SmartFusion2 and IGLOO2

Microsemi Design Separation Methodology

Table of Contents

Introduction

This document describes the design separation methodology required to implement security and safety critical applications. For a system to be secure and reliable, all critical subsystems in the design should be independent of each other.

Traditionally such a system with security and safety critical requirements is built with each critical subsystem implemented using multiple ICs. With each critical sub-system as an independent IC, fault and reliability analysis is simplified. In a traditional FPGA design, netlists generated for place and route are often flattened for efficient placement. Design functions from various parts of the design hierarchy may share physical resources. To meet critical security and safety application requirements, critical sub-systems within an FPGA design may need to be isolated, both to simplify failure analysis and as a measure to prevent propagation of faults from one sub-system adversely affecting another.

The Microsemi Design Separation methodology provides a method for creating independent critical subsystems on a single FPGA. Functional blocks that must be independent can be physically isolated from other functional elements in the FPGA via place and route constraints within the Libero SoC software. Figure 1 shows a top level view implementing a security and safety critical application in a Microsemi FPGA.

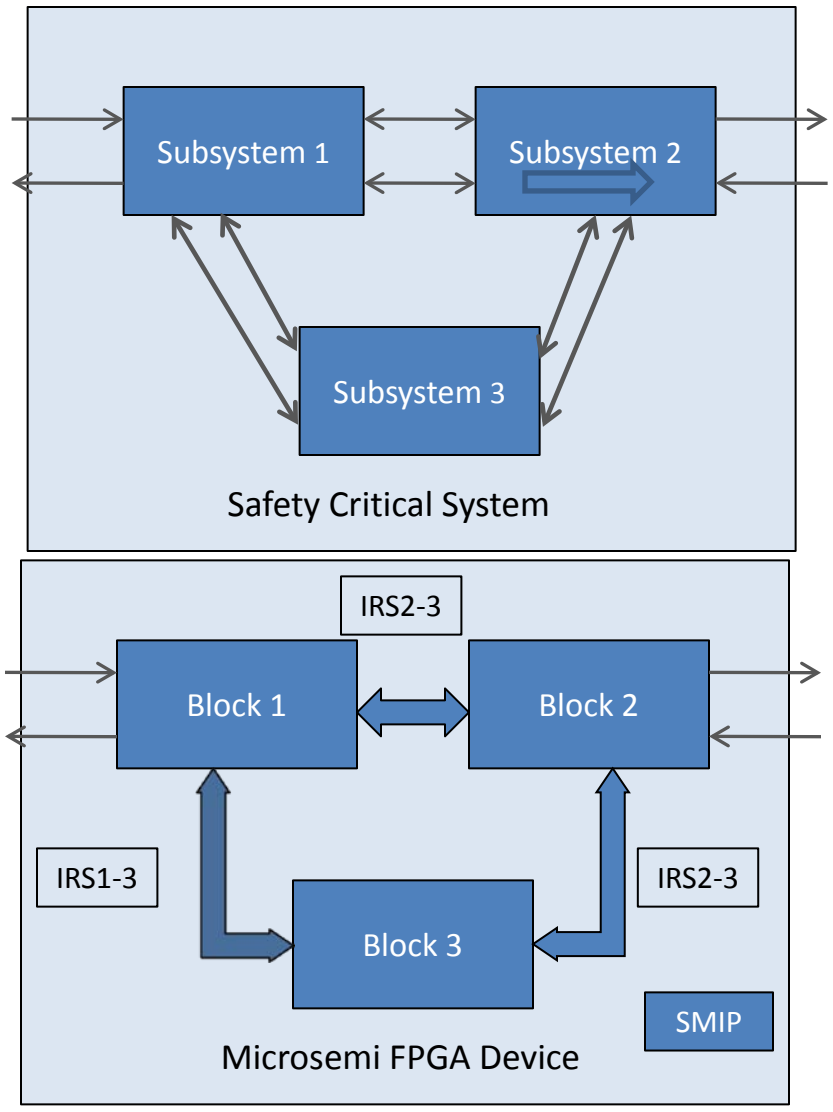


Figure 1. Implementing Security and Safety Critical Applications in Microsemi FPGAs

Overview

The Microsemi design separation methodology is comprised of three components:

1. Ability to create independent subsystems

2. Ability to validate that isolation

3. Ability to monitor for faults

The design separation methodology leverages an existing design methodology, Block Flow, to achieve isolated functions. The Block flow is a bottom-up design methodology that enables an incremental design approach. In a Block flow Compile, component modules within a design are compiled and optimized in independent stages from the rest of the project. After compile, the Component modules

are published as a netlist (“Block” in Libero terminology), and then exported up to a top level project for integration with other modules in the larger system design. A key component of the Block flow is resource reservation, since the Block needs to be compiled with all physical resources it needs a priori. Routing reservation and logic reservation are both constraint options that are available from the Block flow methodology. Creating isolated sub-systems for the purposes of security and safety critical applications, then, is an application of the Block flow. All critical sub-systems are assigned to an exclusive region (a region with strict resource reservation) and floorplanned with a guard-band of unused clusters away from all other logic.[1] Signal connections to another module (known as “Inter-Region Signals” within this flow) are assigned to another resource-reserved block region (“Inter-Region Block”) that overlaps the source and sink regions. The Inter-Region Signals are thus members of the source region, the sink region, as well as the Inter-Region Block. The Inter-Region Block acts as a constrained routing channel.

A separate tool, known as the Microsemi Separation Verification tool, is used to check that a design meets separation requirements, as defined by the system requirements of the design. Libero automatically generates a parameter file (MSVT.param) that details Blocks that are present in the design, and the number of signals entering and leaving a Block. MSVT checks the final design place and route against the MSVT.param file and reports any violations based the separation requirement defined by the user.

Anti-Tamper (AT) must be considered in addition to the isolation of critical design subsystems. SmartFusion2 and IGLOO2 devices come standard with robust design security, a critical portion of which is Anti-Tamper and fault detection. SmartFusion2 and IGLOO2 devices include mechanisms to allow an FPGA design to monitor the integrity of the device during operation. As Fault detection is a critical part of security and safety critical systems, CoreSMIP is a valuable element in this design flow. CoreSMIP is an IP core that ties together the relevant AT hooks within SmartFusion2 and IGLOO2 devices.

The flow chart in Figure 2 shows the various steps of the design separation methodology. Below is a brief description of each step mentioned in the flow-chart.

[1] Width of guard-band dependent on individual project requirements.

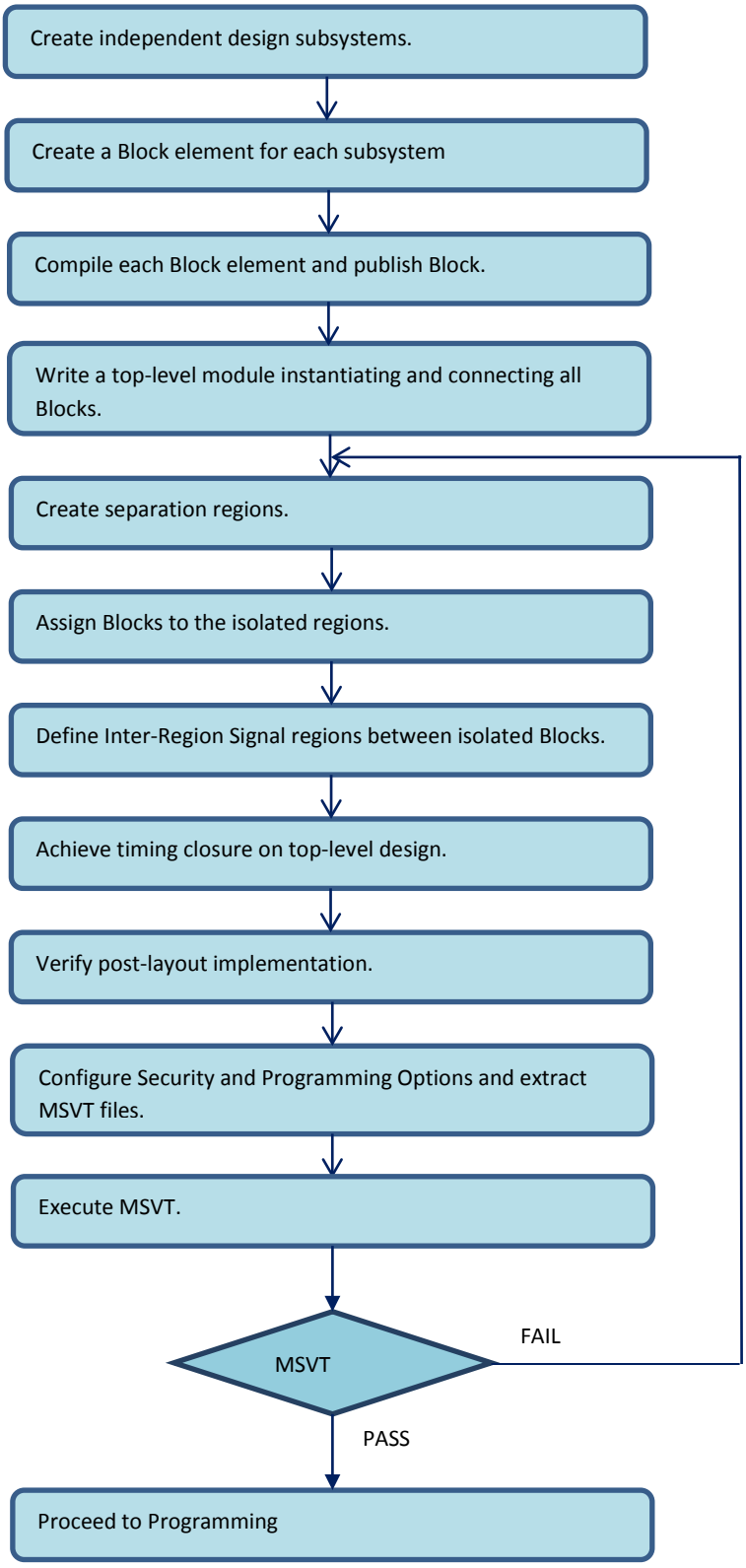


Figure 2. Microsemi Design Separation Methodology

1. Create independent design subsystems
   Create an RTL description for each of the subsystems. Each subsystem should be independent from the others with its own logic resources. The RTL module defining each subsystem should be independent of other subsystems.

2. Create a Block element for each subsystem
   Each independent subsystem should be defined as a Block. The Block design flow creates logical partitions for the subsystems in question as a handle for place and route constraints in later stages of the design separation methodology. All corresponding I/O ports of a subsystem should be assigned to the respective block. In the design separation methodology, all logic must be a member of an isolated block region.

3. Perform analysis on each Block element and publish the Block
   For each Block, run Synthesis and Compile. Assess the size and shape of suitable regions based on the types of I/Os, length of cascaded MACC, RAM count and fabric resource usage. Publish the block without place and route.

4. Write a top-level module instantiating all Blocks
   Write a top-level module that only instantiates and connects all the Blocks. Import each published Block. Enable the Design Separation Methodology option in the Compile tool.

5. Create separation regions
   For each Block in the design, create separation regions by specifying region constraints using ChipPlanner or defining regions in a PDC file.

6. Assign Blocks to the isolated regions

7. Define Inter-Region Signal regions between isolated Blocks
   If two Blocks interact with each other, create an overlapping Inter-region Signal (IRS) region connecting the Blocks. These IRS regions should also be physically isolated from other IRS regions. Assign IRS nets to each respective IRS region.

8. Achieve timing closure on top-level design
   Enter the necessary Timing constraints. Perform design iterations to achieve timing closure.

9. Verify post-layout implementation
   Verify all aspects of timing and power. Generate back-annotated files and perform post-layout simulations.

10. Configure Security and Programming Options and extract MSVT files
    Generate the programming file. This step will also export information for Microsemi Separation Verification Tool (MSVT).

11. Execute MSVT

Run MSVT tool from the command line. If MSVT fails, re-examine the floorplan and iterate the entire design flow with corrected region constraints.

Design Methodology

Creating Blocks

A security and safety critical application may consist of one or more independent subsystems. Each subsystem should be defined using HDL in such a way as to have independence from the rest of the system. Each subsystem should have its own resources, including I/O buffers for external FPGA signals. A Block element is created for each such subsystem, which is then instantiated in a top-level design.

To create a Block element for each subsystem, set the module as a Root module. To mark a design module as a root module, right-click the target module in the Design Hiearchy tab, and choose **Set as root**. Enable Block Creation and Use Standalone Initialization for MDDR/FDDR/SERDES Peripherals from Project Settings as shown in Figure 3.

