Reports for use in your design flow.

Interpreting Manifest Files

When you open a manifest file (in the Reports Tab), you will see paths to files in your Libero project, and pointers on where in the design flow to use them. You may see the following types of files in a manifest file:

- HDL source files for all Synthesis and Simulation tools
- HDL source files for Synopsys SynplifyPro Synthesis tool
- HDL source files for Mentor Precision 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

```
HDL source files for all Synthesis and Simulation tools:
C:/temp/testmani/component/Actel/DirectCore/CoreAHBLite/5.2.100/rtl/vlog/core/coreahblite.v
C:/temp/testmani/component/Actel/DirectCore/CoreAHBLite/5.2.100/rtl/vlog/core/coreahblite_addrdec.
v
C:/temp/testmani/component/Actel/DirectCore/CoreAHBLite/5.2.100/rtl/vlog/core/coreahblite_defaults
lavesm.v
C:/temp/testmani/component/Actel/DirectCore/CoreAHBLite/5.2.100/rtl/vlog/core/coreahblite_masterst
age.v
C:/temp/testmani/component/Actel/DirectCore/CoreAHBLite/5.2.100/rtl/vlog/core/coreahblite_matrix4x
16.v

C:/temp/testmani/component/Actel/DirectCore/CoreAHBLite/5.2.100/rtl/vlog/core/coreahblite_slavearb
iter.v
C:/temp/testmani/component/Actel/DirectCore/CoreAHBLite/5.2.100/rtl/vlog/core/coreahblite_slavesta
ge.v
C:/temp/testmani/component/Actel/DirectCore/CoreConfigP/7.0.105/rtl/vlog/core/coreconfigp.v
C:/temp/testmani/component/Actel/DirectCore/CoreResetP/7.0.104/rtl/vlog/core/coreresetp.v
C:/temp/testmani/component/Actel/DirectCore/CoreResetP/7.0.104/rtl/vlog/core/coreresetp_pcie_hotre
set.v
C:/temp/testmani/component/work/hhk_sb/CCC_0/hhk_sb_CCC_0_FCCC.v
C:/temp/testmani/component/work/hhk_sb/FABDDR_0/hhk_sb_FABDDR_0_FDDRC.v
C:/temp/testmani/component/work/hhk_sb/FABOSC_0/hhk_sb_FABOSC_0_OSC.v
C:/temp/testmani/component/work/hhk_sb/hhk_sb.v
C:/temp/testmani/component/work/hhk_sb_MSS/hhk_sb_MSS.v

HDL source files for Synopsys SynplifyPro Synthesis tool:
C:/temp/testmani/component/Actel/SgCore/OSC/1.0.105/osc_comps.v
C:/temp/testmani/component/work/hhk_sb/FABDDR_0/hhk_sb_FABDDR_0_FDDRC_syn.v
C:/temp/testmani/component/work/hhk_sb_MSS/hhk_sb_MSS_syn.v

HDL source files for Mentor Precision Synthesis tool:
C:/temp/testmani/component/Actel/SgCore/OSC/1.0.105/osc_comps_pre.v
C:/temp/testmani/component/work/hhk_sb/FABDDR_0/hhk_sb_FABDDR_0_FDDRC_pre.v
C:/temp/testmani/component/work/hhk_sb_MSS/hhk_sb_MSS_pre.v

Stimulus files for all Simulation tools:
C:/temp/testmani/component/Actel/SmartFusion2MSS/MSS/1.1.400/peripheral_init.bfm
C:/temp/testmani/component/work/hhk_sb/subsystem.bfm
C:/temp/testmani/component/work/hhk_sb_MSS/CM3_compile_bfm.tcl
C:/temp/testmani/component/work/hhk_sb_MSS/test.bfm
C:/temp/testmani/component/work/hhk_sb_MSS/user.bfm

Firmware files for all Software IDE tools:
C:/temp/testmani/component/work/hhk_sb/FABDDR_0/sys_config_fddr_define.h
C:/temp/testmani/component/work/hhk_sb_MSS/sys_config_mddr_define.h
C:/temp/testmani/component/work/hhk_sb_MSS/sys_config_mss_clocks.h

Configuration files to be used for Programming:

C:/temp/testmani/component/work/hhk_sb_MSS/ENVM.cfg
Configuration files to be used for all Simulation tools:
C:/temp/testmani/component/work/hhk_sb/FABDDR_0/FDDR_init.bfm
C:/temp/testmani/component/work/hhk_sb_MSS/ENVM.cfg
C:/temp/testmani/component/work/hhk_sb_MSS/MDDR_init.bfm

Configuration files to be used for Power Analysis:
C:/temp/testmani/component/work/hhk_sb/FABDDR_0/FDDR_init.reg
C:/temp/testmani/component/work/hhk_sb_MSS/MDDR_init.reg
```

