

Design Separation Methodology User Guide

Introduction

This guide 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, a system with security and safety-critical requirements is built with each critical subsystem implemented using multiple integrated circuits (ICs). With each critical subsystem as an independent IC, fault and reliability analysis is simplified. In a traditional Field Programmable Gate Array (FPGA) design, netlists generated for placeand-route often are 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 subsystems within an FPGA design might need to be isolated to simplify failure analysis and prevent propagation of faults from one subsystem adversely affecting another.

The Microchip Design Separation methodology provides a way to create independent critical subsystems on a single FPGA. Functional blocks that must be independent can be isolated physically from other functional elements in the FPGA using place-and-route constraints in the Libero SoC software. The following figure shows a top-level view for implementing a security and safety-critical application in a Microchip FPGA.

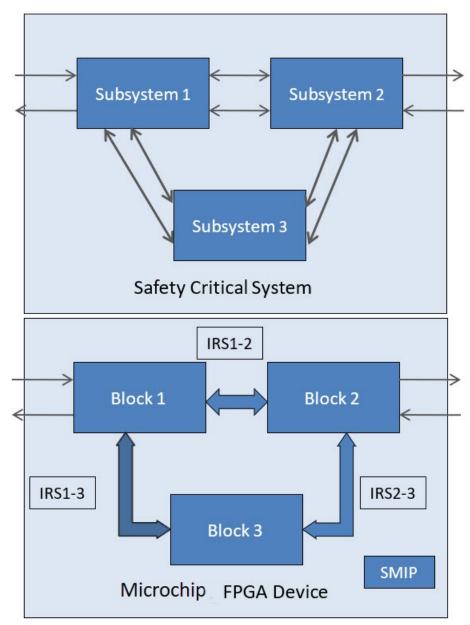


Figure 1. Implementing Security and Safety-Critical Applications in Microchip FPGAs

Table of Contents

Intr	oductio	n	1
1.	Desig	n Methodology	5
	1.1.	Design Separation Methodology Components	5
	1.2.	Design Separation Methodology Steps	
	1.3.	Creating Blocks	
	1.4.	Assigning I/Os to the Block	9
	1.5.	Optional CoreSMIP Block	. 11
	1.6.	Creating a Top-level Design	. 11
	1.7.	FloorPlanning with Design Separation Regions	. 12
	1.8.	IRS Regions	. 15
	1.9.	Considerations for Global Clock Resources	.16
	1.10.	Initialization of Hard ASIC Blocks	.16
	1.11.	Complete Place-and-Route	. 17
	1.12.	Configuring Security Settings and Generating the Programming File	.17
	1.13.	Auditing by MSVT	.17
	1.14.	Executing MSVT	.18
	1.15.	Further Considerations and Adjustments	. 19
2.	Exam	ple	. 21
	2.1.	Creating HDL Subsystems	.21
	2.2.	Creating Blocks	
	2.3.	Publishing the Block	
	2.4.	Creating a Top-level Design	
	2.5.	Floorplanning Design with Separation Regions	
	2.6.	Complete Place-and-Route	
	2.7.	Configure Security Settings and Generate the Programming File	
	2.8.	Execute MSVT	. 38
3.	Revis	on History	. 41
4.		chip FPGA Technical Support	
4.			
	4.1.	Customer Service.	
	4.2.	Customer Technical Support	
	4.3. 4.4.	Website Outside the U.S.	
	4.4.		.42
The	Micro	chip Website	.43
Pro	duct C	nange Notification Service	.43
Cus	stomer	Support	. 43
Mic	rochip	Devices Code Protection Feature	.43
Lec	al Noti	ce	. 44
		íS	
110	aoman		

Quality Management System	45
Worldwide Sales and Service	46

1. Design Methodology

The following topics describe the design methodology.

1.1 Design Separation Methodology Components

The Microchip Design Separation methodology comprises of the following features:

- Ability to create independent subsystems.
- Ability to validate that isolation.
- Ability to monitor for faults.

The Design Separation methodology leverages an existing design methodology referred to as a "Block flow" to achieve isolated functions. Block flow is a bottom-up design methodology that allows an incremental design approach. In a Block flow compile, component modules in a design are compiled and optimized in independent stages from the rest of the project. After compiling, the component modules are published as a netlist (or "Block"), and then imported to a top-level project for integration with other modules in the larger system design.

Because the Block must be compiled with all the required physical resources, resource reservation is a key component of Block flow. Routing reservation and logic reservation are both constraint options available from the Block flow methodology. As a result, creating isolated subsystems for security and safety-critical applications is an application of Block flow. All critical subsystems 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. **Note:** The width of the guard-band depends on individual project requirements.

Signal connections to another module (known as "Inter-Region Signals" within this flow) are assigned to another resource-reserved region called the "IRS Region". This region overlaps the source and sink regions. In this way, the Inter-Region Signals are members of the source region, the sink region, and the IRS Region. The IRS Region acts as a constrained routing channel.

A separate tool known as the Microchip Separation Verification Tool (MSVT) checks that a design meets the separation requirements defined by the system requirements of the design. MSVT is an independent tool included with the Libero installation. Libero SoC generates the parameter file MSVT.param automatically, which describes the Blocks 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 on the separation requirement defined by the user.

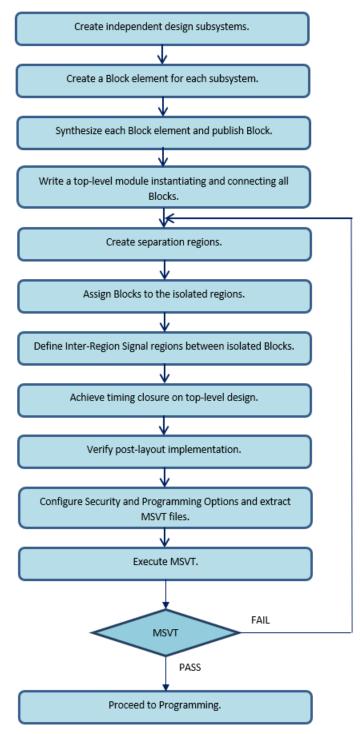
Anti-Tamper (AT) must be considered in addition to the isolation of critical design subsystems. PolarFire[®], SmartFusion[®]2, and IGLOO[®]2 devices come with standard robust design security, a critical portion of which is AT and fault detection. PolarFire, SmartFusion2 and IGLOO2 devices include mechanisms that allow an FPGA design to monitor the integrity of the device during operation.

Fault detection is a critical part of security-and safety-critical systems. To address this requirement, an IP core ties together the relevant AT hooks in CoreSMIP_PF (PolarFire) and CoreSMIP (SmartFusion2 and IGLOO2) devices.

1.2 Design Separation Methodology Steps

The following flow chart shows the various steps of the design separation methodology.

Figure 1-1. Microchip Design Separation Methodology



- 1. Create an RTL description for each subsystem. 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. Define each independent subsystem 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.