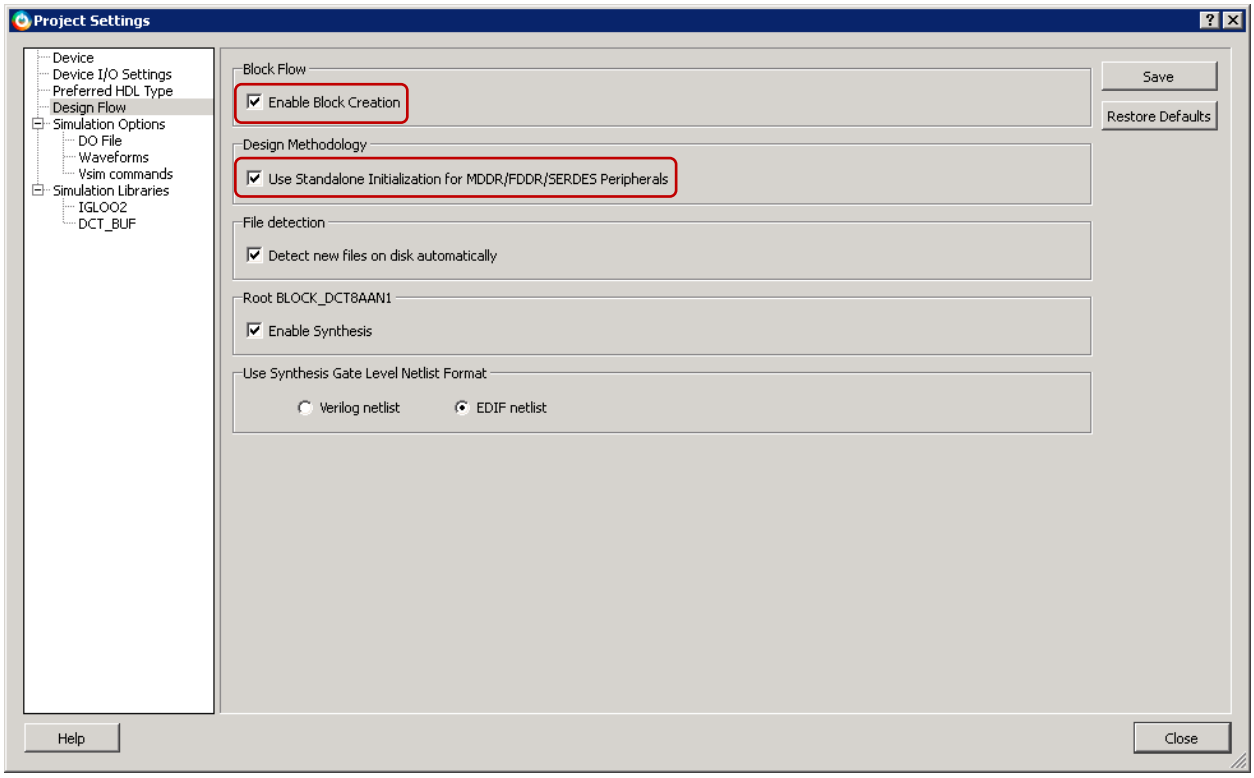


Figure 3. Project Settings for Block Creation

Block Flow is a bottom-up design methodology. The Block attribute within the Block flow identifies components within an HDL hierarchy to be re-usable within a team based design flow as a modular resource. For more details on Block Flow refer to Microsemi Block Flow User Guide for SmartFusion2, IGLOO2. For more details on Peripheral Initialization refer to SmartFusion2 DDR Controller and Serial High Speed Controller Standalone Initialization Methodology or IGLOO2 DDR Controller and Serial High Speed Controller Standalone Initialization Methodology.

If a Block uses physical I/O pins, then those physical resources should be defined as part of that block. This requires explicit definition of I/O to be assigned to the Block using I/O pads. This can be achieved through direct instantiation of an I/O buffer within the module in question, or through the Libero SmartDesign tool. Please refer to "Assigning I/Os to Block" section below.

For each module with its I/Os defined, run Synthesis and Compile. Analyze the Compile report to assess the size and shape of suitable regions based on types of I/O, length of cascaded MATH blocks (MACC), number of RAM blocks, PLLs and CCC count and fabric resource usage.

Optionally, enter timing constraints and run place and route followed by timing analysis to achieve timing closure for each individual Block. This exercise will give you an indication of the difficulty of timing closure at the top level of the design.

Once these steps are completed, the Block is ready to be published.

Since we will be assigning these Blocks to isolated separation regions (explained in subsequent sections), you need to publish the Block without placement and routing information. Configure Publish Block options to exclude Placement and Routing information, as shown in Figure 4.

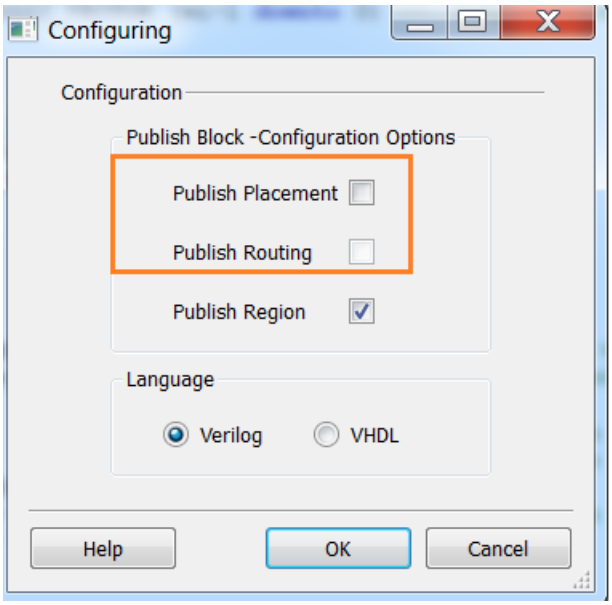


Figure 4. Configuring Publish Block Options

Libero exports the <block_name>.cxz file to <project_path>/designer/<block_name>/export directory when a Block is published. The <block_name>.cxz file is the published Block. This is the file you import into the top-level design when you want to instantiate the Block.

Assigning I/Os to Block

Signals that route to physical I/O pins within each module should belong to the corresponding Block. For the purpose of design separation, physical I/O resources must be associated with an isolation region. Enabling block flow disables automatic I/O insertion by the Synthesis tool. Hence, the design separation methodology requires explicit instantiation of I/O buffers that are required per Block. These I/O buffers can be inserted from the I/O configurator in the SgCore Catalog or the Macro Library Catalog.

Please refer to the SmartDesign sections in the Libero Online Help. These macros ensure that all design ports assigned to them infer an I/O port assigned to the given Block. Do not insert any I/O buffer on ports that will be used to interconnect with other blocks i.e. Inter-Region Signal (IRS) nets.

To insert I/Os in a Block, Microsemi recommends that you use Libero's SmartDesign tool. Follow the steps below to create a SmartDesign component of the subsystem.

i. Create a SmartDesign (SD) and instantiate the module in SD.
ii. Insert appropriate macros from the Macro library Catalog for each type of port. The relevant macros are: INBUF, INBUF_DIFF, OUTBUF, OUTBUF_DIFF, TRIBUFF, TRIBUFF_DIFF, BIBUF, and BIBUF_DIFF.
iii. In case ports belong to a bus, use the I/O configurator with required width and type of buffer.
iv. Once required macros are instantiated in SD, connect the ports of the design with the respective macros.
v. Rename the I/O pads with names defined in the module. Generate SD.
vi. Set the generated SD as the root module and create a Block using this module as described above.

Figure 5 shows a SmartDesign component in which a subsystem, DCT_BUF12, has been instantiated, and Top-Level ports are assigned to its I/Os using INBUF, OUTBUF macros and the I/O Configurator.

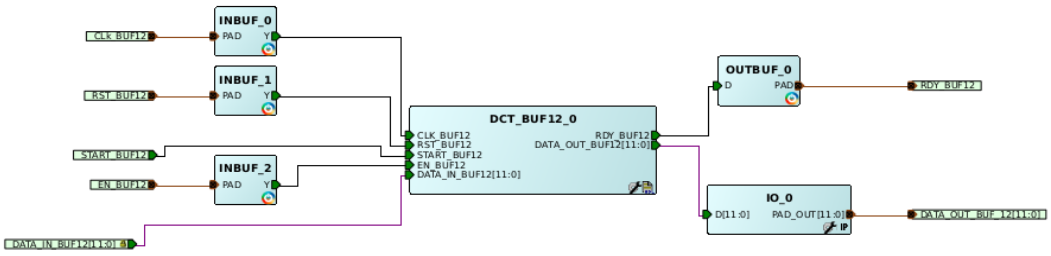


Figure 5. SmartDesign Instantiating Subsystem along with I/Os

As an alternative to SmartDesign, you can also instantiate the macros and connect them to the top-Level ports of the design.

Since lower-level modules are compiled independently from the top level design, you must check to ensure unique I/O pin names across the design are not used within the lower level project. Microsemi recommends checking the I/O pin names across the project to ensure for uniqueness when building the lower level project.

Optional CoreSMIP Block

The Security Monitor IP (CoreSMIP) is a core provided by Microsemi for Tamper detection, thus enhancing the security of the system. More details on CoreSMIP can be found in the CoreSMIP User Guide. CoreSMIP is present in the catalog under SC/Tamper section.

In the event your design includes CoreSMIP, and because the design separation methodology requires each subsystem to be defined as a Block, you must create a CoreSMIP Block following the same steps as the other Blocks.

Creating a Top-level Design

Once all Blocks are published, create a new Libero project for the top-level design. Write a top-level module in whatever language you use for the Libero project. This top-level module should contain instantiations of all Blocks along with interconnects between them, replicating a complete system.

Set this top-level module as the root module in Libero and disable Block Creation. Using the File → Import → Blocks menu, import all published subsystem Blocks (<block_name>.cxz files) into this design. Since all published Blocks have already completed Synthesis and have I/Os assigned to them, you do not need to run Synthesis.

Figure 6 shows Project Settings required for top level design with Block Creation and Synthesis disabled.

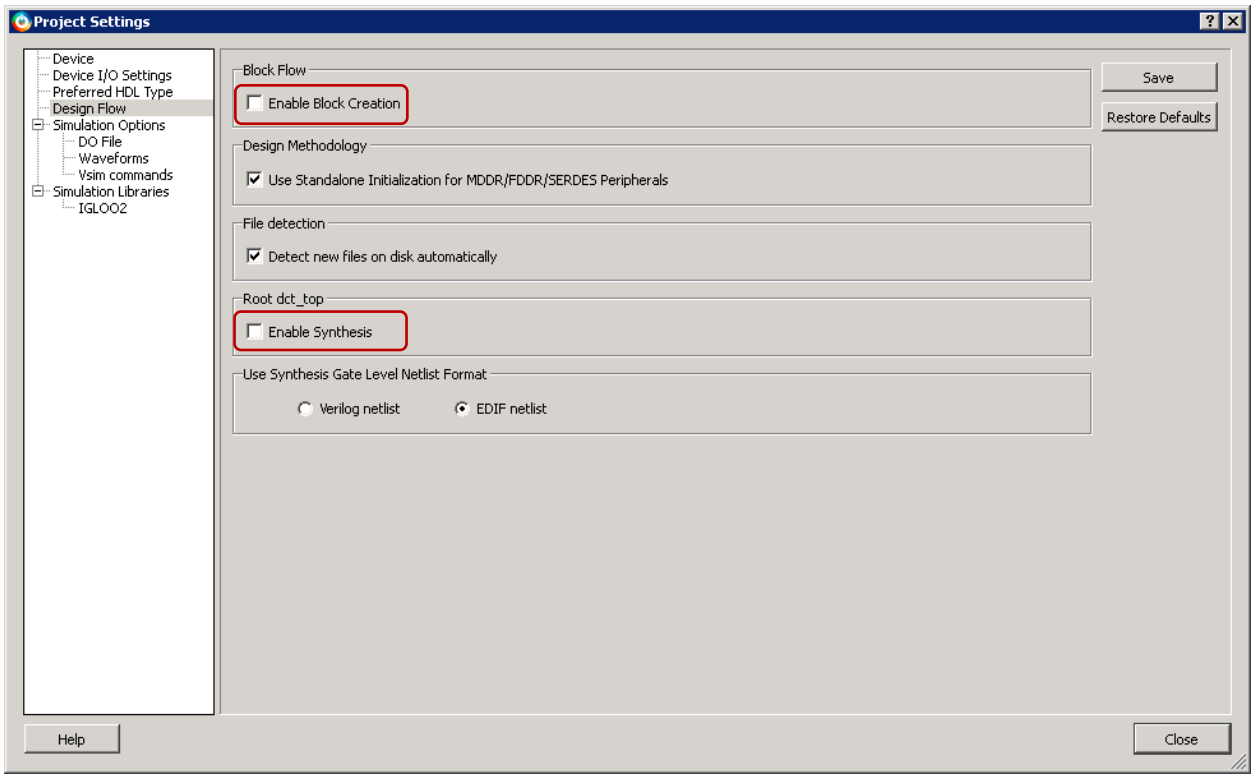


Figure 6. Project Settings for Top-level Design

In addition, configure Compile and click the checkbox to Enable Design Separation Methodology. This option facilitates the generation of files needed by MSVT while creating the Programming file later in the flow. Figure 8 shows the values for Compile. Run Compile from Libero to compile the design.