***Figure 2-2* • Example Component Manifest Report - System Builder Block for SmartFusion2, with MDDR, FDDR, and eNVM**

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 <proj_location>/<top_level>_derived_constraints.sdc and the <proj_location>/<top_level>_derived_constraints.pdc file are not included in the Manifest file. Note the location of the *.sdc file and *.pdc file. These two files are used downstream in the custom flow.The two files are given in."SDC Timing Constraints" section on page 27 and "PDC Physical Design Constraints" section on page 28 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 with Libero. The custom flow supports the use of Synopsys SynplifyPro or Mentor Graphics Precision as third-party synthesis tool. To synthesize your project, follow the steps below:

1. Create a new project in your Synthesis tool, targeting for 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 either EDIF (.edn) or Structural Verilog (.vm).

2. Import Component HDL Files into your Synthesis project:

   a. For each "Component Manifests" Report:

   – For each file under “HDL source files for all Synthesis and Simulation tools", Import the file into your Synthesis Project.

   – Depending on whether you are using Synplify or Precision, also import all files under "HDL source files for Synopsys SynplifyPro Synthesis tool" or "HDL source files for Mentor Precision Synthesis tool", respectively.

3. If your design contains the CCC, OSC, CoreConfigP, or CoreResetP IP cores, import the <top_level>_derived_constraints.sdc in Appendix A "<top_level>_derived_constraints.sdc" into the Synthesis tool. This constraint file constrains the synthesis tool to achieve timing closure with less effort and fewer design iterations.

*Note: If you plan to use the same *.sdc file to constrain Place and Route during the design implementation phase, You must import this <top_level>_derived_constraints.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 don’t include this *.sdc file in the Synthesis step, the netlist generated from Synthesis may fail the Place and Route step because of design object name mismatches.*

4. Import any Netlist Attributes *.ndc, if any, into the Synthesis tool.
5. Run Synthesis.

Note the location of your Synthesis tool output *.edn or *.vm netlist. You need to 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 (i.e., using your own simulation environment and simulator), follow the steps below:

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 Section 3, "Synthesizing Your Design") into your simulation project and compile it

   c. Post-layout simulation:

   First, complete implementing your design (see Chapter 5, "Implementing Your Design"). Ensure that your final Libero project is in post-layout state.

   Double-click **Generate Back Annotated Files** in the Libero Design Flow window. This will generate 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 above lists (in 2.a) are executed first, before the start of simulation.

   c. 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
   #define USE_SERDESIF_0
   #define USE_SERDESIF_1
   #define USE_SERDESIF_2
   #define USE_SERDESIF_3
   ```

   d. ENVM_init.mem: If you are using the eNVM(SmartFusion2 or IGLOO2), or if you are using IGLOO2 and are using 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 via 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.

e. UPROM.mem: If you are using the RTG4 uPROM, you must use the pa4rtupromgen.exe to generate the UPROM.mem file. The pa4rtupromgen executable takes the UPROM.cfg file as inputs via 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.

1. Create a working folder and a sub-folder named "simulation" under the working folder.

   *Note: The pa4mssenvmgen and pa4rtupromgen executables executable expects the presence of the "simulation" subfolder in the working folder and the *.tcl script (step 3) be placed in the "simulation" subfolder.*

2. 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:
   - MDDR_init.reg
   - FDDR_init.reg
   - SERDESIF_0_init.reg
   - SERDESIF_1_init.reg
   - SERDESIF_2_init.reg
   - SERDESIF_3_init.reg

3. 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.

4. Put the following commands in a *.tcl script and put it in the simulation folder created in step 1.

   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
```

   Notes:

   - *For the proper internal name to use for the die and package, refer to 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 would be generated upon execution of the script file with the pa4mssenvmgen executable.*

   *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
```

   Notes:

   - *For the proper internal name to use for the die and package, refer to 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 Manifest" 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 would be generated upon execution of the script file with the pa4mssenvmgen executable

Sample.tcl for RTG4 devices*

```
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
```

Executing the Tcl file:

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
```

Notes:

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

– *For RTG4, there is provision only 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 would be generated upon execution of the script file with the pa4rtupromgen executable.*

5. 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 3.