Note: 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. For each Block, run Synthesis and Compile. Assess the size and shape of suitable regions based on the types of I/O, count and length of cascaded Math blocks, RAM count, PLLs, peripherals and fabric resource usage. Publish the Block without place-and-route.
- 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 Project Settings... > Design Flow > Design Separation > Enable Design Separation Methodology.
- 5. For each Block in the design, create separation regions by specifying region constraints using Chip Planner or defining regions in a PDC file.
- 6. Assign Blocks to the isolated regions.
- 7. If two Blocks interact with each other, create an overlapping IRS region constraint connecting the Blocks. These IRS regions should also be physically isolated from other blocks and IRS regions. Assign IRS nets to each respective IRS region.
- 8. Enter the necessary timing constraints. Perform design iterations to achieve timing closure.
- 9. Verify all aspects of timing and power.
- 10. Generate back-annotated files and perform post-layout simulations when required.
- 11. Configure security and programming options before generating the programming file. This step also exports information for MSVT.
- 12. Run MSVT from the command line. If MSVT fails, re-examine the floorplan and iterate the entire design flow with corrected region constraints.

1.3 Creating Blocks

A security and safety-critical application may consist of one or more independent subsystems. Using Hardware Description Language (HDL), define each subsystem to be independent 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:

- 1. Right-click the target module in the **Design Hierarchy** tab and choose **Set as root**.
- 2. Enable block creation from Project Settings (see the following figure).

Figure 1-2. Project Settings for Block Creation

evice selection				
evice settings	HDL source files language options			Save
esign flow halysis operating conditions mulation options DO file Waveforms	syntax option if your Verilog files contain System Ve		L-2008 and VHDL-93.	Discard
Vsim commands Timescale	Verilog			
mulation libraries 	C Verlog 2001	System Verlig	Multi File Compilation Unit	
COREAPB3 LIB	WHDL			
eneral Settings obal Include Paths	© WHDL-2008	C VHDL-93		
	HDL generated files language options			
		rtDesign components and post-layout gate level netlists use the preferred l	anguage option.	
	(* Verlog	C VHDL		
	* Verling Block flow	C VHDL		
	Block flow	be instantiated into another design. A block component may not contain I/C	cells and cannot be programmed by itself.	
	Block flow	be instantiated into another design. A block component may not contain I/C	cells and cannot be programmed by itself.	
	Block flow Block flow Block flow enables you to publish a reusable component that can It could include timing constraints, physical constraints, placement F Finable block creation	be instantiated into another design. A block component may not contain I/C	I cells and cannot be programmed by itself.	
	Block flow Block flow Block flow enables you to publish a reusable component that can It could include timing constraints, physical constraints, placement If Prable block creation Root block4	be instantiated into another design. A block component may not contain I/C) cells and cannot be programmed by itself.	
	Block flow Block flow enables you to publish a reusable component that can It could include timing constraints, physical constraints, placement IF Enable block creation Root block4 IF Enable synthesis	be instantiated into another design. A block component may not contain I/C	i cells and cannot be programmed by itself.	
	Block flow Block flow castles you to publish a reusable component that can I it could relate the grants physical constraints, phasement If could relate the constraints, physical constraints, phasement P Enable block creation Root block 4 P Enable synthesis Enable PPGA Hardware Breakpoint Auto Instantistion	be instantiated into another design. A block component may not contain I/C	I cells and cannot be programmed by itself.	
	Block flow Block flow enables you to publish a resultative component that cont It could include timing constraints, physical constraints, placement IP Enable block creation Root block4 IP Enable synthesis IP Enable PPGA Hardware Breakpoint Auto Instantistion Design Separation	be instantiated into another design. A block component may not contain I/C	scels and cannot be programmed by itself.	
	Block flow Constraints, physical constraints, physical constraints, placement P Block flow Root block 4 P Enable synthesis Enable PEGA Hardware Breakpoint Auto Instantiation Design Separation Enable Design Separation Methodology	be instantiated into another design. A block component may not contain I/C	I cells and cannot be programmed by itself.	
	Block flow Constraints, physical component that can I to constraints, physical constraints, physical constraints, physical constraints, physical constraints, physical constraints, physical Root block 4 P Enable lock creation P Enable physical constraints, physical constrationts, physical constrationts, physical constrationts, physical	be instantiated into another design. A block component may not contan I/C for routing.	t cels and cannot be programmed by itself.	

Note: Block flow is a bottom-up design methodology. The Block attribute in the Block flow identifies components in an HDL hierarchy to be reused within a team-based design flow as a modular resource.

- 3. If a Block uses physical I/O pins, define those physical resources as part of that block. This requires explicit definition of I/O to be assigned to the Block using I/O pads. Use direct instantiation of an I/O buffer within the module in question or from the Catalog in Libero's SmartDesign tool.
- 4. For each module that has 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, count and length of cascaded Math blocks, RAM count, PLLs, peripherals and fabric resource usage.
- 5. Optional: Enter timing constraints and run place-and-route followed by timing analysis to achieve timing closure for each individual Block. This step indicates the difficulty of timing closure at the top level of the design.
- 6. The Block is ready to be published. Because these Blocks will be assigned to isolated separation regions (explained in subsequent sections), publish the Block without placement and routing information. Configure Publish Block options to exclude placement and routing information, as shown in the following figure.

Figure 1-3. Configuring Publish Block Options

Folder:	990\Desk	Browse		
-Publish Block (Configuration —			
Publish P	lacement			
Publish R	outing			
Publish R	legion			
-Language				
Verilog		C VHDL		

Libero exports the <block_name>.cxz file to the <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 to instantiate the Block.

1.4 Assigning I/Os to the Block

Signals that route to physical I/O pins within each module should belong to the corresponding Block. For 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. Therefore, 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. For more information, see 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.

Note:

Do not insert I/O buffer on ports that are used to interconnect with other blocks (that is, IRS nets).

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

- 1. Create a SmartDesign and instantiate the module in SmartDesign.
- 2. 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.
- 3. If ports belong to a bus, use the I/O configurator with required width and type of buffer.
- 4. After required macros are instantiated in SmartDesign, connect the ports of the design with the respective macros.
- 5. Rename the I/O pads with names defined in the module. Generate SmartDesign.
- 6. Set the generated SmartDesign as the root module and create a Block using this module as described in 2.2 Creating Blocks.

The following figure shows a SmartDesign component in which a subsystem has been instantiated and top-level ports are assigned to its I/Os using OUTBUF macros.

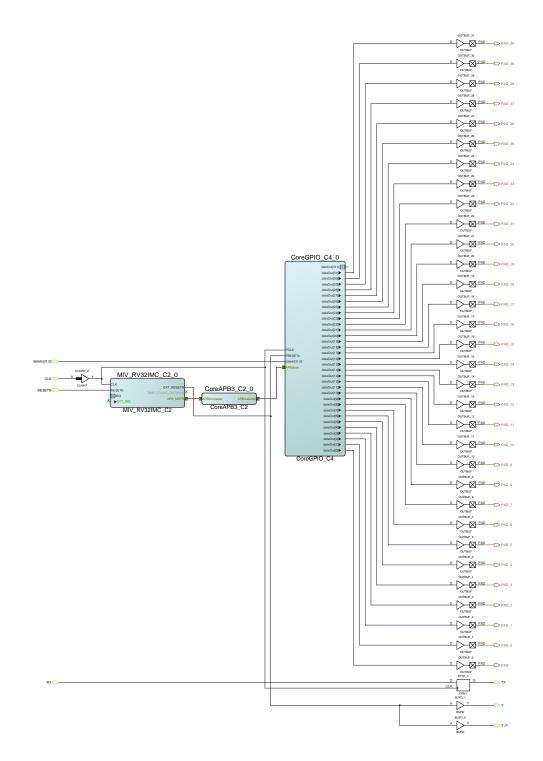


Figure 1-4. SmartDesign Instantiating Subsystem Along with I/Os

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

Because lower level modules are compiled independently from the top-level design, ensure unique I/O pin names across the design are used in the lower level project. Microchip recommends checking the I/O pin

names across the project to ensure uniqueness when building a lower level project. It is important to realize that some cores have pins that are not true single point inputs or outputs that can also be fed back internally. Therefore, they should be carefully placed between blocks.