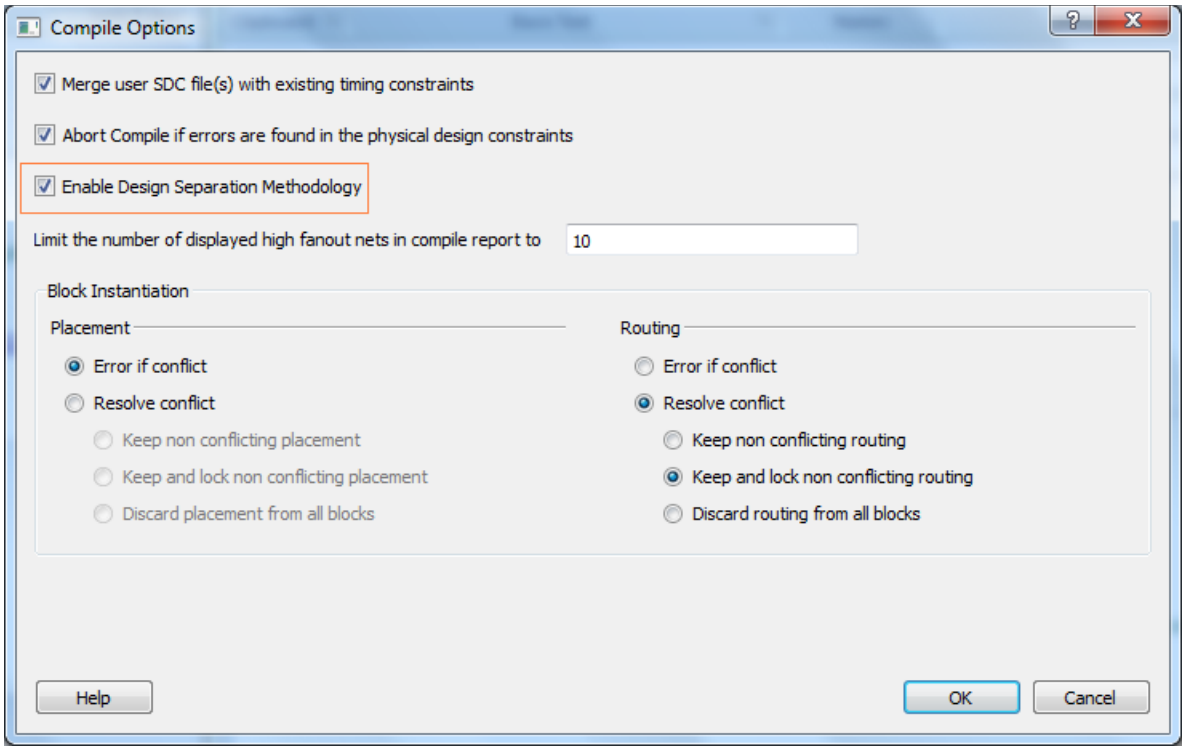


Figure 7. Enabling Design Separation Methodology before Compile

Floor-planning with Design Separation Regions

Once the top-level design with subsystem Blocks has been created, floorplan the design by defining separation regions and Inter-Region Signal (IRS) Regions. In a design that follows the Microsemi Design Separation Methodology, all logic should be contained in a logic region with dedicated place and route resources.

A logic region is a user-defined area on the device within which logic can be assigned. A Separation region is a logic region with the following features:

- It is a resource reserved (place and route) region.
- May be a non-rectangular region (built from a union of multiple rectangular floorplan regions)
- Regions are separated from each other by reserving a perimeter of unused clusters

You can use ChipPlanner to create regions or create them with PDC commands. ChipPlanner is the **floorplanning** tool you use to create and edit regions on the chip and assign logic to these regions.

Create a Separation region for each Block present in the design. The size of and shape of the region should depend on quantity of fabric resources, I/O types, RAM, MACC and CCC being used in each block.

Each Block region is a place and route constraint for logic elements that are associated with it. Physical separation is achieved by allowing some unused logic clusters as a guard-band around each block region. The unused cluster spacing between regions is dependent upon final system requirements. Floorplan

according to the guard-band that is appropriate for the security and safety requirements of the target design.

SmartFusion2 and IGLOO2 architecture is cluster-based. For SmartFusion2 and IGLOO2 devices, a cluster is made up of 12 Logic Elements. A Logic Element includes a 4 input-LUT, a register, and a carry chain. In the ChipPlanner coordinate system, each Logic Element component has a unique coordinate. As such, each Cluster occupies an area of 12x3. Figure 8 shows a single cluster as shown in the ChipPlanner with dimensions noted. The granularity of ChipPlanner region sizes is one cluster.

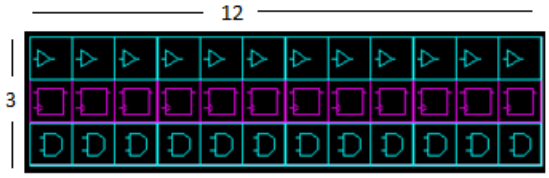


Figure 8. One Cluster of SmartFusion2/IGLOO2 FPGA

Regions may be defined for each Block using either the ChipPlanner or with a Physical Design Constraints (PDC) file. The size of each such region should be able to accommodate all resources used by a given block, including all embedded hard IP blocks such as I/Os, RAM, Math Blocks, and Clock Conditioning Circuit blocks (CCCs).

Note that RAM, Math Blocks and CCCs take up a footprint of three clusters in the SmartFusion2 and IGLOO2 floorplans. The makeup of embedded hard-IP blocks include the hard IP resource itself abutted to a set of Interface Clusters. The Interface Clusters help route signals to and from the hard IP resource to the rest of the fabric array. Figure 9 and Figure 10 detail the makeup of a hard IP block and its corresponding visualization within the ChipPlanner floorplan, respectively.

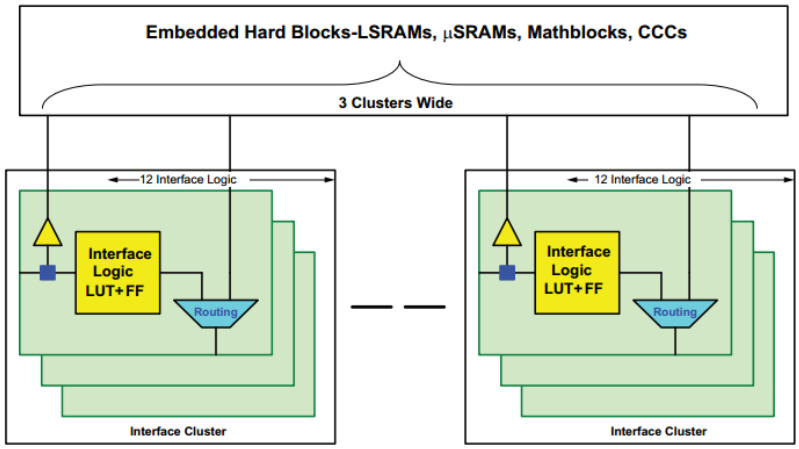


Figure 9 : Interface cluster for a hard IP block.

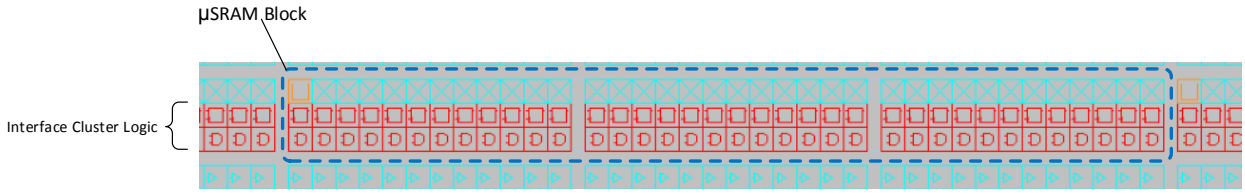
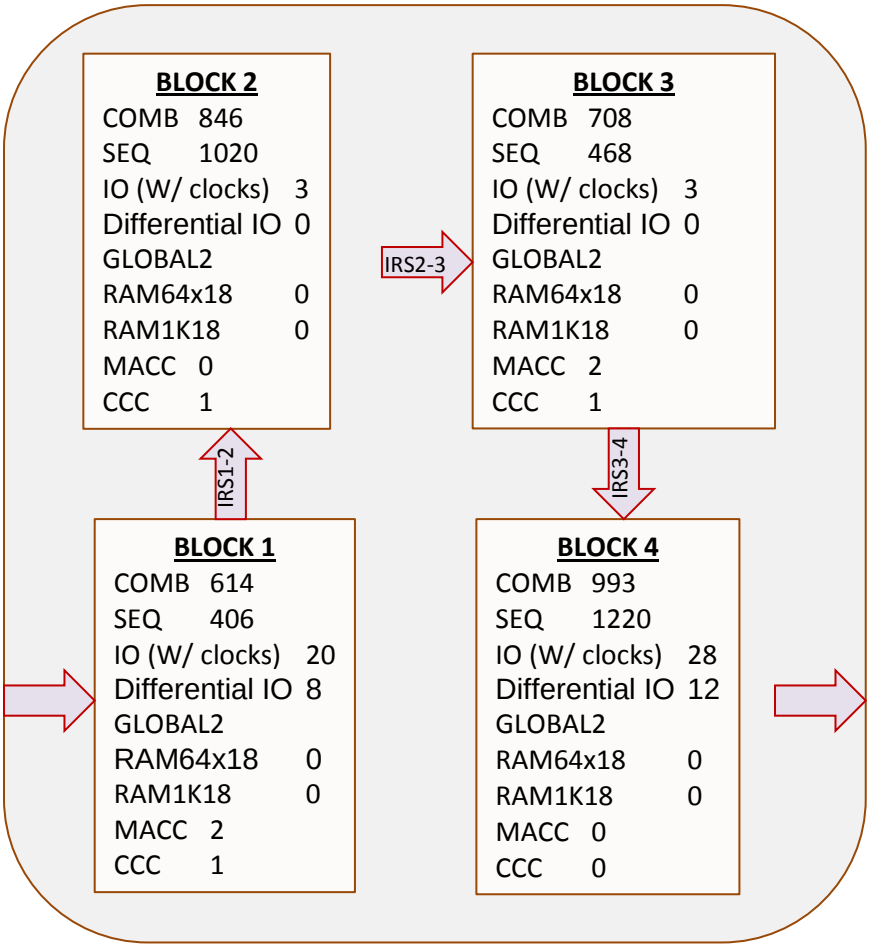


Figure 10 : uSRAM block as shown in Chip Planner

An isolated region constraint must include the entire embedded hardblock resource within its boundaries for the resource to be usable within the target region. Use Non-rectangular regions to help efficiently allocate a floorplan to include these hard IP blocks.

More information about the fabric architecture for SmartFusion2 and IGLOO2 devices can be found in the SmartFusion2 fabric user guide and the IGLOO2 fabric user guides, respectively.

Figure 11 and Figure 12 provide an example floorplan from a sample design using a PDC file and the floorplan as shown in the ChipPlanner, respectively. The granularity of placement units are logic modules in the ChipPlanner coordinate system and the granularity of region sizes is clusters. Hence, regions must be a multiple of 12 in the horizontal direction and a multiple of 3 in the vertical direction.



```
// Define Separation and IRS Regions
define_region -name Block1_Region  -route yes 156   6 263  71
define_region -name Block2_Region  -route yes 324  78 443 143
define_region -name Block3_Region  -route yes 480  90 599 128
define_region -name Block4_Region  -route yes 156  84 275 143
define_region -name IRS_1_2_Region -route yes 156  84 443 143
define_region -name IRS_2_3_Region -route yes 240 102 371 143
define_region -name IRS_3_4_Region -route yes 420  99 515 125

// Assign Blocks to Respective Separation Regions
assign_region Block1_Region Block1
assign_region Block2_Region Block2
assign_region Block3_Region Block3
assign_region Block4_Region Block4

// Assign IRS Signals to respective IRS Regions
assign_net_macros IRS_1_2_Region rdy1 data1<*>  -include_driver yes // Block 1 and Block 2
assign_net_macros IRS_2_3_Region rdy2 data1r<*> -include_driver yes // Block 2 and Block 3
assign_net_macros IRS_3_4_Region rdy3 data2<*>  -include_driver yes // Block 3 and Block 4
```

Figure 11. Example floor-plan PDC file

For more information about floor planning with the Chip planner and PDC syntax refer to the MultiView Navigator online help in Libero SoC.

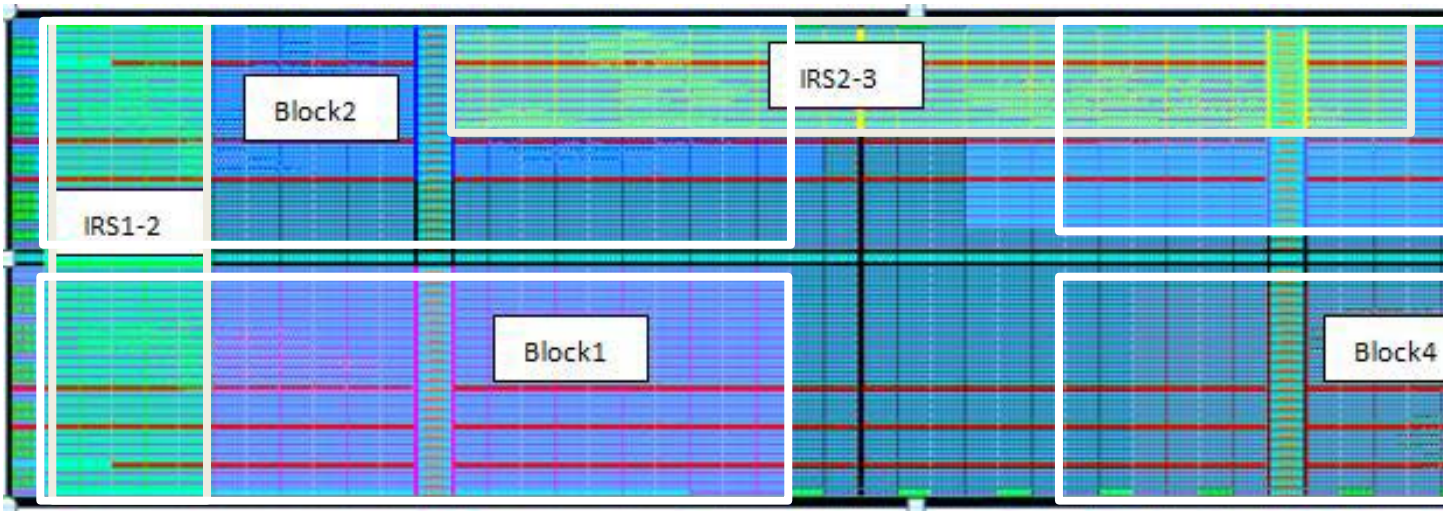


Figure 12. Example Floor-Plan of Top-level Design

Inter-Region Signal (IRS) Regions

Since each Block is defined in an isolated region, we must ensure that a routing channel with valid inter-Block communication interconnect exists that is separated from other unrelated regions. These inter-Block interconnect channels are defined using IRS regions.