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
```

6. For SmartFusion2 and IGLOO2, after successful execution of the pa4mssenvmgen command, 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 command, check that the UPROM.mem file is generated in the location specified in the set_sim_mem command in the *.tcl script.

7. 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).

Note: To simulate the functionality of SoC Components, download the precompiled SmartFusion2/IGLOO2 or RTG4 simulation libraries and import them into your simulation environment as described here.

5 – Implementing Your Design

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

1. Create a new Libero project for the physical implementation and layout of the design. Make sure to target the same device as in the reference project you created in Chapter 2, "Component Configuration".
2. Select the same design flow (Enhanced Constraint or Classic Constraint Flow) as the reference project you first create for design component configuration and generation. If the reference project you first create is for the Classic Constraint Flow, create a Libero Classic Constrain Flow project. for design implementation. If the reference project you first create is for the Enhanced Constraint Flow, create an Enhanced Constraint Flow project for design implementation.
3. After project creation, remove Synthesis from the tool chain in the Design Flow Window (**Project > Project Settings > Design Flow > Uncheck Enable Synthesis**)
4. Import your post-synthesis *.edn or *.vm file into this project, (**File > Import > Others or File > HDL Source Files**).

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 bold in Figure 5-1). >

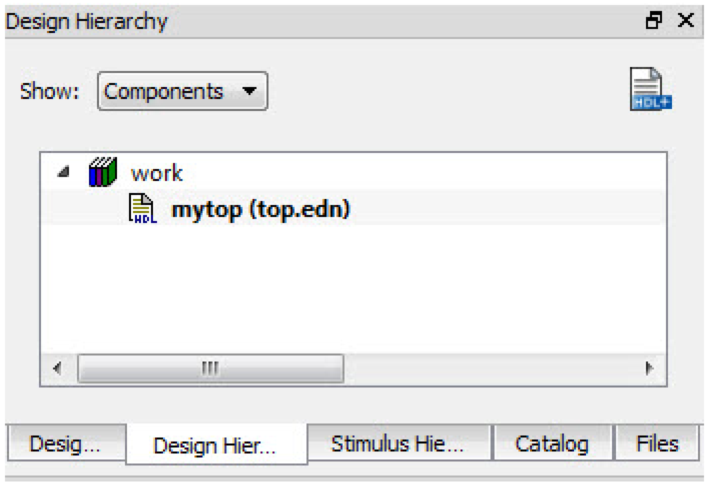


Figure 5-1 • **Root Module in Design Hierarchy**

5. Import the constraints into the Libero project.

- For Classic Constraint Flow, use the **File > Import** menu to import constraints file
  - Import I/O PDC files (**File > Import > I/O Constraints (PDC) File)**
  - Import floorplanning PDC files **(File > Import > Floorplanning Constraints (PDC) Files)**
  - Import SDC timing constraint file **(File > Import > Timing Constraints (SDC) Files**).