The cells connected to IRS regions must be isolated from other IRS regions 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.

1.5 Optional CoreSMIP Block

The Security Monitor IP (CoreSMIP or CoreSMIP_PF) is a core provided by Microchip for tamper detection to enhance the security of the system. For more information, see the *CoreSMIP or CoreSMIP_PF User Guide*. CoreSMIP and CoreSMIP_PF is present in the catalog under the Tamper section.

The Design Separation methodology requires each subsystem to be defined as a Block. Therefore, if your design includes CoreSMIP or CoreSMIP_PF, create such a block using the same steps as the other Blocks.

1.6 Creating a Top-level Design

After all Blocks are published, create a new Libero project for the top-level design using the following steps.

- 1. Create SmartDesign block where you instantiate all the individual blocks and connect their IRS signals. This top-level module should contain instantiations of all Blocks along with interconnects between them to replicate a complete system.
- Set this top-level module as the root module in Libero and disable Block Creation. Navigate to the File > Import > Blocks menu and import all published subsystem Blocks (<block_name>.cxz files) into this design. Typically, you need not run Synthesis, because all published Blocks have already completed Synthesis and have I/Os assigned to them.

The following figure shows **Project Settings** required for top-level design with Block Creation and Synthesis disabled and **Enable Design Separation Methodology** enabled.

Figure 1-5. Project Settings for Top-level Design

Project settings	-		×
Device selection Device settings Design flow Analysis operating condit. Simulation options DO file Waveforms Vsim commands	HDL source files language options Libero SoC supports mixed-HDL language designs; you can import Verilog and VHDL in the same project. For Verilog files you can enable the System Verilog syntax option if your Verilog files contain System Verilog constructs. For VHDL files, you may choose between VHDL-2008 and VHDL-93. Verilog	Discar	
- Timescale - Simulation libraries - PolarFire - COREAHBLITE_LIB - COREAPB3_LIB	© System Verlog C Verlog 2001 WHDL © VHDL-2008 C VHDL-93		
General Settings Global Include Paths	HDL generated files language options - HDL files generated by Libero SoC such as configured cores, SmartDesign components and post-layout gate level netlists use the preferred language option. C Verlog C Verlog		
	Block flow Block flow enables you to publish a reusable component that can be instantiated into another design. A block component may not contain I/O cells and cannot be programmed by itself. It could include timing constraints, physical constraints, placement or routing.		
	Root block4 Enable synthess Enable FPGA Hardware Breakpoint Auto Instantiation		
	Design Separation		
	Reports Maximum number of high fanout nets to be displayed: 10		
	Image: Abort flow if errors are found in Physical Design Constraints (PDC) Image: Abort flow if errors are found in Timing Constraints (SDC)		
Help		Ck	ise

1.7 FloorPlanning with Design Separation Regions

After you create the top-level design with subsystem Blocks, floorplan the design by defining separation regions and IRS regions. In a design that follows the Microchip 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 which may be an exclusive or inclusive constraint.
- 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.

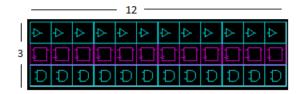
You can use Chip Planner to create regions or create them with PDC commands. Chip Planner is the floorplanning tool used to create and edit regions on the chip and assign logic to these regions.

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, Math blocks, PLL and peripherals 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.

PolarFire, SmartFusion2 and IGLOO2 FPGA architecture is cluster-based. 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 Chip Planner coordinate system, each Logic Element component has a unique coordinate. As such, each Cluster occupies an area of 12x3. The following figure shows a single cluster as shown in the Chip Planner with dimensions noted. The granularity of Chip Planner region sizes is one cluster.

Figure 1-6. One Cluster of PolarFire, SmartFusion2 and IGLOO2 FPGA



You can define regions for each Block using either the Chip Planner or a Physical Design Constraints (PDC) file. The size of each region should accommodate all resources used by a given block, including all embedded hard blocks such as I/Os, RAM, Math blocks, PLL and peripheral blocks.

Note: LSRAM and Math blocks take up a footprint of three clusters in the FPGA floorplans. The makeup of such embedded blocks include the hard IP resource itself abutted to a set of Interface Clusters. The Interface Clusters help route signals to and from the embedded hard Block to the rest of the fabric array. The following figures provide details of the makeup of a embedded hard Block and its corresponding visualization within the Chip Planner floorplan, respectively.

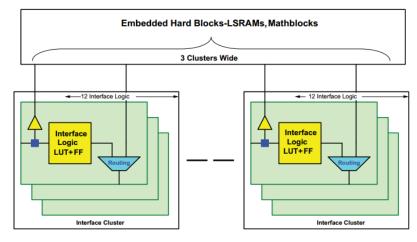
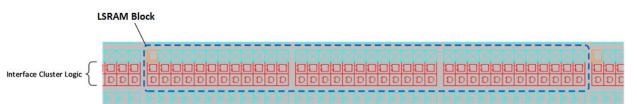


Figure 1-7. Interface Cluster for an Embedded Hard IP Block

Figure 1-8. LSRAM Block as Shown in Chip Planner

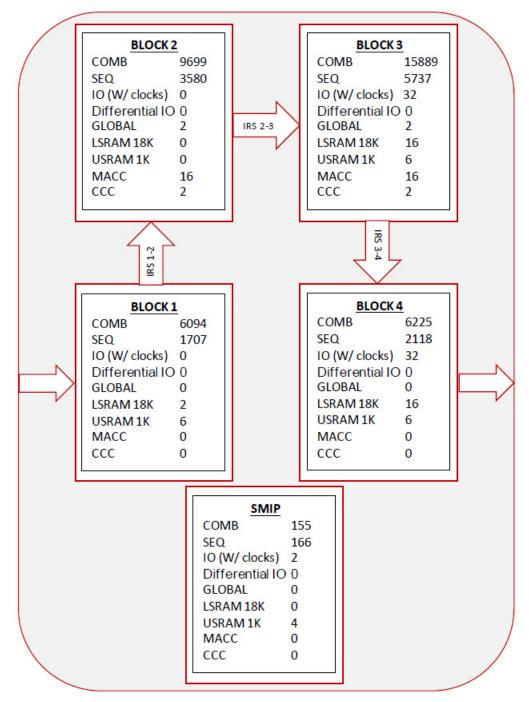


An isolated region constraint must include the entire embedded hard Block resource within its boundaries for the resource to be usable within the target region. Use non-rectangular regions to efficiently allocate a floorplan to include these embedded hard blocks.

More information about the fabric architecture for FPGA devices can be found in the PolarFire FPGA fabric user guide or the SmartFusion2 FPGA fabric user guide.