An IRS region is another routing region that overlaps with the isolated Block regions. All signals that have a valid connection point between the source and destination Blocks are explicitly assigned to the IRS routing region.

IRS regions have essentially the same requirements as separation regions mentioned above. IRS regions should contain valid communication interconnect nets assigned to them. An IRS region overlaps with the separation regions being connected.

Each IRS region should connect only one set of connected Blocks. Each set of IRS should also be separated by a certain number of clusters from all other Blocks in all directions, both inside and outside the connected Blocks. The amount of separation required will be dependent on your system requirements.

The cells connected to IRS must be isolated from other IRS connected to a different set of Blocks. It is best to insert a buffer or register (with only global clocks and resets) at the source and sinks of IRS.

Considerations for Global clock resources

The global clock network on SmartFusion2 and IGLOO2 devices provide a dedicated low skew, high fanout network to all logic clusters within the fabric array. There are a number of global buffers per device with the following potential inputs :

- Dedicated Global I/Os
- Clock Conditioning Circuits (inclusive of PLLs)
- On-Chip (hardened) oscillators
- FPGA Fabric routing

A detailed description of the clock distribution architecture and associated clocking resources can be found in the SmartFusion2 Clocking Resources User Guide and the IGLOO2 Clocking Resources User's Guide.

The design separation flow only considers physical isolation of logic regions through the analysis of routing elements on the programmable switch fabric in the FPGA. Global networks, as it is a dedicated routing tree, are not analyzed as part of this flow. Hence, for the purposes of design separation, global signals that are common to multiple regions (such as clock and reset) do not have to be separated from any other signal.

However, note that high fanout-signals from the fabric array are often automatically promoted onto the global network. In such cases, the designer may wish for high fanout signals that are meant for a region to use local routing resources only. To understand which signals are promoted onto the global network, inspect the Compile log and Global Net report to confirm which nets get assigned to GB and which nets get implemented on Row Global Buffer (RGB) resources.

Note that promotion and demotion of signals can also be controlled via synthesis attributes. Refer to the SynplifyPro User guide to explore directives to modify the threshold values where global promotion occurs.

As MSVT only audits the programmable switch fabric, any hard macro resources (such as the CCC or an RC oscillator) are not audited. Most inputs to the CCC are from hard blocks, such as from an dedicated I/O pin or the RC oscillator are routed on dedicated metal traces. However, CCC inputs may also be driven from the fabric. If an input or output of a CCC is routed, then design separation constraints will apply. In such a case, the physical CCC resource must also be encapsulated within same region as the source signal driving the CCC. The locations of the CCC may be restrictive for planning the regions – they occur in pairs in each quadrant and some quadrants may not have any CCCs.

RGB resources (RCLKINT/RGCLKINT macros), if used, must be included in a design separation region. Connectivity in the row served by a RGB is dictated by programmable switches, and therefore, is analyzed by the MSVT. RGBs are present on every row and are distributed along a few columns across the fabric array (locations are device dependent). You need to be aware of the location of the RGB columns and the width of such regions is determined by the span of the RGB output.

Initialization of Hard ASIC blocks

SmartFusion2 and IGLOO2 devices contain a number of hardened peripherals, such as SERDES blocks, and hardened memory controllers. These peripherals often rely on initialization routines, where register values are configured to the desired operational parameters. In the standard Libero flow, initialization of these hardened peripherals is controlled via a centralized initialization controller. Fabric routing resources are used to connect with the centralized configuration controller, and in such cases will cause a violation of design separation constraints. If hardened peripherals are used in the design, then the standalone initialization flow must be used in conjunction with the design separation flow.

For more details about standalone initialization of peripheral blocks refer to the Standalone Peripheral Initialization User Guide.

Complete P&R

Once a floorplan of the entire design is complete with separation regions and IRS regions defined, run place and route and verify post-Layout implementation as per the regular Libero design flow.

Verify that timing closure can be achieved for the design. If the design does not meet timing, clone and modify the timing constraints scenario for Timing-driven place and route (TDPR) and explore alternative optimization through High-effort or Power-driven options. You could also change the floorplan and iterate through the design. Standard FPGA design practices like Incremental flow are available. Care should be taken that all criteria required for separation of design remains intact while changing the design floorplan.

Configure Security Settings and Generate Programming file

You can use the Security Policy Manager(SPM) to set design security attributes after place and route is complete, and before you generate programming files. This includes setting user encryption keys and hardware access control policies. Configure the SPM as appropriate for the target system design. For more information about design security and the options available in the Security Policy Manager refer to the SmartFusion2 Security and Reliability Guide and the IGLOO2 Security and Reliability Guide.

Auditing by MSVT

Microsemi Separation Verification Tool (MSVT) is a standalone tool provided with the Libero installation. It is used to verify that the design meets design separation requirements.

The tool takes as input the design database (the ".dtf" directory within the active project directory) and a parameter file that is generated once per project. The parameter file describes the isolation regions in the design as well as the inter-region signals between isolation regions. This file is created when the Enable Design Separation Methodology checkbox in the Compile options dialog is enabled. This file is exported to the following location:

<project_path>/designer/<Top_Level_Module>/msvt.param

The tool can work on any placed and routed design which has a Block that requires a separation from all elements external to the block. The tool works iteratively on every Block to be verified. Internal signals and IRS are verified separately. The tool checks whether the separation criteria is satisfied for each Block and corresponding sets of IRS signals.

MSVT prints an exhaustive report on each Block and corresponding IRS regions being verified. If any of the Block or IRS signal does not satisfy minimum separation criteria, the tool reports details of affected instances. More details on the MSVT output report is defined in the MSVT User Guide.

An MSVT failure indicates that the design has not met the design separation criteria and one or more sub-blocks (or signals) are not independent of rest of the system. In this case, identify instances that cause violations in the MSVT output, and accordingly modify the design floor-plan. Recompile the design to generate a new place and routed netlist. Verify the modified design using MSVT tool.

If the design satisfies separation criteria, then MVST output reports “MSVT Check succeeded”, indicating that the required design separation has been achieved on the design.

Executing MSVT

MSVT is a stand-alone tool present in <Libero_path>/bin64/msvt_check. The tool is executed from the command line. A required argument is –p along with the path to the msvt.param file that is generated from Libero as described in the previous section.

The Msvt.param file contains the parameters required by MSVT to verify design separation. Figure 13 shows an example parameter file.

```
//**********************************************************************
//
//  This is input parameters file for MSVT Check program
//
//**********************************************************************
DEVICE = M2S050T                        // Device Selected
DESIGN = DCT_AAN.msvt                   // Location of Files required for MSVT
VERIFY_BLOCKS = U_B1 U_B2 U_ST2 U_ST1   // List of blocks to be verified using MSVT
REQUIRED_SEPARATION = 1                 // specifies the desired number of switch separation
MAX_VIOLATIONS_PER_REPORT_SECTION = 1   // maximum number of violations

// IRS Regions between each pair of blocks. This contains list of inter-block communicating signals.
IRS U_B1 U_ST2 = rdy2 data1r[0] data1r[1] data1r[2] data1r[3] data1r[4] data1r[5]
        data1r[6] data1r[7] data1r[8] data1r[9]

IRS U_ST1 U_B1 = rdy1 data1[0] data1[1] data1[2] data1[3] data1[4] data1[5] data1[6]
        data1[7] data1[8] data1[9]

IRS U_ST2 U_B2 = rdy3 data2[0] data2[1] data2[2] data2[3] data2[4] data2[5] data2[6]
        data2[7] data2[8] data2[9] data2[10] data2[11]

REGIONS_VERBOSITY = 0
```

Figure 13. Example MSVT parameter file

Inspect the generated MSVT parameter file. Edit the required separation parameter per guideline requirements and adjust other parameters to refine the verification criteria. You can specify the blocks you want to verify, the names of each IRS signal, and limit the maximum number of violations to be reported. More description on each parameter can be found in the MSVT User Guide.

The command to execute to verify the design using MSVT is:

<Libero_path>/bin64/msvt_check –p <project_path>/designer/<Top_Level_Module>/msvt.param [–o msvt_check.log]

This command will print an exhaustive report into the filename given with the –o argument or to *stdout* if –o is omitted.

On successful completion of this command, you will see a final message of either “MSVT Check failed” indicating that the design has not met one or more of separation criteria or “MSVT Check succeeded”, indicating the design has met all separation criteria.

EXAMPLE

This section implements a complete system using Microsemi Design Separation Methodology.

The example design is an open source soft core performing 8x8 point Discrete Cosine Transform (DCT). The design is comprised of four subsystems defined in VHDL.

i. DCT8AAN1.vhd
ii. DCT_BUF10.vhd
iii. DCT8AAN2.vhd
iv. DCT_BUF12.vhd

Figure 14 shows a top level view of these subsystems with interconnects between them.

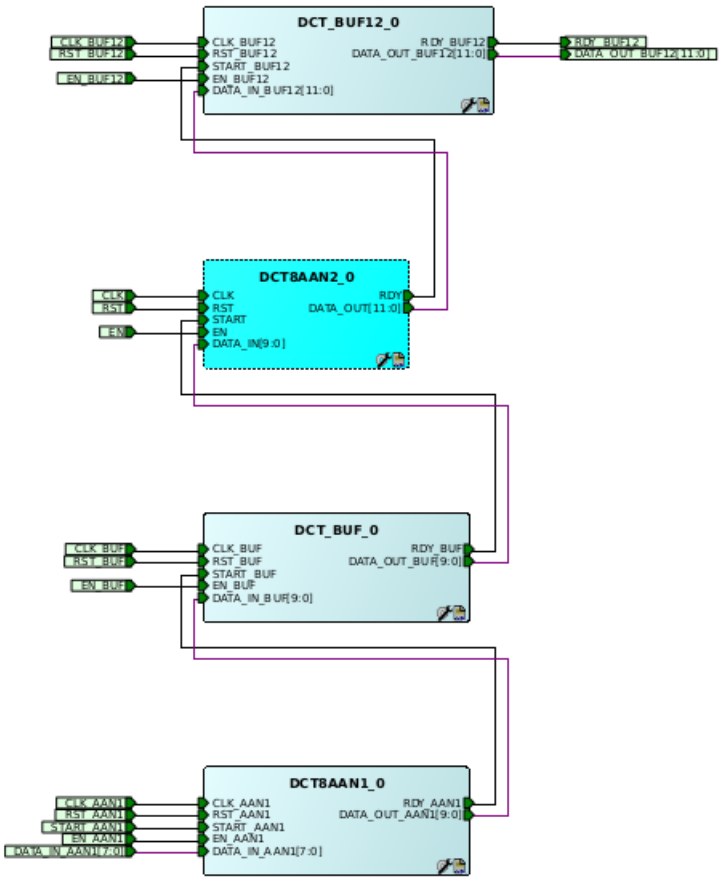


Figure 14. Top level view of DCT design

We will now implement this design using the design separation methodology steps defined above.
Note: Please refer to Libero SoC Design Flow and ChipPlanner help topics in Libero to understand the design flow and floor-planning terminology followed in subsequent steps.

Creating HDL subsystems

The first step is to achieve logical separation of various subsystems. Create logically separate HDL modules corresponding to the system.

In this example we have defined four such subsystems, DCT8AAN1.vhd, DCT_BUF10.vhd, DCT8AAN2.vhd and DCT_BUF12.vhd, which are independent of each other. They communicate with each other using interconnect signals as shown in Figure 14.

Once you have identified the subsystems to be implemented using design separation, import these modules into a Libero project.

Create a new Libero project for the FPGA device chosen for the design. In this example, we are implementing the design using a M2GL025, 484 FBGA device. Import the four HDL files using File → Import → HDL Source file menu into the Libero Project.

Creating Blocks

The next step is to create a Block for each subsystem of this design.

To create a Block, select a module to be the root module. For example, select DCT8AAN1 as shown in Figure 15.

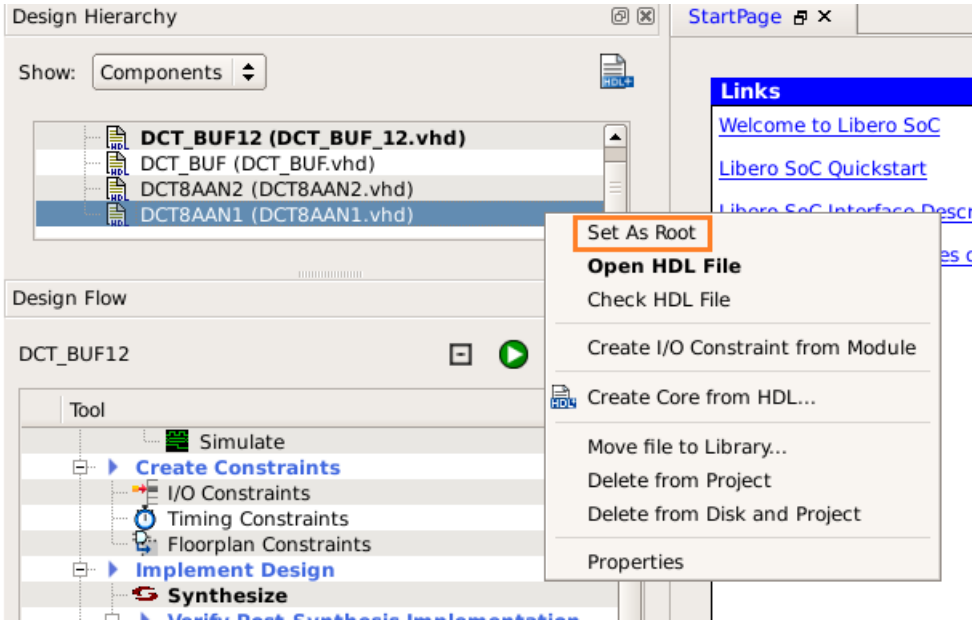


Figure 15. Selecting a module as the root of a Block

Enable Block flow for this module, using Project → Project Settings → Design Flow → Enable Block Creation as shown in Figure 16. Also, enable Use Standalone Initialization for MDDR/FDDR/SERDES Peripherals.