- For Enhanced Constraint Flow, use the Constraint Manager to import *.pdc and *.sdc constraints.
  - Import I/O *.pdc constraint files (**Constraints Manager > I/O Attributes > Import**).

– Import Floorplanning *.pdc constraint files (**Constraints Manager > Floor Planner > Import).** If your design contains CoreConfigP (SmartFusion2 and IGLOO2 only), make sure to import the "<top_level>_derived_constraints.pdc (Generated in Enhanced Constraint Flow only)" file.

– Import *.sdc timing constraint files (**Constraints Manager > Timing > Import).** If your design contains CCC, OSC, CoreConfigP and CoreResetP, SERDES, and SmartFusion2 MSS/ IGLOO2 HPMS IP cores, makes sure to import the "<top_level>_derived_constraints.sdc (Available in Enhanced Constraint Flow only)" file.

– Import *.ndc constraint files (**Constraints Manager > Netlist Attributes > Import**)

6. Associate Constraints Files to design tools

– For Classic Constraint Flow, right-click the imported constraint file in the Design Flow window and select **Use for Compile**. This associates the design constraint file to the Compile step.

– For Enhanced Constraint Flow, open Constraint Manager (**Manage Constraints > Open Manage Constraints View**). Check the Place and Route and Timing Verifications checkbox 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 <top_level>_derived_constraints.SDC timing constraint file constrains IP Cores such as CCC, OSC, CoreResetP and CoreConfigP 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: *The floorplanning *.pdc file constrains the CoreConfigP in an optimal location for placement. and improves timing performance of the design.*

– For Classic Constraint Flow, click **Compile** and then **Place and Route** to complete the layout step.

– For Enhanced Constraint flow, 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 will not be running 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. If you need to change eNVM content, open the "Update eNVM Memory Content" dialog box (Figure 5-2). Changes you make in this dialog box will be saved to the eNVM *.cfg file you imported in Step 7.

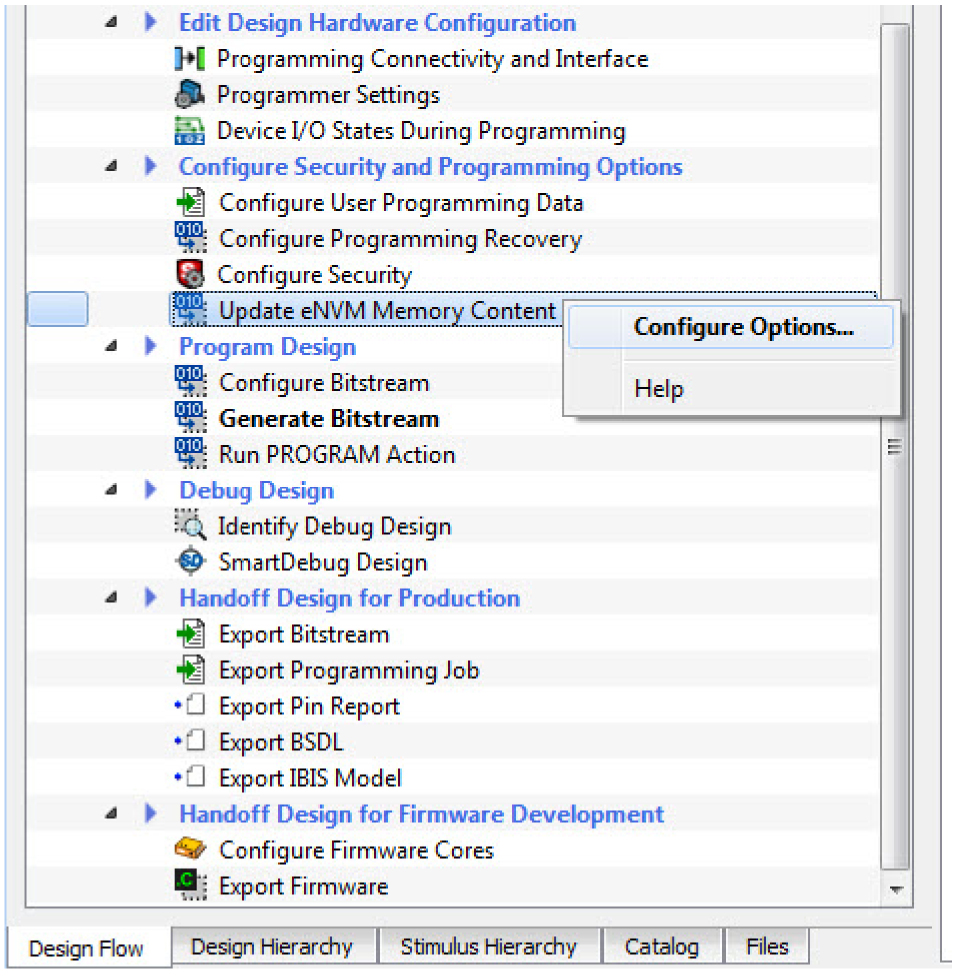