The following figures provide a sample floorplan from a sample design using a PDC file and the floorplan as shown in the Chip Planner, respectively. The granularity of placement units are logic modules in the Chip Planner coordinate system and the granularity of region sizes is clusters. Therefore, regions must be a multiple of 12 in the horizontal direction and a multiple of 3 in the vertical direction.





Following are the details of Physical design constraints file of the top-level design. Note the regions are defined with -route true to constrain routing.

```
define_region -region_name Blocklregion -type exclusive -color 2143338688 -route true - push_place true -x1 456 -y1 195 -x2 1631 -y2 371 define_region -region name Block2region -type exclusive -color 2143338688 -route true - push_place true -x1 1752 -y1 189 -x2 2435 -y2 377 define_region -region_name Block3region -type exclusive -color 2143338688 -route true - push_place true -x1 0 -y1 0 -x2 335 -y2 41 \
```

```
0 -y1 42 -x2 1067 -y2 161 \
                 -x1
                 -x1 804 -y1
                                0 -x2 1067 -y2 41
define_region -region_name Block4region -type exclusive -color 2143338688 -route true -
push place true -x1 1200 -y1 0 -x2 2351 -y2 158
define region -region name SMIPregion -type exclusive -color 2143338688 -route true -
push place true -x1 384 -y1 0 -x2 755 -y2 11
define_region -region_name IBR1_2 -type inclusive -color 2147442270 -route true -push place
false -x1 1584 -y1 282 -x2 2027 -y2 362
define_region -region_name IBR1_3 -type inclusive -color 2147442270 -route true -push_place
false -x1 636 -y1 102 -x2 851 -y2 239
define region -region name IBR1 4 -type inclusive -color 2143338688 -route true -push place
false -x1 1356 -y1 126 -x2 1499 -y2 245
define_region -region_name IBR2_4 -type inclusive -color 2147442270 -route true -push_place
false -x1 2148 -y1 105 -x2 2327 -y2 266
define_region -region_name IBR3_4 -type inclusive -color 2147442270 -route true -push_place
false -x1 888 -y1 45 -x2 1463 -y2 98
assign_region -region_name Block1region -inst_name block1_0
assign_region -region_name Block2region -inst_name block2_0
assign_region -region_name Block3region -inst_name block3_0
assign region -region name Block4region -inst name block4 0
assign region -region name Block4region -inst_name RESETN_ibuf
assign region -region name SMIPregion -inst_name pf_smip_0
assign net macros -region name IBR1 2 -net name block1 0 APBmslave0 PENABLE
include driver true
assign net macros -region name IBR1 2 -net name block1 0 APBmslave0 PSELx
include driver true
assign_net_macros -region_name IBR1_2 -net_name block1_0_APBmslave0_PWRITE
include driver true
assign net macros -region name IBR1 2 -net name block1 0 APBmslave0 PREADY
include driver true
assign_net_macros -region_name IBR1_2 -net_name {block1 0 APBmslave0 PADDR[*]}
include driver true
assign net macros -region name IBR1 2 -net name {block1 0 APBmslave0 PRDATA[*]} -
include driver true
assign net macros -region name IBR1 2 -net name {block1 0 APBmslave0 PWDATA[*]} -
include driver true
assign_net_macros -region_name IBR1_3 -net_name {block1_0_dataOut[*]}
                                                                              -include driver true
assign net macros -region name IBR1 3 -net name {block3 0 dataOut 0[*]} -include driver true
assign_net_macros -region_name IBR1_4 -net_name block4_0_TX -include_driver true
assign_net_macros -region_name IBR1_4 -net_name block4_0_Y_0 -include_driver true
assign_net_macros -region_name IBR1_4 -net_name block1_0_TX -include_driver true assign_net_macros -region_name IBR2_4 -net_name block4_0_Y -include_driver true
assign net macros -region name IBR3 4 -net name {block3 0 dataOut[*]} -include driver true
```

For more information about floorplaning with the Chip Planner and PDC syntax, see the Chip Planner online help in Libero SoC.

Figure 1-10. Sample Floorplan of Top-level Design



1.8 IRS Regions

Since each Block is defined in an isolated region, it must be ensured 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 the same requirements as the separation regions mentioned in 1.7 FloorPlanning with Design Separation Regions. 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 regions 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 extent of separation required depends on your system requirements.

The cells connected to IRS regions must be isolated from other IRS regions 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.

1.9 Considerations for Global Clock Resources

The global clock network on PolarFire, SmartFusion2, and IGLOO2 FPGA 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
- Transceivers
- FPGA fabric routing

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

The design separation flow only considers physical isolation of the 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 design separation, global signals that are common to multiple regions (such as clock and Reset) need not be separated from any other signal. If the location of a global resource like PLL or CCC overlaps with a block region, you must make the type of the block region inclusive. Alternatively, you could include the global resource within the block and bring out the global net for distributing to other blocks.

Note: High fanout-signals from the fabric array often are promoted automatically to the global network. In such cases, you may want 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. You can control the promotion and demotion of signals using Synthesis attributes. You can also configure Synthesis options in Libero SoC 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, PLL, DLL, clock divider, or an RC oscillator) are not audited. Most inputs to the CCC are from hard blocks, such as from a dedicated I/O pin or the RC oscillator and 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 the 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 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. The width of such regions is determined by the span of the RGB output.

1.10 Initialization of Hard ASIC Blocks

SmartFusion2 and IGLOO2 FPGA 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 through 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, the standalone initialization flow must be used with the design separation flow.

For more information about standalone initialization of peripheral blocks, see the *Standalone Peripheral Initialization User Guide*.

1.11 Complete Place-and-Route

After 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 can also change the floorplan and iterate through the design. Standard FPGA design practices like incremental flow are available. Make sure all criteria required for separation of design remains intact while changing the design floorplan.

1.12 Configuring Security Settings and Generating the Programming File

You can use the Security Policy Manager (SPM) to set design security attributes after completing place-and-route and before you generate programming files. This procedure 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, see the PolarFire FPGA Security User Guide or the SmartFusion2 FPGA Security User Guide.

1.13 Auditing by MSVT

MSVT is a standalone tool provided with the Libero installation. It is used to verify that the design meets design separation requirements.

The tool accepts as input the design database and a parameter file that is generated every time a programming file is generated. The parameter file describes the isolation regions in the design as well as the inter-region signals between isolation regions. This file is generated when the **Enable Design Separation Methodology** check box in the **Project Settings** 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 the corresponding sets of IRS signals.

MSVT prints a comprehensive report on each Block and the corresponding IRS regions being verified. If any Block or IRS signals do not satisfy minimum separation criteria, the tool reports details of affected instances. For more information about the MSVT output report, see 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 such a case, do the following:

- Identify instances that cause violations in the MSVT output, and modify the design floorplan accordingly.
- · Recompile the design to generate a new place-and-routed netlist.
- Verify the modified design using the MSVT tool.

If the design satisfies separation criteria, the MSVT output reports MSVT Check succeeded to indicate that the required design separation has been achieved in the design.