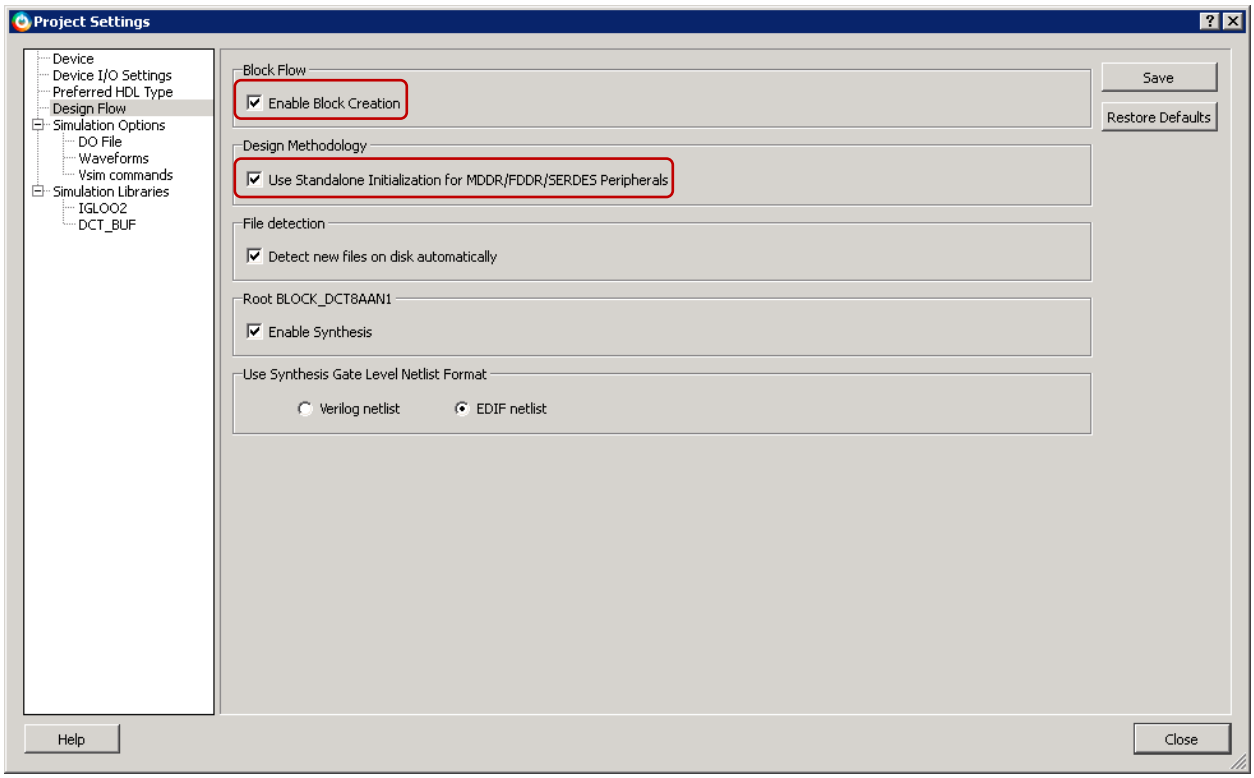


Figure 16. Project Settings for Block Creation

The Publish Block option will now be enabled in the design flow, as shown in Figure 17.

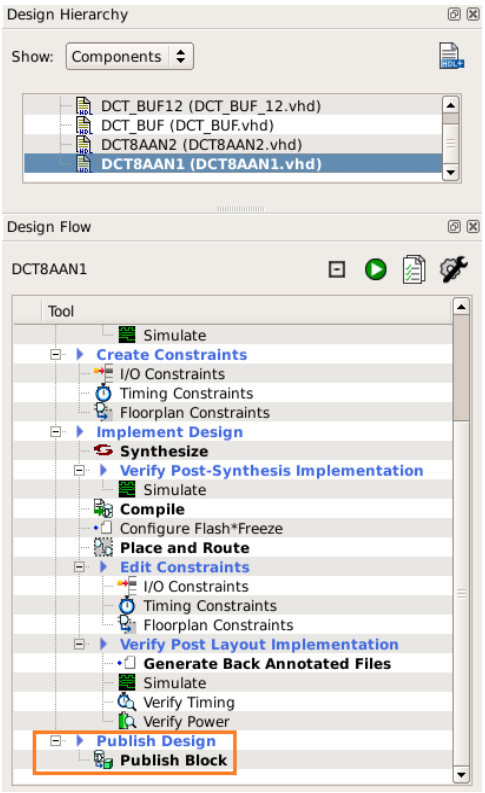


Figure 17. Publish Block option enabled in the design

If a Block requires physical I/O resources you must specify explicit instantiation of I/O resources. All logic within the Microsemi Design Separation methodology must be incorporated within an isolated region. As such, all physical I/O resources must be associated with an isolated region. I/Os can be inserted through direct instantiation or through insertion of I/O buffers through the SmartDesign feature.

In this example, we are going to use SmartDesign feature to insert I/Os to top level signals. DCT8AAN1 subsystem has following port list:

- CLK , RST , START , EN, DATA_IN :- Top-level Input Signals
- RDY , DATA_OUT :- IRS signals to DCT_BUF10 block

Since each block should have I/Os inserted for its top-level I/O signals, we need to insert I/O ports to the top-Level signals of this subsystem.

Create a SmartDesign with name TOP_DCT8AAN1. Instantiate the DCT8AAN1 HDL component into TOP_DCT8AAN1.

For each top-level Input signal assign an INBUF Macro. This will instantiate a single I/O port for each of the signals. The input signal DATA_IN has a width of 8 Bits. To promote this signal bus to a top-Level port, use the I/O Configurator from the catalog.

Figure 18 shows the location of the I/O Configurator in the catalog.

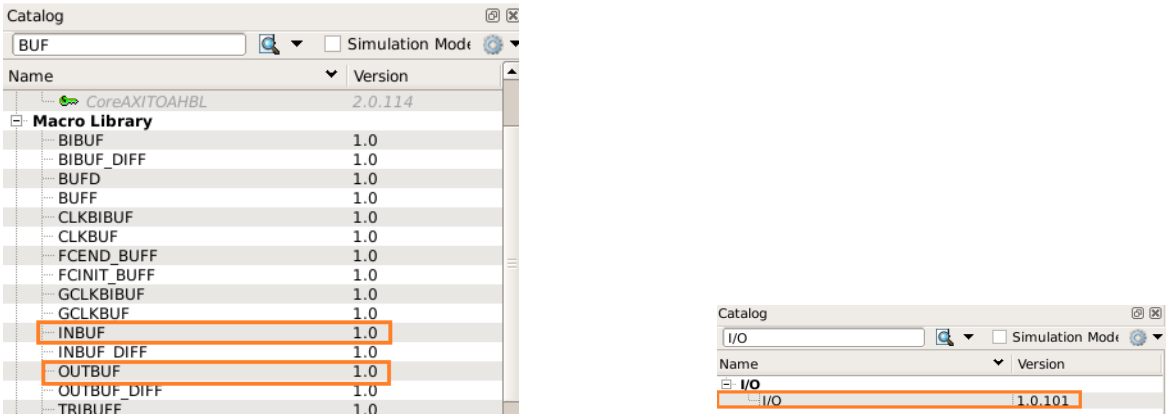


Figure 18. I/O Configurator in the Catalog

Once you instantiate all required I/O macros, rename them to a unique name and connect these I/O pads to respective ports of DCT8AAN1 instance.

Since RDY_AAN1 and DATA_OUT_AAN1 are interconnect signals, right-click the ports and promote them to the top.

Figure 19 shows the schematics of the TOP_DCT8AAN1 SmartDesign component.

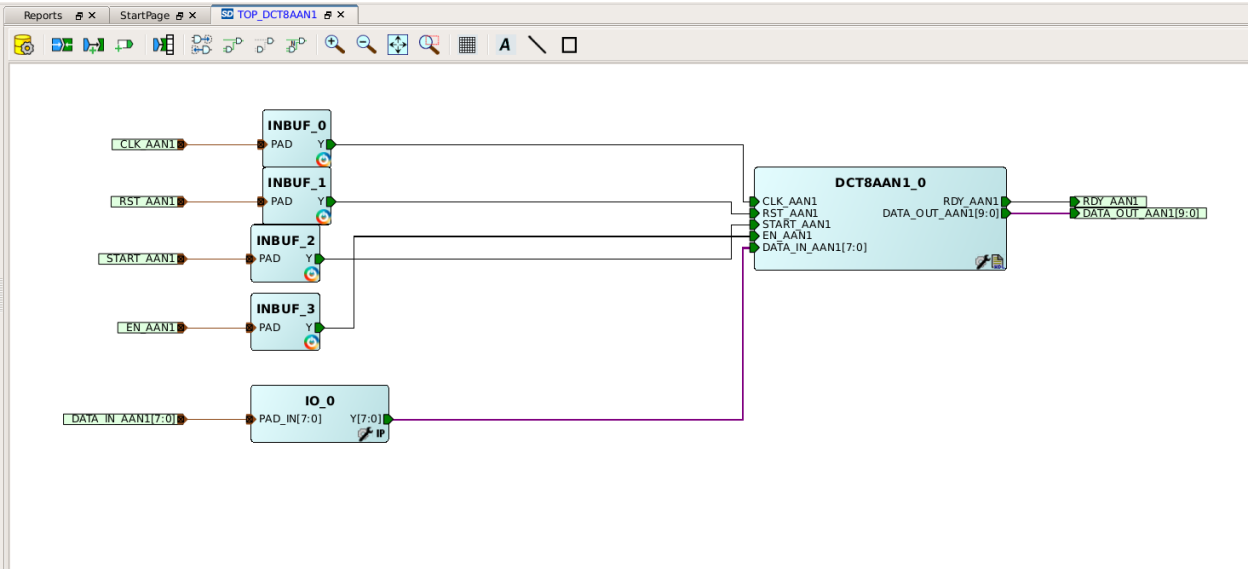


Figure 19. SmartDesign Component for DCT8AAN1 subsystem along with I/Os

Generate TOP_DCT8AAN1, set this module as the root module, and Enable Block Creation for this module as described in the previous section.

Publish Block

Once you have created the module with I/Os inserted, you can now publish the Block. Run Synthesis, and Compile the subsystem (or its wrapper SD or HDL component). You can check for timing closure on the Block. Publish the Block without place and route information.

In this example, for TOP_DCT8AAN1, we ran Synthesis and Compile. Study the Compile report generated for this subsystem to estimate the area occupied and resources needed by this design.

Disable Placement and Routing information in Publish Block → Configure Options (as shown in Figure 20). Publish the block. Placement and routing information is not needed until the Block is integrated with the top level project. Enabling these options results in a longer Compile cycle.

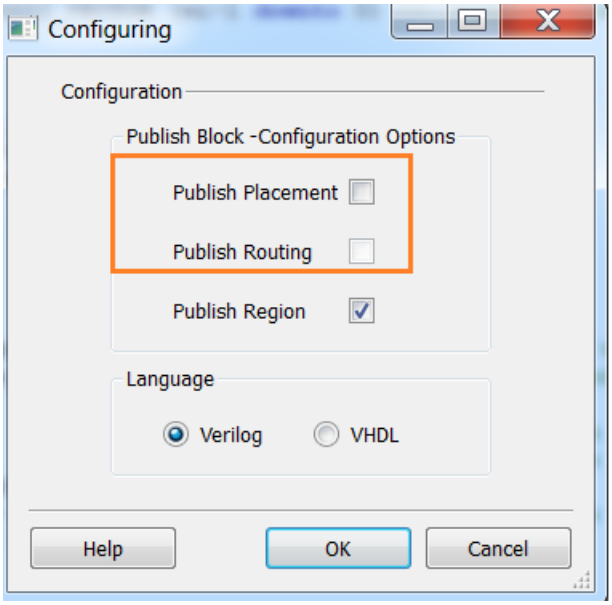


Figure 20. Publishing the block without Placement and Routing information

Figure 21 shows the state of the completed design flow after you have published the Block.