***Figure 5-2* • Update eNVM Memory Content**

9. For RTG4, if you need to change uPROM content, open the "Update uPROM Memory Content" dialog box (Figure 5-3). Changes you make in this dialog box will be saved to the uPROM.cfg file you imported in Step 7.

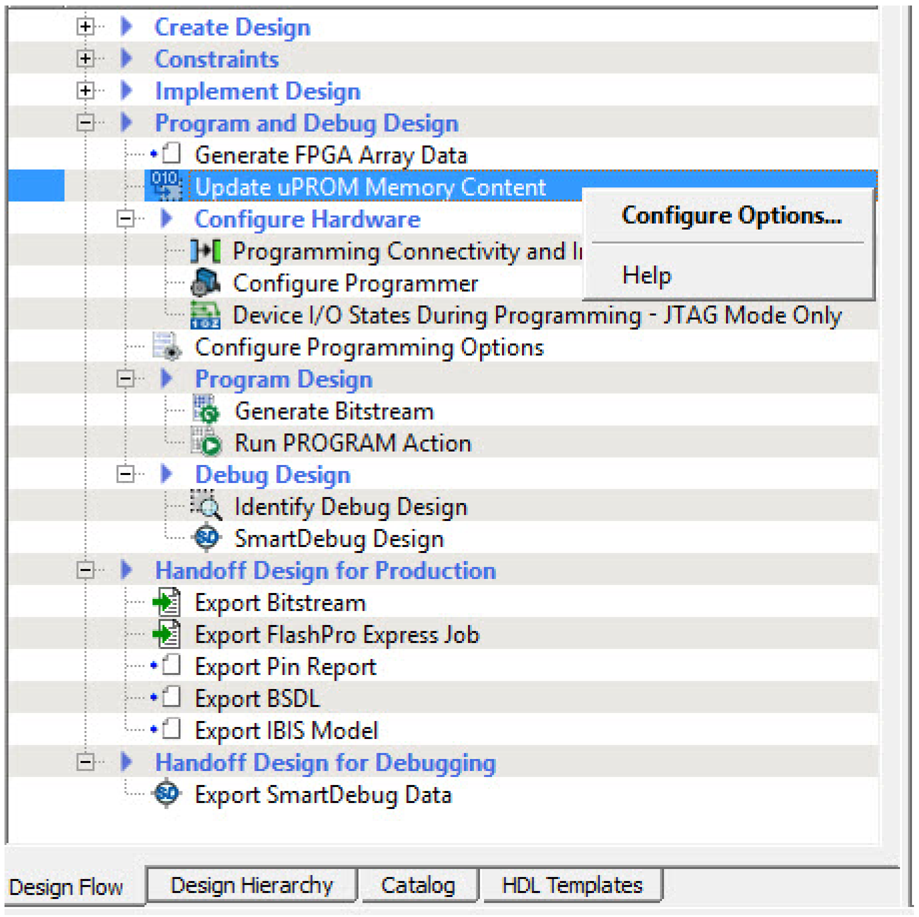


Figure 5-3 • **Update uPROM Memory Content**

10. 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. There are three types of files that make up a firmware project:

- Source files (i.e., your firmware application)
- Drivers: These are drivers provided to facilitate your use of SmartFusion2 SoC Components as well as Microsemi-provided 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 are using 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 shown below

Build your firmware project as follows:

(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—Microsemi provides drivers compatible with Microsemi's SoftConsole, Keil MDK, or IAR Embedded Workbench.
2. Use the Firmware Catalog to download driver files for SoC Components or Microsemi Soft IP you are using 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 "<my_project>/drivers_config/sys_config" directory in your firmware project.
6. For each "Component Manifests" Report (generated in "Component Configuration"):

   a.Import each file in the "Firmware files for all Software IDE tools" section (Figure 2-2 on page 15) into your firmware project's drivers_config/sys_config directory.
7. Navigate to your Libero installation directory (i.e., where Libero is installed), and then navigate to the following directory:

   ```
   <Libero install dir>\data\aPA4M\sysconfig
   ```
   a.There are two files here—sysconfig.c and sysconfig.h.

   b.Import sysconfig.c (as is, do not modify the file) into your firmware project's drivers_config/sys_config directory.

   c.Edit the local copy of sysconfig.h:

   – If you are using 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 are using FDDR and SERDES blocks 0 to 3 in your design, change their respective lines as above. Figure 6-1 highlights the areas that must be changed.

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

8. Proceed with compilation of your firmware project.