1.14 Executing MSVT

The msvt.param file contains the parameters required by MSVT to verify design separation. The following shows a sample parameter file.

```
11
11
    This is input parameters file for MSVT Check program
11
DEVICE = MPF300TS
DESIGN = SD Top.msvt
VERIFY BLOCKS = block4 0 block2 0 block3_0 block1_0 pf_smip_0 // empty list means all blocks
in design will be verified
REQUIRED SEPARATION = 1
MAX VIOLATIONS PER REPORT SECTION = 1
IRS block4 0 block2 0 = block4 0 Y
IRS block2_0 block4_0 =
IRS block4_0 block3_0 =
IRS block3 0 block4 0 = block3 0 dataOut[31] block3 0 dataOut[30] block3 0 dataOut[29]
         block3 0 dataOut[28] block3 0 dataOut[27] block3 0 dataOut[26] block3 0 dataOut[25]
block3 0 dataOut[24] block3 0 dataOut[27] block3 0 dataOut[26] block3 0 dataOut[25]
block3 0 dataOut[24] block3 0 dataOut[23] block3 0 dataOut[22] block3 0 dataOut[21]
block3 0 dataOut[20] block3 0 dataOut[19] block3 0 dataOut[18] block3 0 dataOut[17]
block3 0 dataOut[16] block3 0 dataOut[15] block3 0 dataOut[14] block3 0 dataOut[16]
         block3_0_dataOut[12] block3_0_dataOut[11] block3_0_dataOut[10] block3_0_dataOut[9]
         block3_0_dataOut[8] block3_0_dataOut[7] block3_0_dataOut[6] block3_0_dataOut[5]
block3_0_dataOut[4] block3_0_dataOut[3] block3_0_dataOut[2] block3_0_dataOut[1]
block3_0_dataOut[0]
IRS block4 0 block1 0 = block4 0 TX block4 0 Y 0
IRS block1 0 block4 0 = block1 0 TX
IRS block4 0 pf smip 0 =
IRS pf_smip_0 block4_0 =
IRS block2 \overline{0} block3 \overline{0} = IRS block3 \overline{0} block2 \overline{0} =
IRS block2 0 block1 0 = block1 0 APBmslave0 PRDATA[31] block1 0 APBmslave0 PRDATA[30]
block1 0_APBmslave0_PRDATA[29] block1_0_APBmslave0_PRDATA[28]
block1_0_APBmslave0_PRDATA[27]
         block1 0 APBmslave0 PRDATA[26] block1 0 APBmslave0 PRDATA[25]
block1 0 APBmslave0 PRDATA[24]
         block1 0 APBmslave0 PRDATA[23] block1 0 APBmslave0 PRDATA[22]
block1 0 APBmslave0 PRDATA[21]
         block1 0 APBmslave0_PRDATA[20] block1_0_APBmslave0_PRDATA[19]
block1_0_APBmslave0_PRDATA[18]
         block1 0 APBmslave0 PRDATA[17] block1 0 APBmslave0 PRDATA[16]
block1_0_APBmslave0 PRDATA[15]
_ block1_0_APBmslave0_PRDATA[14] block1_0_APBmslave0_PRDATA[13]
block1_0_APBmslave0_PRDATA[12]
         block1 0 APBmslave0 PRDATA[11] block1 0 APBmslave0 PRDATA[10]
block1 0 APBmslave0 PRDATA[9]
         block1 0 APBmslave0 PRDATA[8] block1 0 APBmslave0 PRDATA[7]
block1 0 APBmslave0 PRDATA[6]
         block1 0 APBmslave0 PRDATA[5] block1 0 APBmslave0 PRDATA[4]
block1 0 APBmslave0 PRDATA[3]
         block1 0 APBmslave0 PRDATA[2] block1 0 APBmslave0 PRDATA[1]
block1_0_APBmslave0 PRDATA[0]
         block1_0_APBmslave0_PREADY
IRS block1_0 block2_0 = block1_0_APBmslave0_PADDR[11] block1 0 APBmslave0 PADDR[10]
         block1 0 APBmslave0 PADDR[9] block1 0 APBmslave0 PADDR[8]
block1 0 APBmslave0 PADDR[7]
         block1 0 APBmslave0 PADDR[6] block1 0 APBmslave0 PADDR[5]
block1 0 APBmslave0 PADDR[4]
         block1 0 APBmslave0 PADDR[3] block1 0 APBmslave0 PADDR[2]
block1 0 APBmslave0 PADDR[1]
         block1 0 APBmslave0 PADDR[0] block1 0 APBmslave0 PWDATA[31]
block1_0_APBmslave0_PWDATA[30]
_ block1_0_APBmslave0_PWDATA[29] block1_0_APBmslave0_PWDATA[28]
block1_0_APBmslave0_PWDATA[27]
         block1 0 APBmslave0 PWDATA[26] block1 0 APBmslave0 PWDATA[25]
block1_0_APBmslave0_PWDATA[24]
block1_0_APBmslave0_PWDATA[23] block1_0_APBmslave0_PWDATA[22]
block1 0 APBmslave0 PWDATA[21]
         block1 0 APBmslave0 PWDATA[20] block1 0 APBmslave0 PWDATA[19]
block1 0 APBmslave0 PWDATA[18]
```

```
block1 0 APBmslave0 PWDATA[17] block1 0 APBmslave0 PWDATA[16]
block1 0 APBmslave0 PWDATA[15]
         block1 0 APBmslave0 PWDATA[14] block1 0 APBmslave0 PWDATA[13]
block1 0 APBmslave0 PWDATA[12]
         block1 0 APBmslave0 PWDATA[11] block1 0 APBmslave0 PWDATA[10]
block1 0 APBmslave0 PWDATA[9]
         block1 0 APBmslave0 PWDATA[8] block1 0 APBmslave0 PWDATA[7]
block1_0_APBmslave0_PWDATA[6]
block1_0_APBmslave0_PWDATA[5] block1_0_APBmslave0_PWDATA[4]
block1_0_APBmslave0_PWDATA[3]
         block1 0 APBmslave0 PWDATA[2] block1 0 APBmslave0 PWDATA[1]
block1 0 APBmslave0 PWDATA[\overline{0}]
         block1 0 APBmslave0 PENABLE block1 0 APBmslave0 PSELx block1 0 APBmslave0 PWRITE
IRS block2_0 p\overline{f}_smip_0 =
IRS pf smip 0 block2 0 =
IRS block3 0 block1 0 = block3 0 dataOut 0[7] block3 0 dataOut 0[6] block3 0 dataOut 0[5]
         block3 0 dataOut 0[4] block3 0 dataOut 0[3] block3 0 dataOut 0[2]
block3 0 dataOut 0[1]
         block3 0 dataOut 0[0]
IRS block1 0 block3 0 = block1 0 dataOut[7] block1 0 dataOut[6] block1 0 dataOut[5]
         block1 0 dataOut[4] block1 0 dataOut[3] block1 0 dataOut[2] block1 0 dataOut[1]
block1_0_dataOut[0]
IRS block3_0 pf_smip_0 =
IRS pf_smip_0 block3_0 =
IRS block1_0 pf_smip_0 =
IRS pf_smip_0 block1_0 =
REGIONS VERBOSITY = \overline{0}
```

- 1. 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 and the names of each IRS signal, and limit the maximum number of violations to be reported. For more descriptions about each parameter, see the *MSVT User Guide*.
- ^{2.} To verify the design using MSVT for SmartFusion2 and IGLOO2 devices, issue the following command:

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

To verify the design using MSVT for PolarFire devices, issue the following command:

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

A comprehensive report is printed into the filename given with the -o argument or to stdout if -o is omitted. On successful completion of this command, the message "MSVT Check failed" indicates that the design failed to meet one or more of separation criteria and the message "MSVT Check succeeded" indicates that the design met all separation criteria.