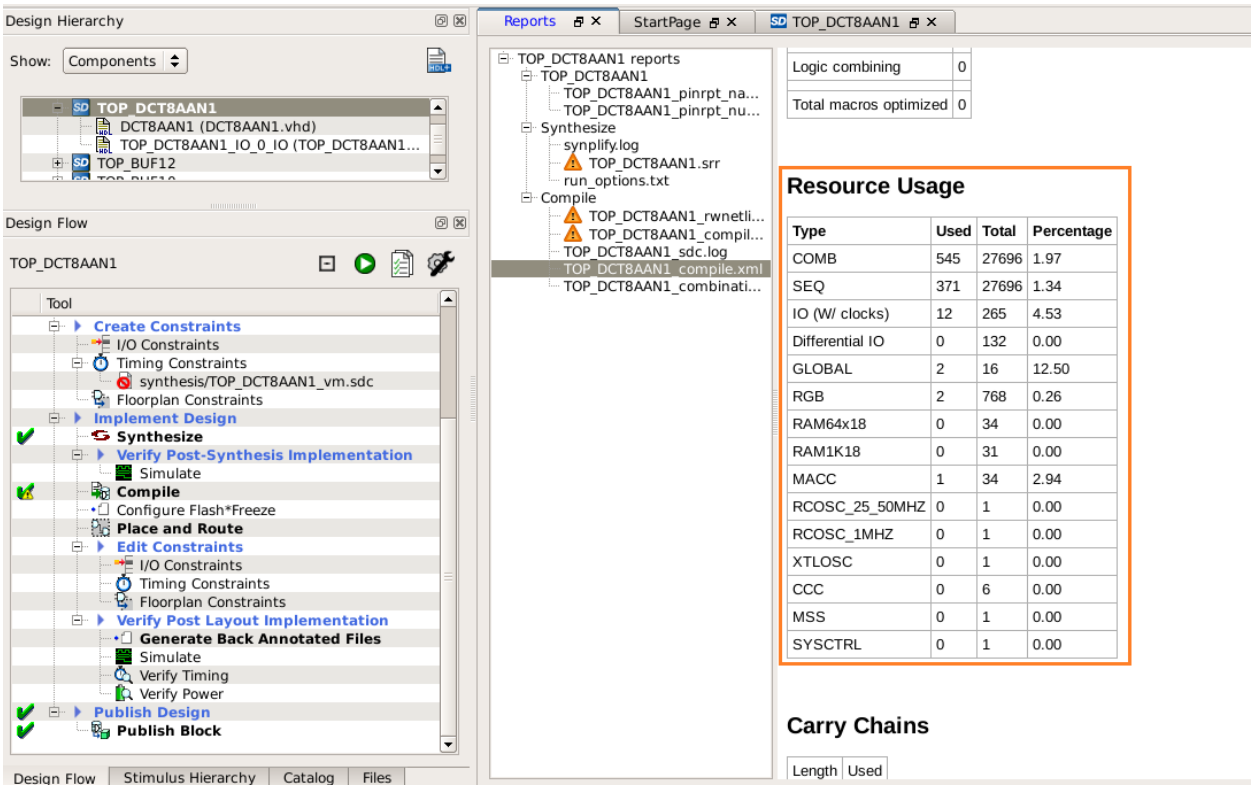


Figure 21. Completed Block Flow along with Resource Usage from Compile

Publishing a Block creates a <block_name>.cxz file in the <project_path>/designer/<block_name>/export directory.

For the above subsystem, Libero will create the TOP_DCT8AAN1.cxz file in the export directory under the designer directory of the Project location as shown in Figure 22.

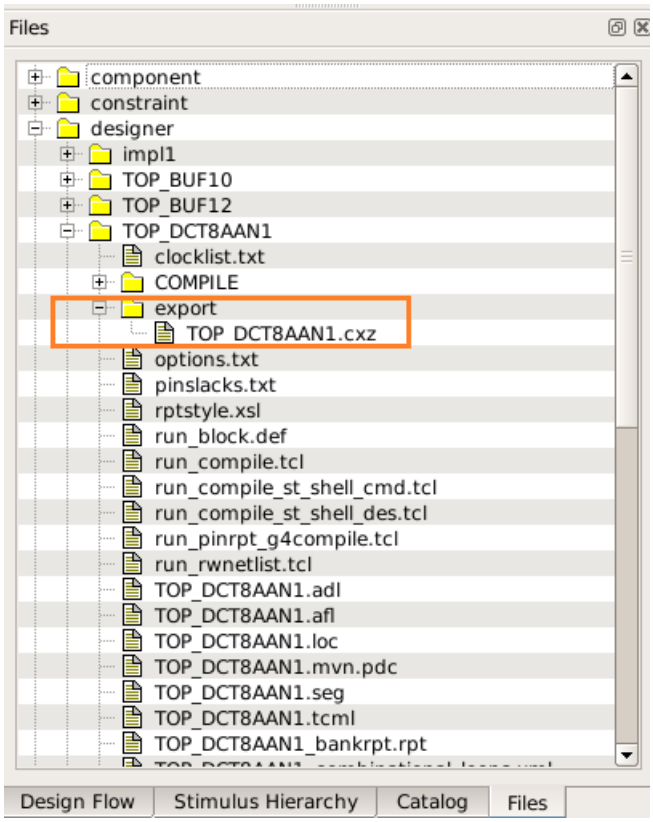


Figure 22. Published Block as .cxz file in export Directory

Repeat this step for the other three subsystems, and publish Blocks for each of them. Use the Block names shown in the table below.

Subsystem	Block Name
DCT_BUF10	TOP_BUF10
DCT8AAN2	TOP_DCT8AAN2
DCT_BUF12	TOP_BUF12

Creating a Top-level Design

Once you have published all the subsystem Blocks, create a new Libero project for the top-level design. You must write a HDL module instantiating these Blocks with corresponding interconnects between them defined.

In this example, we have written a top level module DCT_AAN.vhd, instantiating these Blocks along with required interconnects. Figure 23 shows an example description of the top-level DCT_AAN module, with the instance names of each Block highlighted.

```
entity DCT_AAN is
	port(
		CLK_AAN1 : in STD_LOGIC;
		RST_AAN1 : in STD_LOGIC;
		START_AAN1: in STD_LOGIC;
		EN_AAN1: in STD_LOGIC
		CLK_AAN2 : in STD_LOGIC;
		RST_AAN2 : in STD_LOGIC;
		EN_AAN2: in STD_LOGIC;
		CLK_BUF10 : in STD_LOGIC;
		RST_BUF10 : in STD_LOGIC;
		EN_BUF10: in STD_LOGIC;
		CLK_BUF12 : in STD_LOGIC;
		RST_BUF12 : in STD_LOGIC;
		EN_BUF12: in STD_LOGIC;
		DATA_IN_AAN1 : in STD_LOGIC_VECTOR(7 downto 0);
		RDY_BUF12 : out STD_LOGIC;
		DATA_OUT_BUF12 : out STD_LOGIC_VECTOR(11 downto 0);
		CLK_SMIP : in std_logic;
		RESET_SMIP : in std_logic;
		HEARTBEAT : out std_logic
	);
end DCT_AAN;

architecture CONSTR of DCT_AAN is

	component TOP_DCT8AAN1 is
		port (
			CLK_AAN1: in STD_LOGIC;
			RST_AAN1: in STD_LOGIC;
			START_AAN1: in STD_LOGIC;
			EN_AAN1: in STD_LOGIC;
			DATA_IN_AAN1: in STD_LOGIC_VECTOR (7 downto 0);
			RDY_AAN1: out STD_LOGIC;
			DATA_OUT_AAN1: out STD_LOGIC_VECTOR (9 downto 0)
		);
	end component ;

	component TOP_DCT8AAN2 is
		port (
			CLK_AAN2: in STD_LOGIC;
			RST_AAN2: in STD_LOGIC;
			START: in STD_LOGIC;
			EN_AAN2: in STD_LOGIC;
			DATA_IN: in STD_LOGIC_VECTOR (9 downto 0);
			RDY: out STD_LOGIC;
			DATA_OUT: out STD_LOGIC_VECTOR (11 downto 0)
		);
	end component;

	component TOP_BUF10 is
		port (
			CLK_BUF10: in STD_LOGIC;
			RST_BUF10: in STD_LOGIC;
			START_BUF: in STD_LOGIC;
			EN_BUF10: in STD_LOGIC;
			DATA_IN_BUF: in STD_LOGIC_VECTOR (9 downto 0);
			RDY_BUF: out STD_LOGIC;
			DATA_OUT_BUF: out STD_LOGIC_VECTOR (9 downto 0)
		);
	end component ;

	component TOP_BUF12 is
		port (
			CLK_BUF12: in STD_LOGIC;
			RST_BUF12: in STD_LOGIC;
			START_BUF12: in STD_LOGIC;
			EN_BUF12: in STD_LOGIC;
			DATA_IN_BUF12: in STD_LOGIC_VECTOR (11 downto 0);
			RDY_BUF12: out STD_LOGIC; -
			DATA_OUT_BUF_12: out STD_LOGIC_VECTOR (11 downto 0)
```

```
        );
    end component ;

    component TOP_coreSMIP is
        -- Port list
        port(
            DEVRST_N : in std_logic;
            ERASE_N : in std_logic;
            ERASE_P : in std_logic;
            RESET_WATCHDOG_N : in std_logic;
            RESET_WATCHDOG_P : in std_logic;
            DIGEST_ERROR : out std_logic;
            HEARTBEAT : out std_logic;
            JTAG_ACTIVE : out std_logic;
            LOCK_TAMPER_DETECT : out std_logic;
            MASTER_ALARM : out std_logic;
            MESH_SHORT_ERROR : out std_logic;
            PASSCODE_ERROR : out std_logic;
            POWERUP_DIGEST_ERROR : out std_logic;
            SC_ROM_DIGEST_ERROR : out std_logic
        );
    end component;

    signal rdy1,rdy2,rdy3 : STD_LOGIC;
    signal data1,data1r: STD_LOGIC_VECTOR (9 downto 0);
    signal data2: STD_LOGIC_VECTOR (11 downto 0);

begin
    U_ST1: TOP_DCT8AAN1
        port map(CLK_AAN1 => CLK_AAN1, RST_AAN1 => RST_AAN1,
            START_AAN1 =>START_AAN1, -- after this impulse the 0-th datum is sampled
            EN_AAN1 =>EN_AAN1,       -- operation enable to slow-down the calculations
            DATA_IN_AAN1 =>DATA_IN_AAN1,
            RDY_AAN1 =>rdy1,      -- delayed START impulse, just after it the 0-th
            DATA_OUT_AAN1=>data1 -- output data
        );

    U_B1:TOP_BUF10
        port map(CLK_BUF10=>CLK_BUF10, RST_BUF10=>RST_BUF10,
            START_BUF=>rdy1,      -- after this impulse the 0-th datum is sampled
            EN_BUF10=>EN_BUF10,  -- operation enable to slow-down the calculations
            DATA_IN_BUF=>data1,
            RDY_BUF=>rdy2,        -- delayed START impulse, after it the 0-th result
            DATA_OUT_BUF=>data1r -- output data
        );

    U_ST2: TOP_DCT8AAN2
        port map(CLK_AAN2=>CLK_AAN2, RST_AAN2=>RST_AAN2,
            START =>rdy2,         -- after this impulse the 0-th datum is sampled
            EN_AAN2 =>EN_AAN2,    -- operation enable to slow-down the calculations
            DATA_IN =>data1r,
            RDY =>rdy3,           -- delayed START impulse, after it the 0-th result
            DATA_OUT=>data2       -- output data
        );

    U_B2:TOP_BUF12
        port map(CLK_BUF12=>CLK_BUF12, RST_BUF12=>RST_BUF12,
            START_BUF12=>rdy3,    -- after this impulse the 0-th datum is sampled
            EN_BUF12=>EN_BUF12,   -- operation enable to slow-down the calculations
            DATA_IN_BUF12=>data2,
            RDY_BUF12=>RDY_BUF12, -- delayed START impulse, after it the 0-th result
            DATA_OUT_BUF_12=>DATA_OUT_BUF12 -- output data
        );

    SMIP_BLOCK:TOP_coreSMIP
        port map(DEVRST_N=>DEVRST_N,
            ERASE_N=>ERASE_N,
            ERASE_P=>ERASE_P,
            RESET_WATCHDOG_N=>RESET_WATCHDOG_N,
            RESET_WATCHDOG_P=>RESET_WATCHDOG_P,
            DIGEST_ERROR=>DIGEST_ERROR,
            HEARTBEAT=>HEARTBEAT,
            JTAG_ACTIVE=>JTAG_ACTIVE,
            LOCK_TAMPER_DETECT=>LOCK_TAMPER_DETECT,
```

```
                    MASTER_ALARM=>MASTER_ALARM,
                    MESH_SHORT_ERROR=>MESH_SHORT_ERROR,
                    PASSCODE_ERROR=>PASSCODE_ERROR,
                    POWERUP_DIGEST_ERROR=>POWERUP_DIGEST_ERROR,
                    SC_ROM_DIGEST_ERROR=>SC_ROM_DIGEST_ERROR
                );
end CONSTR;
```

Figure 23. Top-level Design description in VHDL

Set the top-level module as the root module, and import all Blocks (<block_name>.cxz files) using File → Import → Blocks in Libero. Figure 24 shows the Design Hierarchy of the top-level DCT_AAN module with all Blocks instantiated.

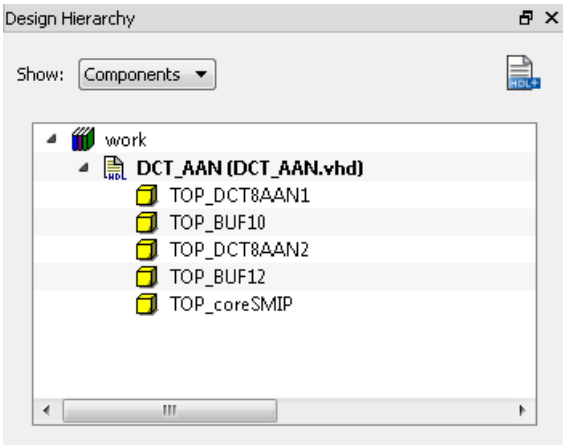


Figure 24. Top-level Design Hierarchy

Uncheck the boxes for Enable Block Creation and Enable Synthesis for this module (in Project Settings as shown in Figure 25).