```
/*==============================================================================
 * MDDR configuration
 */
#define MSS_SYS_MDDR_CONFIG_BY_CORTEX       0

/*==============================================================================
 * FDDR configuration
 */
#define MSS_SYS_FDDR_CONFIG_BY_CORTEX       0

/*==============================================================================
 * SERDES Interface configuration
 */
#define MSS_SYS_SERDES_0_CONFIG_BY_CORTEX   0
#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
```

***Figure 6-1* • Editing sysconfig.h—only edit the values highlighted in red to bring them in line with your design's usage**

A – Libero-generated hardware configuration files

This appendix describes the hardware configuration files that are generated by Libero. These files are intended to be imported into a firmware project (Chapter 6, "This chapter applies to SmartFusion2 only."). Depending on the components present in a design, not all of these files will be present.

sys_config.h

This header file contains information about the SmartFusion2 MSS hardware configuration. It is generated by the Libero hardware design flow. The content of this file is hardware design specific. This file should not be included in application code.

sys_config.c

This C source file contains information about the SmartFusion2 MSS hardware configuration. It is generated by the Libero hardware design flow. 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. It is generated by the Libero hardware design flow. The content of this file is hardware design specific. This file should not be included in application code.

sys_config_mddr_define.h

This header file contains information about the SmartFusion2 MSS DDR hardware configuration. It is generated by the Libero hardware design flow if DDR is included in the Libero design. The content of this file is hardware design specific. This file should not be included in application code.

sys_config_SERDESIF_<0-3>.c

These C source files contain information about the SmartFusion2 SERDES interface hardware configuration. They are generated by the Libero hardware design flow 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. They are generated by the Libero hardware design flow 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 should not be included in application code.

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.

SDC Timing Constraints

For the Libero SoC Enhanced Constraint Flow, this top-level SDC constraint file is available from the Constraint Manager (**Design Flow > Open Manage Constraint View >Timing > Derive Constraints**). Modify, if necessary, the full hierarchical path and design object names to match the names used in your design.

Note: *This file is NOT generated for the Classic Constraint Flow, even If your design contains CCC, OSC, CoreResetP and CoreConfigP components. Refer to this file to set the SDC constraints for these components. Modify the full hierarchical path, if necessary, to match your design hierarchy. 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.*

<top_level>_derived_constraints.sdc (Available in Enhanced Constraint Flow only)

```
# Microsemi Corp.

# Date: 2016-Aug-08 11:49:33

#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}]
```

PDC Physical Design Constraints

For the Libero SoC Enhanced Constraint Flow, this top-level PDC constraint file is available from the Constraint Manager (**Design Flow > Open Manage Constraint View >Timing > Derive Constraints**). Modify, if necessary, the full hierarchical path and design object names to match the names used in your design.

*Note: This file is NOT generated for the Classic Constraint Flow, even If your design contains the CoreConfigP component. Refer to this file to set the PDC constraints for the CoreConfigP component. Modify the full hierarchical path, if necessary, to match your design hierarchy. 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.*

<top_level>_derived_constraints.pdc (Generated in Enhanced Constraint Flow only)

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. For the Enhanced Constraint Flow project, 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.

```
# Microsemi Corp.
# Date: 2016-Aug-08 11:49:33
# 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}
```

C – Product Support

Microsemi SoC Products Group backs its products with various support services, including Customer Service, Customer Technical Support Center, a website, electronic mail, and worldwide sales offices. This appendix contains information about contacting Microsemi SoC Products Group and using these support services.

Customer Service

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**

Customer Technical Support Center

Microsemi SoC Products Group staffs its Customer Technical Support Center with highly skilled engineers who can help answer your hardware, software, and design questions about Microsemi SoC Products. The Customer Technical Support Center spends a great deal of time creating application notes, answers to common design cycle questions, documentation of known issues, and various FAQs. So, before you contact us, please visit our online resources. It is very likely we have already answered your questions.

Technical Support

For Microsemi SoC Products Support, visit http://www.microsemi.com/products/fpga-soc/design-support/fpga-soc-support.

Website

You can browse a variety of technical and non-technical information on the Microsemi SoC Products Group home page, at www.microsemi.com/soc.

Contacting the Customer Technical Support Center

Highly skilled engineers staff the Technical Support Center. The Technical Support Center can be contacted by email or through the Microsemi SoC Products Group website.

Email

You can communicate your technical questions to our email address and receive answers back by email, fax, or phone. Also, if you have design problems, you can email your design files to receive assistance. We constantly monitor the email account throughout the day. When sending your request to us, please be sure to include your full name, company name, and your contact information for efficient processing of your request.

The technical support email address is soc_tech@microsemi.com.

My Cases

Microsemi SoC Products Group customers may submit and track technical cases online by going to My Cases.

Outside the U.S.

Customers needing assistance outside the US time zones can either contact technical support via email (soc_tech@microsemi.com) or contact a local sales office.

Visit About Us for sales office listings and corporate contacts.

Sales office listings can be found at www.microsemi.com/soc/company/contact/default.aspx.

ITAR Technical Support

For technical support on RH and RT FPGAs that are regulated by International Traffic in Arms Regulations (ITAR), contact us via soc_tech_itar@microsemi.com. Alternatively, within My Cases, select **Yes** in the ITAR drop-down list. For a complete list of ITAR-regulated Microsemi FPGAs, visit the ITAR web page.

5-02-00672-3/02.18