1.15 Further Considerations and Adjustments

- It might be convenient to have the chip-level resources related to the global network at the top-level design; particularly, if they are connected to multiple blocks.
- Certain PolarFire XCVR ERM related cells are automatically inserted or duplicated in the pre-placer that circumvent the floorplanning in the PDC. These instances do not appear in any of the user blocks and cannot be constrained by your region constraints.
- See the following table of coordinate that spans per device to floorplan any of the indicated instances. Overlapping spans must belong to the same block.

Cell	MPF100T		MPF200T		MPF300T		MPF500T	
	Min-span X	Min-span Y	Min-span X	Min-span Y	Min-span X	Min-span Y	Min-span X	Min-span Y
DRI	240 408	01	240 408	01	384 552	01	384 552	01
APBM	240 408	01	240 408	01	384 552	01	384 552	01

Table 1-1. Coordinates that Span Per Device to Floorplan any of the Indicated Instances

Libero[®] SoC v2021.2 Design Methodology

cont	inued							
Cell	MPF100T		MPF200T		MPF300T		MPF500T	
	Min-span X	Min-span Y	Min-span X	Min-span Y	Min-span X	Min-span Y	Min-span X	Min-span Y
SCB	240 363	02	240 363	02	384 507	02	384 507	02
ENFORC E	252 362	01	252 362	01	396 506	0 1	396 506	01
DEBUG	252 396	02	252 396	02	396 540	02	396 540	02
TVS	240 371	02	240 371	02	384 515	02	384 515	02
OSC_RC 200MHZ	240 368	02	240 368	02	384 512	02	384 512	02
PF_SPI	240 408	02	240 408	02	384 552	02	384 552	02
SC_STAT US	252 366	02	252 366	02	396 510	02	396 510	02
UJTAG_S EC	240 360	02	240 360	02	384 504	02	384 504	02
SYS_SE RVICES	240 408	02	240 408	02	384 552	02	384 552	02
VOLTAG EDETEC T	240 363	01	240 363	01	384 507	01	384 507	01
OSC_RC 2MHZ	240 367	02	240 367	02	384 511	02	384 511	02
INIT	240 364	02	240 364	02	384 508	02	384 508	02
TAMPER	288 408	02	288 408	02	432 552	02	432 552	02
PCIE	1572 1596	92 153	1572 1596	180 234	2436 2460	180 234	2724 2748	261 315
PCIE	1572 1596	159 201	1572 1597	227 282	2436 2460	240 282	2724 2748	321 363
XCVR_PI PE_AXI1	1572 1596	159 182	1572 1598	236 252	2436 2462	236 252	2724 2748	321 344
XCVR_PI PE_AXI0	1572 1596	111 155	1572 1597	192 236	2436 2461	192 236	2724 2748	273 317
XCVR_PI PE_AXI0	1572 1597	159 189	1572 1599	236 270	2436 2463	236 270	2724 2749	321 351
XCVR_PI PE_AXI1	1572 1596	128 147	1572 1596	210 236	2436 2460	210 236	2724 2748	290 309

2. Example

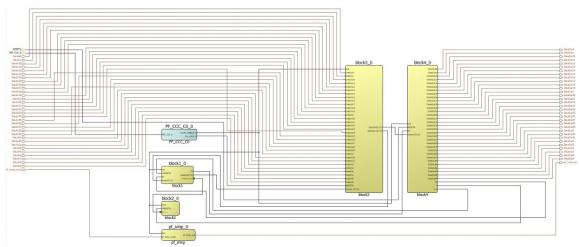
The following topics describe how to implement a complete PolarFire design using Microchip Design Separation methodology.

The design consists of six subsystems defined in Verilog:

- block1.v
- block2.v
- block3.v
- block4.v
- pf_smip.v
- PF CCC C0.v

The following figure shows a top-level view of these subsystems with interconnects between them.

Figure 2-1. Top-Level View of Example Design



This design is implemented using the design separation methodology steps defined in 1.2 Design Separation Methodology Steps.

Note: To understand the design flow and floorplanning terminology in the following topics, see the Libero SoC Design Flow and Chip Planner help topics in Libero.

2.1 Creating HDL Subsystems

The first step when implementing a complete system using Microchip Design Separation methodology is to achieve logical separation of various subsystems. Create logically separate HDL modules corresponding to the system.

This example defines the following subsystems:

- block1.v
- block2.v
- block3.v
- block4.v
- pf smip.v
- PF CCC C0.v

These subsystems are independent of each other. They communicate with each other using the interconnect signals. To begin with creating the HDL subsystem, follow these steps:

1. After you identify 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, the design using a MPF300TS, 484 FCVG device is implemented. Import the HDL files using File > Import > HDL Source file menu into the Libero Project.
- 3. Create a Block for each subsystem of this design.

2.2 Creating Blocks

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

- 1. Select a module to be the root module. For example, select **block4** as shown in the following figure.
 - Figure 2-2. Selecting a Module as the Root of a Block

lesign Hierarchy		8,
Top Module(root): block3		
Build Hierarchy	🔍 Show: Components 💌 🛨 📑 📍	8
	32IMC_ram_singleport_addreg (opsrv_merged.v) [work 32IMC_opsrv_dummy_ram (opsrv_merged.v) [work] 32IMC_common_buffer_behav (submicron_merged.v) [. 32IMC_axi_xaddr_buffer (opsrv_merged.v) [work] 32IMC_axi_egress_buffer (opsrv_merged.v) [work] RTapb_C1 (CoreUARTapb_v5.6.102)	
		Н
So block2 So block1 So block2 So block1 So block2 So block1 So block2 So block1 So block2 So block2 So block1 So block2 So block1 So block2 So block1	Export Component Description(ICI) Hierarchical Export Component Description(TcI) Rename Component Open HDL File Check HDL File Create I/O Constraint from Module	
COREAPB3_I COREA	Delete	
Diser HDL Ser HDL	Course File Doth	
	Show Module Parameters Export Parameter Report	
Design Flow Design Hier	Properties	
og	Show Module	

2. Use **Project > Project Settings > Design Flow > Enable Block Creation** to enable Block flow for this module, as shown in the following figure.