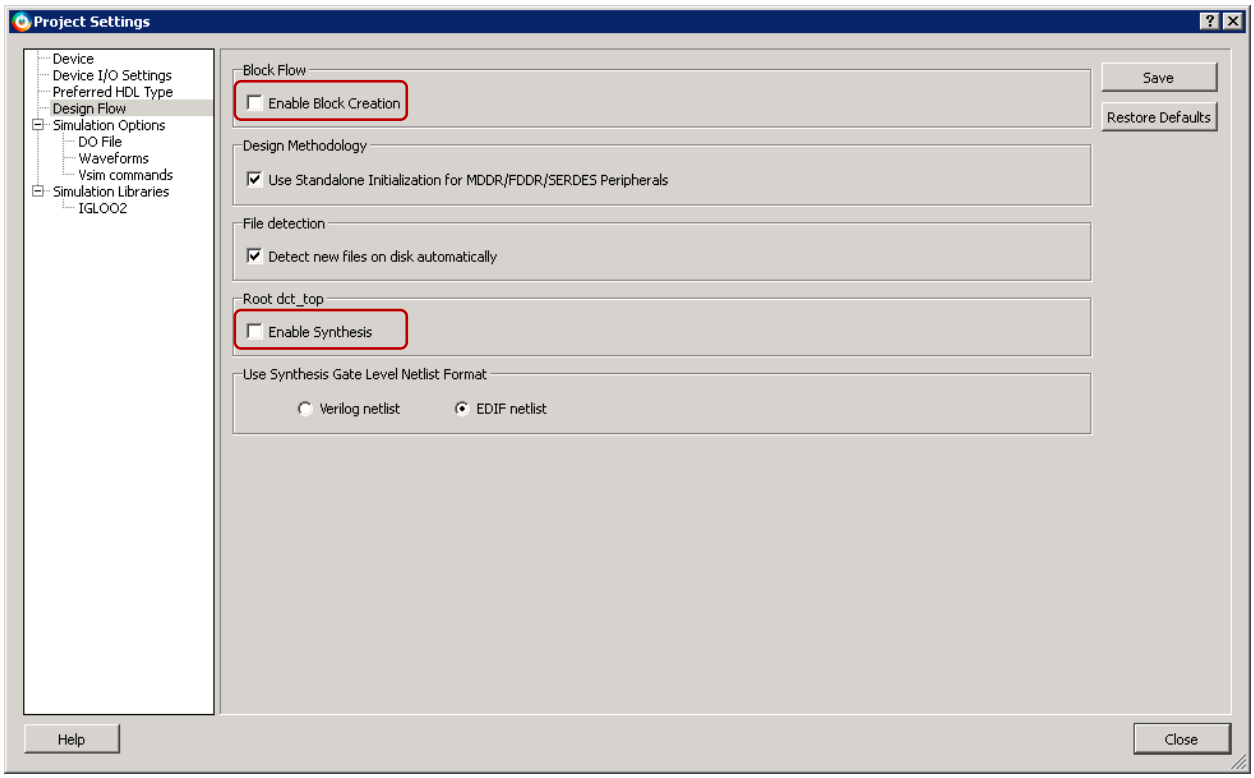


Figure 25. Project Settings for the Top-level Design

In addition, configure Compile to enable Design Separation Methodology. From Compile → Configure as shown in Figure 27. Complete Compile for the top-level module (DCT_AAN in this example).

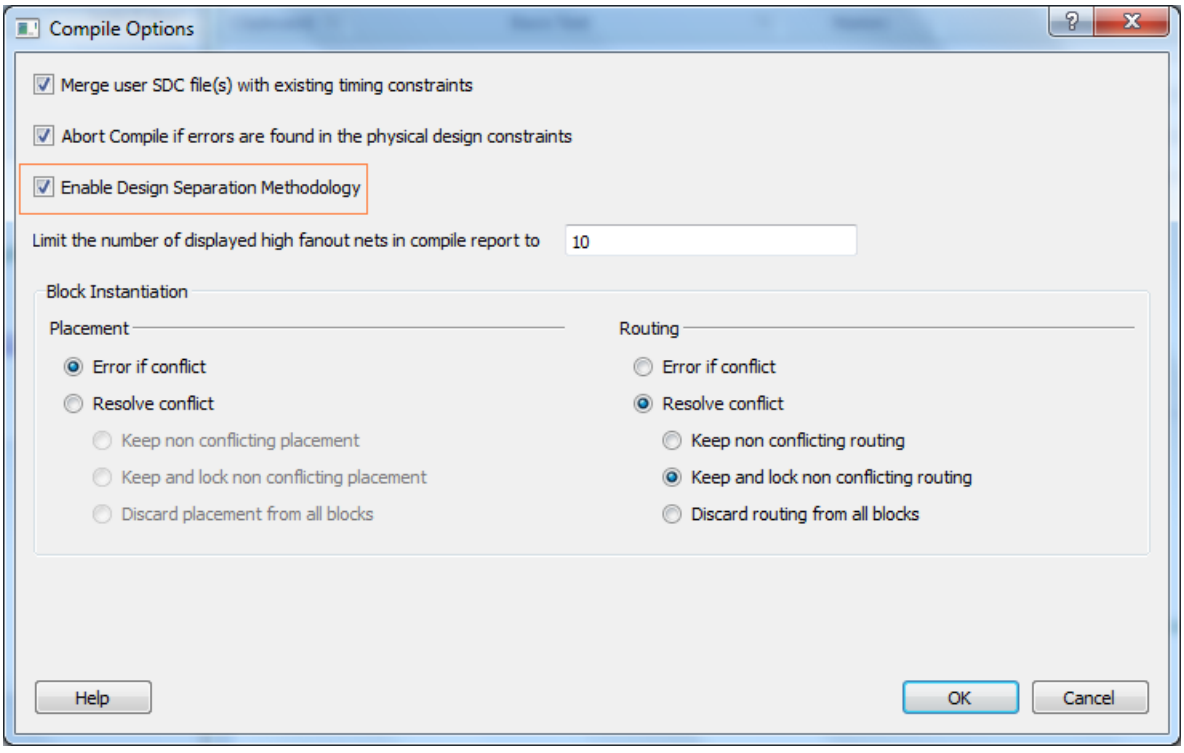


Figure 26. Enabling Design Separation Methodology before Compile

Floorplanning Design with Separation Regions

Each block of the design must be assigned to a separation region. You can define separation regions in ChipPlanner or use a PDC file.

More information on floorplanning a design using ChipPlanner can be found in the Libero online help.

In this example we are going to define a separation region using ChipPlanner for instance U_ST1 (corresponding to the TOP_DCT8AAN1 block).

Invoke ChipPlanner from Implement Design → Edit Constraints → Floorplan Constraints in Design Flow window, as shown in Figure 27. This opens ChipPlanner interactively.

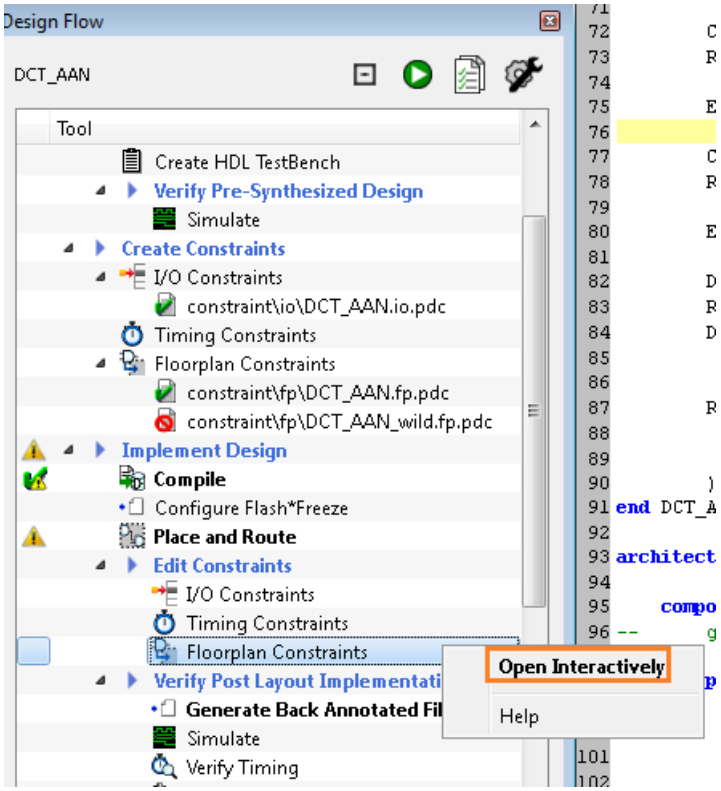


Figure 27. Opening ChipPlanner

Create a separation region for each of the blocks according to the estimate obtained from the resource usage reports described above.

Note: Refer to Creating routing regions in the ChipPlanner help topics in Libero for more information.

Figure 11 shows an example floorplan for the design. In this example, we will create an inclusive routing region constraint for instance U_ST1, as shown in Figure 28.

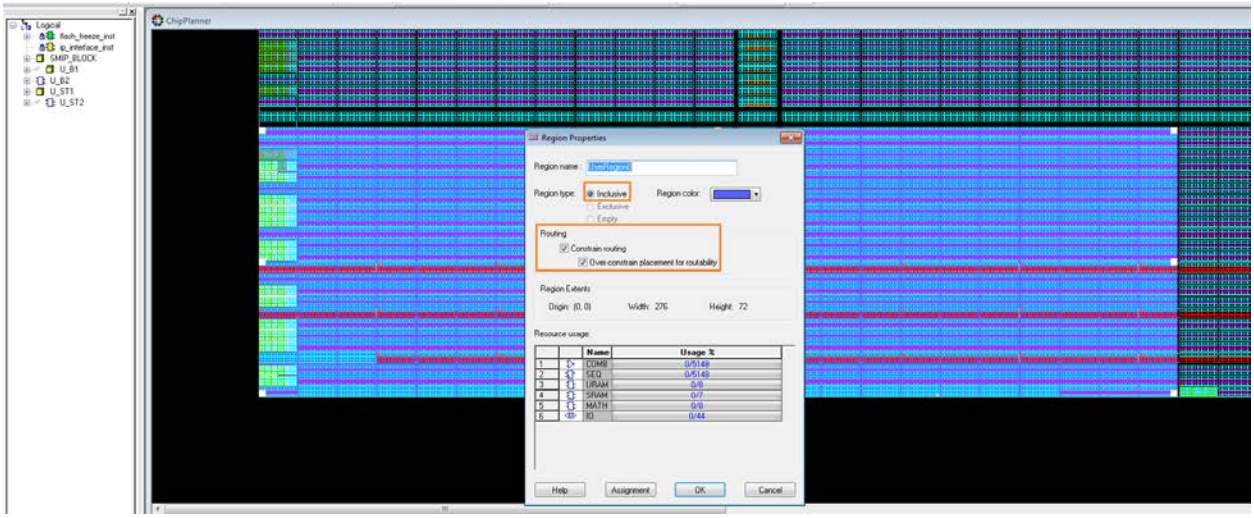


Figure 28. Creating an Inclusive Routing Region

Create separation regions for all blocks of the design. Then assign Block instances to the respective separation region. For this example, the floorplan would look like Figure 29.

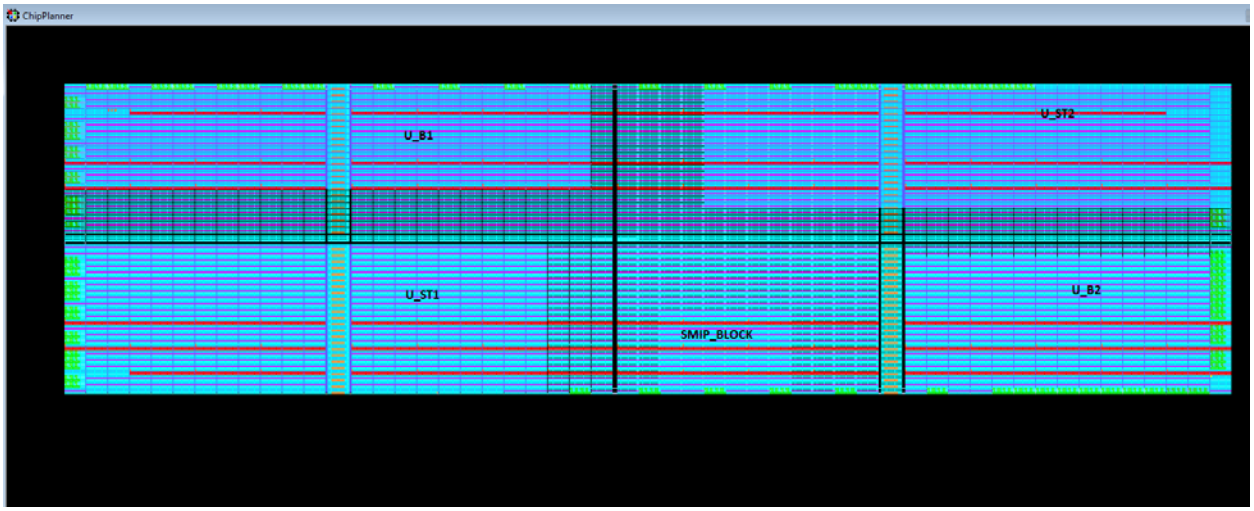


Figure 29. Floor-plan of Design with Separation Regions defined

Once you create separation regions corresponding to the Blocks you must create IRS regions for each set of connections between the Blocks. In the example, U_ST1 connects to U_B1, U_B1 connects to U_ST2 and U_ST2 connects to U_B2. Hence, we need to define three sets of IRS regions.