		-	
vice selection			
evice settings	HDL source files language options		Save
sign flow alysis operating conditions nulation options	syntax option if your Verilog files contain System Verilog o	n import Verilog and VHDL in the same project. For Verilog files you can enable the System Verilog onstructs. For VHDL files, you may choose between VHDL-2008 and VHDL-93.	Discar
- DO file - Waveforms - Vsim commands	and macros defined in one file will be seen by other files of When disabling that option, each file is compiled as a sepa	ne neabling the Multi-File Compilation Unit (MFCU) option, all files are compiled in the same compilation unit, ompiled later based on the file order. rate compilation unit.	
Timescale	Verilog		
nulation libraries - PolarFire - COREAHBLITE LIB	C Verilog 2001	🤄 System Verlog 🦳 Multi Fle Compliation Unit	
COREAPB3 LIB	VHDL		
obal Include Paths	@ VHDL-2008	C VHDL-93	
	HDL generated files language options		
	HDL files generated by Libero SoC such as configured cores, SmartDesign	n components and post-layout gate level netists use the preferred language option. C VHDL	
	Block flow		
	Block flow enables you to publish a reusable component that can be insta It could include timing constraints, physical constraints, placement or rou F Enable block creation	initiated into another design. A block component may not contain I/O cells and cannot be programmed by itself. Imp.	
	It could include timing constraints, physical constraints, placement or rou	initiated into another design. A block component may not contain I/O cells and cannot be programmed by itself. Ing.	
	It could include timing constraints, physical constraints, placement or rou Finable block creation Root block4	nitated into another design. A block component may not contain I/O cels and cannot be programmed by itself. Ing.	
	It could include timing constraints, physical constraints, placement or rou	nitated into another design. A block component may not contain I/O cells and cannot be programmed by itself.	
	It could include timing constraints, physical constraints, placement or rou P Enable block creation Root block4 P Enable synthesis	Intelated into another design. A block component may not contan I/O cells and cannot be programmed by itself.	
	It could reduce timing constraints, physical constraints, placement or rou Image: Second constraints, placement or rou Root block- Image: Second constraints, placement or rou Image:	Instated into another design. A block component may not contain I/O cells and cannot be programmed by itself.	
	It could include timing constraints, physical constraints, placement or rou	ntatated into another design. A block component may not contan I/O cells and cannot be programmed by itself.	
	It could include timing constraints, physical constraints, placement or rou I' Enable block creation Root block I' Enable synthesis I' Enable PPGA Hardware Breakpoint Auto Instantiation Design Separation Enable Design Separation Methodology	ntated into another design. A block component may not contan I/O cells and cannot be programmed by itself.	
	It could not due temp constraints, physical constraints, placement or rou I' Enable block creation Root block- I' Enable synthesis I' Enable PPGA Hardware Breakpoint Auto Instantiation Design Separation Enable Design Separation Methodology Reports	ntatated into another design. A block component may not contan 1/0 cells and cannot be programmed by itself.	
	It could include timing carstraints, physical constraints, placement or rou I' Enable block creation Root block4 I' Enable synthesis I' Enable PPGA Hardware Breakpoint Auto Instantiation Design Separation Enable Design Separation Methodology Reports Maximum number of high fanout nets to be displayed: 10	ntated into another design. A block component may not contan I/O cells and cannot be programmed by itself.	

Figure 2-3. Project Settings for Block Creation

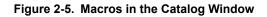
The **Publish Block** option is enabled in the design flow, as shown in the following figure.

gn Flow			
o Module(root): block4		0 4	
tive Synthesis Implementation: synthesis			
Tool			
Create SmartDesign Testbench			
Create HDL Testbench			
Verify Pre-Synthesized Design			
Simulate			
🖻 🕨 Constraints			
Manage Constraints			
🖻 🕨 Implement Design			
🖓 Open Netlist Viewer			
Synthesize			
Verify Post-Synthesized Design	n		
Generate Simulation File			
Simulate			
Place and Route			
Edit Post Layout Design			
Verify Post Layout Implement			
 Generate Back Annotated Fi 	les		
Simulate			
🖳 💁 Verify Timing			
🗠 💁 Open SmartTime			
🗠 🖹 Verify Power			
🗁 🛱 Open SSN Analyzer			
Program Design			
Configure Design Initialization		emories	
• Generate Design Initialization D	ata		
🖻 🕨 Publish Design			
Publish Block			

3. If a Block requires physical I/O resources, specify explicit instantiation of I/O resources. All logic within the Microchip Design Separation methodology must be incorporated within an isolated region. Therefore, you must associate all physical I/O resources with an isolated region. You can insert I/Os through direct instantiation or through insertion of I/O buffers using SmartDesign.

This example uses the Catalog in SmartDesign to insert I/Os to top-level signals. **block4** subsystem has following port list:

- CLK , RESETN , RX, dataIn: Top-level input signals
- DataOut: Top-level output signals
- TX, Y_0: IRS signals to **block1**
- Y: IRS signals to **block2**
- 4. Because each Block should have I/Os inserted for its top-level I/O signals, insert I/O ports to the top-level signals of this subsystem.
- 5. Create a SmartDesign with the name **block4**. Instantiate the COREAPB3, COREGPIO, and MIV_RV32IMC components into **block4**.
- 6. For each top-level output signal, assign an OUTBUF macro. This instantiates a single I/O port for each of the signals. The output signal DataOut has a width of 32 bits.



	🔍 👻 🦵 Simulation Mode	0
Name	/ Version	-
GCLKBUF_DIFF	1.0	
GCLKINT	1.0	
INBUF	1.0	
INBUF_DIFF	1.0	
INV	1.0	
INVD	1.0	
LIVE_PROBE_A	1.0	
LIVE_PROBE_B	1.0	
MACC_PA	1.0	
MACC_PA_BC_ROM	1.0	
MX2	1.0	
MX4	1.0	
NAND2	1.0	
NAND3	1.0	
MAND4	1.0	
-NOR2	1.0	
-NOR3	1.0	-
NOR4	1.0	
-OR2	1.0	
OR3	1.0	
OR4	1.0	
OSC_RC160MHZ	1.0	
OSC_RC2MHZ	1.0	
OUTBUF	1.0	
OUTBUF_DIFF	1.0	-
New cores are availal		

- 7. After you instantiate all required I/O macros, rename them to a unique name and connect these I/O pads to respective ports of **block4** instance.
- 8. Since DataOut, TX, Y, and Y_0 are interconnected signals, right-click the ports and promote them to the top. The following figure shows the schematics of the **block4** SmartDesign component.

Libero[®] SoC v2021.2 Example

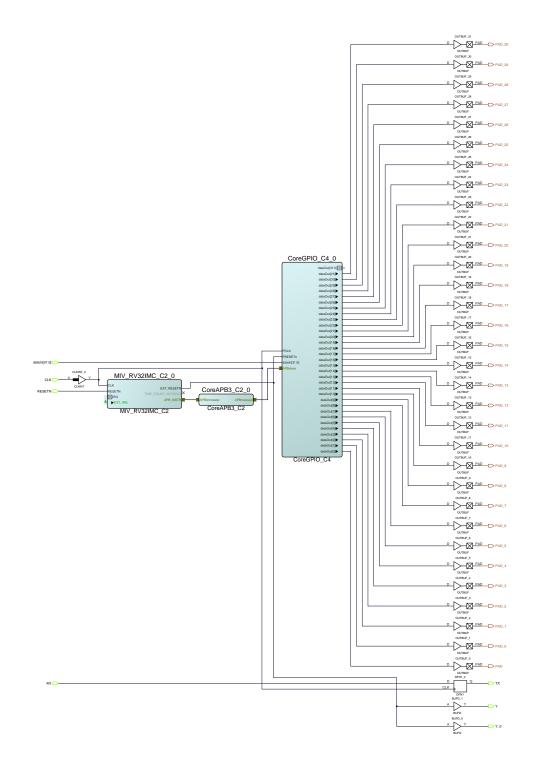


Figure 2-6. SmartDesign Component for block4 Subsystem along with I/Os

- 9. Generate **block4**, set this module as the root module, and enable Block creation for this module as described in the previous section.
- 10. Publish the Block.