IRS region is an inclusive routing region that is created in a similar manner as the separation regions. Figure 30 is an example of one such IRS region for instances U_ST1 to U_B1.

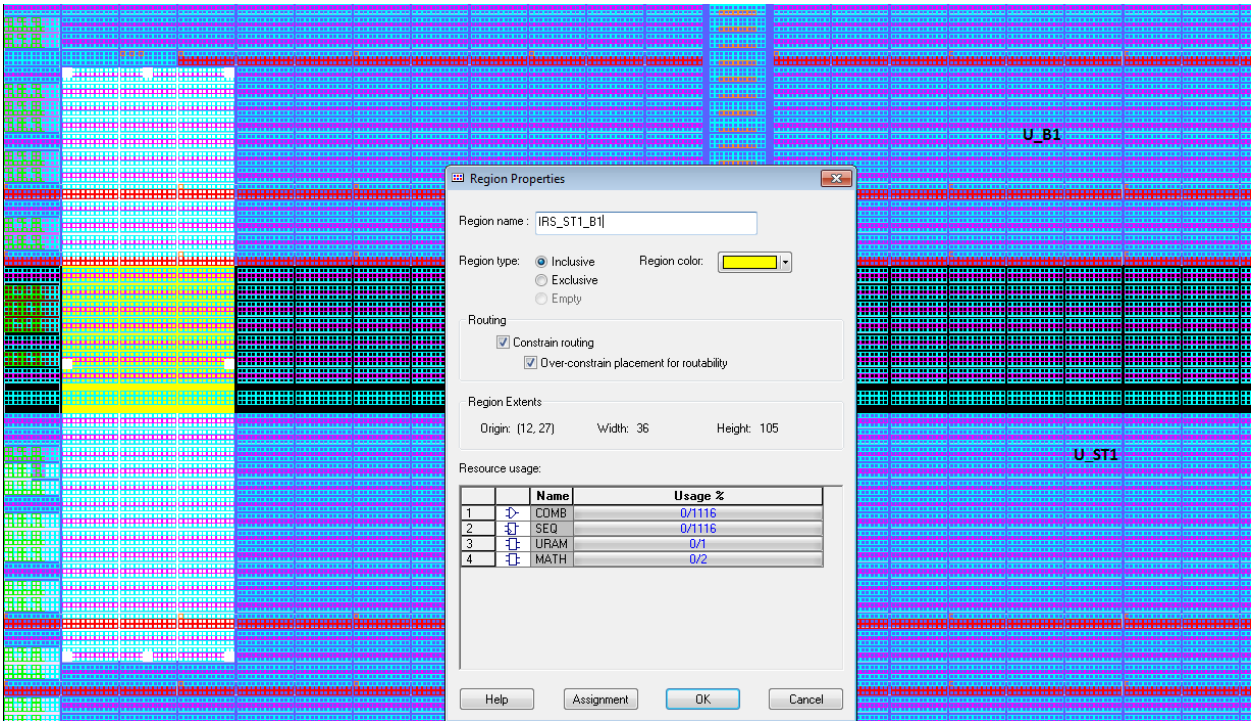


Figure 30. Creating IRS region between two blocks

Similarly, create IRS regions between U_B1 to U_ST2 and U_ST2 to U_B2 Blocks. Assign valid IRS net macros to the respective IRS Regions.

A complete floorplan of the example design will look similar to Figure 31. The separation between each region should be at least the required number of clusters to satisfy the separation criteria.

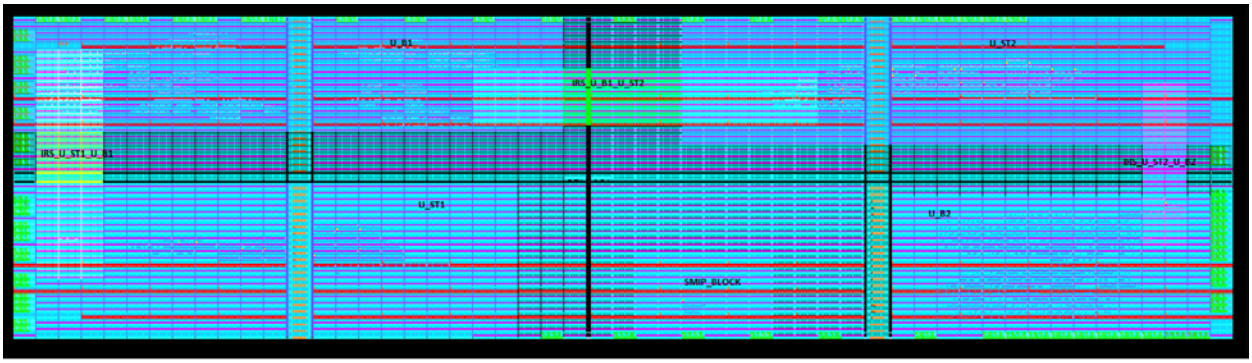


Figure 31. Complete Floor-plan of Design

Figure 32 shows an example PDC file that can be used to implement the above floorplan. Note that separation regions are assigned by their highest level hierarchy name via the assign_region command. IRS nets are assigned using wildcards through the assign_net_macros command.

```
define_region -name U_ST1_Region -route YES   0   0 275  68
define_region -name U_B1_Region  -route YES   0  84 275 146
define_region -name U_ST2_Region -route YES 324  84 635 146
define_region -name U_B2_Region  -route YES 324   0 563  68
define_region -name SMIP_Region  -route YES 612   0 635  26
```

```
define_region -name IRS_B1_ST2   -route YES 190  84 443 146
define_region -name IRS_ST1_B1   -route YES   0   0 131 146
define_region -name IRS_ST2_B2   -route YES 492   0 563 146

assign_region U_ST1_Region U_ST1
assign_region U_B1_Region  U_B1
assign_region U_ST2_Region U_ST2
assign_region U_B2_Region  U_B2
assign_region SMIP_Region  SMIP_BLOCK

assign_net_macros IRS_B1_ST2 rdy2 data1r<*> -include_driver yes
assign_net_macros IRS_ST1_B1 rdy1 data1<*>  -include_driver yes
assign_net_macros IRS_ST2_B2 rdy3 data2<*>  -include_driver yes
```

Figure 32. Example PDC file for floor-plan

Complete Place and Route

Once you have completed the floor plan, edit Timing constraints and run Place and Route until you achieve timing closure on the design.

Configure Security Settings and Generate Programming file

Once you complete place and route, you must extract the design information so as to execute MSVT.

Configure the security and programming options per system requirements. Configure Security and Programming Options → Security Policy Manager → Configure Options → Debug Policy as shown in Figure 33.

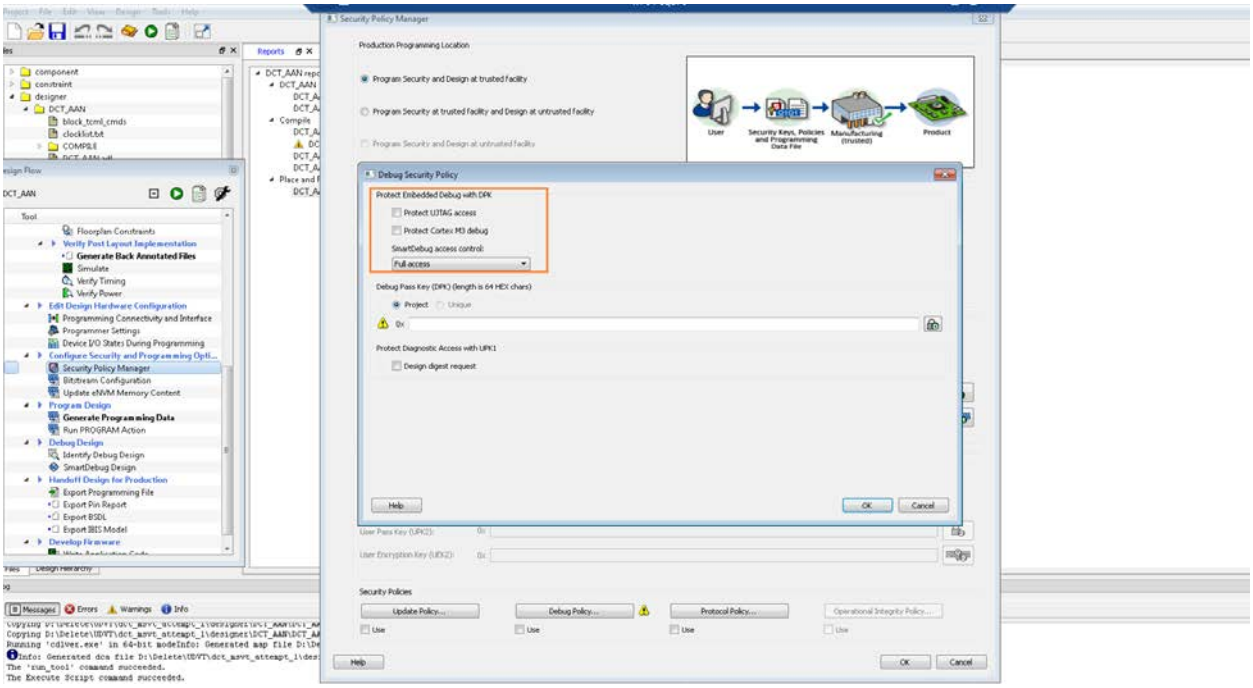


Figure 33. Security Settings before Programming

Export the Programming file from the Program Design → Export Programming File menu.

This generates a programming file as well as files required for MSVT. Libero exports these files into the <project_path>/designer/DCT_AAN/DCT_AAN.msvt.dtf directory and creates a parameter file in <project_path>/designer/DCT_AAN/msvt.param file, as shown in Figure 34. The Msvt.param file contains a list of parameters that you can adjust before executing MSVT.

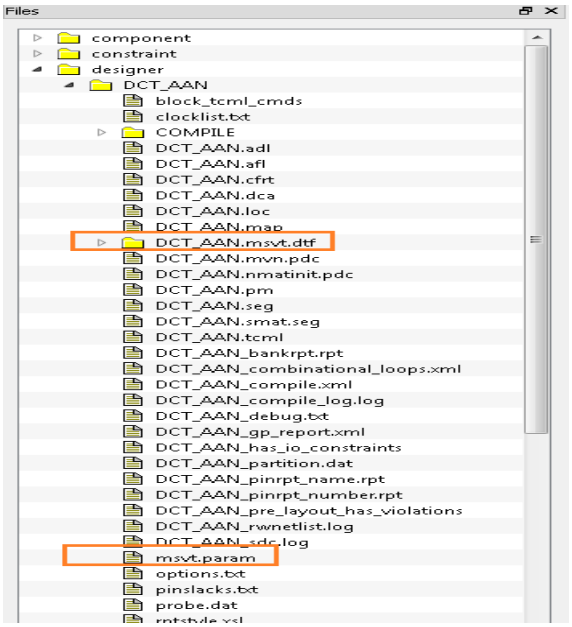


Figure 34. Generated MSVT files

Executing MSVT

You can now run the Microsemi Separation Verification Tool (MSVT) to verify that the design adheres to the required separation criteria.

MSVT is an independent tool provided as part of the Libero installation. It is invoked from <Libero_Path>/bin64/msvt_check. The tool is executed from the command line. A required argument is –p along with the path to the msvt.param file that has been generated from Libero.

The command you must execute to verify the design using MSVT is:

<Libero_path>/bin64/msvt_check –p <project_path>/designer/DCT_AAN/msvt.param [–o msvt_check.log]

This command prints an exhaustive report to msvt_check.log file given with the –o argument or to *stdout* if –o is omitted.

On successful completion of this command, you will see a final message of either “MSVT Check failed” indicating that the design has not met one or more of separation criteria or “MSVT Check succeeded”, indicating the design has met all separation criteria.

Since we have followed the guidelines described in Microsemi Design Separation Methodology in the example, Figure 35 shows the conclusion of MSVT output indicating that the design has been verified for the given separation criteria.

Design has met 1 switches separation requirement

MSVT Check succeeded.
Number of errors: 0

Figure 35. Output Report of MSVT for Design

A – 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, 408.643.6913

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

Visit the Customer Support website (www.microsemi.com/soc/support/search/default.aspx) for more information and support. Many answers available on the searchable web resource include diagrams, illustrations, and links to other resources on the website.

Website

You can browse a variety of technical and non-technical information on the SoC 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. 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.

Microsemi Corporate Headquarters
One Enterprise, Aliso Viejo CA 92656 USA
Within the USA: +1 (949) 380-6100
Sales: +1 (949) 380-6136
Fax: +1 (949) 215-4996

Microsemi Corporation (NASDAQ: MSCC) offers a comprehensive portfolio of semiconductor solutions for: aerospace, defense and security; enterprise and communications; and industrial and alternative energy markets. Products include high-performance, high-reliability analog and RF devices, mixed signal and RF integrated circuits, customizable SoCs, FPGAs, and complete subsystems. Microsemi is headquartered in Aliso Viejo, Calif. Learn more at **www.microsemi.com**.

5-02-00551-1/09.14