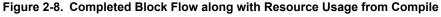
2.3 Publishing the Block

After you create the module with I/Os inserted, publish the Block. Run Synthesis, and compile the subsystem (or its wrapper SmartDeisgnor HDL component). You can check for timing closure on the Block. Publish the Block without place-and-route information.

- 1. For **block4**, run Synthesis.
- Disable Publish Placement and Publish Routing information in Publish Block > Configure Options, as shown in the following figure. Publish the block.
 Note: 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.
- ^{3.} Figure 2-7. Publishing the Block without Placement and Routing Information

Folder:	990\Des	ktop\msvt\design1\desig	ner\block4 Bro	wse.
Publish Block	Configuration —			
Publish	Placement			
Publish	Routing			
Publish	Region			
Language —				
Verilog		C VHDL		

The following figure shows the state of the completed design flow after you publish the Block.



odule(root): block4 🔄 🖸 🖬 📄 🜮	Project Summary Dock4 reports	Resource Usag	ge		
Tool	Components Synthesize	Туре	Used	Total	Percentage
Create SmartDesign Testbench	synplify.log	4LUT	6303	299544	2.10
Create HDL Testbench	block4.srr	DFF	2121	299544	0.71
Verify Pre-Synthesized Design	run_options.txt	I/O Register		1536	0.00
Simulate	block4_dsp_rpt.txt	User I/O		512	6.25
Constraints	block4_ram_rpt.txt				
Manage Constraints	 block4_compile_netlist_resources.xml 	Single-ended I/O	32	512	6.25
Implement Design	block4_compile_netlist_hier_resources.csv	- Differential I/O Pairs	0	256	0.00
- 🖓 Open Netlist Viewer	block4_compile_ioff.xml	uSRAM	6	2772	0.22
Synthesize	I block4_compile_netlist.log	LSRAM	16	952	1.68
Verify Post-Synthesized Design	Non root components	Math		924	0.00
Generate Simulation File Simulate	block1		-		
Place and Route	block2 block3	H-Chip Global		48	2.08
Edit Post Layout Design	CoreAHBLite C0	PLL	0	8	0.00
Verify Post Layout Implementation	CoreAPB3 C0	DLL	0	8	0.00
Generate Back Annotated Files	CoreAPB3_C0 CoreAPB3_C1	Transceiver Lanes	0	16	0.00
- Simulate	CoreGPIO CO	Transceiver PCIe	0	2	0.00
- 💩 Verify Timing	CoreGPIO C1			-	
💩 Open SmartTime	CoreGPIO_C2				
- 🕼 Verify Power	I CoreGPIO_C3	B. f. H. J. Lawle	-		
🖓 Open SSN Analyzer	CoreUARTapb_C0	Detailed Logic	Reso	ource	Usage
Program Design	CoreUARTapb_C1	-		DEE	
-• Configure Design Initialization Data and Memories	MIV_RV32IMC_C0	Туре	4LUT		
Generate Design Initialization Data	MIV_RV32IMC_C1	Fabric Logic	5655	1473	
Publish Design	Image: Berline Ber	uSRAM Interface Logic	72	72	
Publish Block		LSRAM Interface Logic	576	576	
		Math Interface Logic	0	0	

Libero[®] SoC v2021.2 Example

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

For the preceding subsystem, Libero creates the block4.cxz file in the export directory under the designer directory of the Project location, as shown in the following figure.

iles		8 >
🕀 🗋 co	mponent	•
🕀 🗀 co	nstraint	
🖶 🗀 de	signer	
÷	block1	
	block2	
	j block3	
	block4	
	block4.adl	
	🖻 block4.afl	
	block4.cfrt	
	block4.cxz	
	block4.loc	
	block4.mvn.pdc	
	block4.nmatinit.pdc	
	block4.nmatinit.txt	
	block4.seg	
	block4.smat.seg	
	block4_block_region_violation	
	block4_compile_ioff.rpt	
	block4_compile_ioff.xml	
	block4_compile_netlist.log	
	block4_compile_netlist_hier_resources.csv	
	block4_compile_netlist_resources.rpt	
	block4_compile_netlist_resources.xml	
	block4_core.tcml	
	block4_glb_net_report.html	
	block4_glb_net_report.xml	
	block4_has_io_constraints	
	block4_has_violations	
	block4_ioff.rpt	
	block4_maxdelay_timing_report.xml	•
Design Flow	Design Hierarchy Stimulus Hierarchy Catalog Files	

Figure 2-9. Published Block as .cxz file in Export Directory

4. Repeat this procedure for the other four subsystems, and then publish Blocks for each of them. Use the Block names shown in the following table.

Subsystem	Block Name
block1	block1
block2	block2
block3	block3
pf_smip	pf_smip

5. Create a top-level design.

2.4 Creating a Top-level Design

After you published the subsystem Blocks, create a new Libero project for the top-level design. Create a SmartDesign block where you instantiate all the individual blocks and connect their IRS signals. This example writes a top-level module SD_Top.v that instantiates these Blocks along with required interconnects. The following shows an example description of the top-level SD Top module.

Libero[®] SoC v2021.2

Example

///////////////////////////////////////	
`timescale	e 1ns / 100ps
// top	
module top	
// Ing	
	DataIn0,
	DataIn1, DataIn10,
	DataIn11,
-	DataIn12,
	DataIn13,
input	DataIn14,
	DataIn15,
-	DataIn16,
	DataIn17, DataIn18,
	DataIn19,
-	DataIn2,
	DataIn20,
input	DataIn21,
	DataIn22,
-	DataIn23,
	DataIn24, DataIn25,
	DataIn26,
	DataIn27,
input	DataIn28,
	DataIn29,
-	DataIn3,
	DataIn30, DataIn31,
_	DataIn4,
	DataIn5,
_	DataIn6,
	DataIn7,
	DataIn8, DataIn9,
	REF CLK 0,
	RESETN,
	pf_smip_reset,
// Out	-
_	: DataOutO, : DataOut1,
_	DataOut10,
output	DataOut11,
_	DataOut12,
	DataOut13,
	: DataOut14, : DataOut15,
	DataOut16,
-	DataOut17,
output	: DataOut18,
	: DataOut19,
-	DataOut2,
	: DataOut20, : DataOut21,
-	DataOut22,
	DataOut23,
output	DataOut24,
_	DataOut25,
-	DataOut26,
_	: DataOut27, : DataOut28,
-	DataOut29,
_	DataOut3,
-	DataOut30,
_	DataOut31,
_	: DataOut4, : DataOut5,
	DataOut5,
_	DataOut7,
_	DataOut8,
	DataOut9,
output	pf_smip_out

Libero[®] SoC v2021.2 Example

);
//
// Nets
<pre>// wire [31:0] block1 0 APBmslave0 PADDR;</pre>
wire block1_0_APBmslave0_PENABLE;
<pre>wire [31:0] block1_0_APBmslave0_PRDATA; wire block1_0_APBmslave0_PREADY;</pre>
wire block1_0_APBmslave0_PSELx;
<pre>wire block1_0_APBmslave0_PSLVERR; wire [31:0] block1_0_APBmslave0_PWDATA;</pre>
<pre>wire block1_0_APBmslave0_PWRITE; wire [7:0] block1 0 dataOut;</pre>
wire block1_0_TX;
<pre>wire [31:0] block3_0_dataOut; wire [7:0] block3_0_dataOut 0;</pre>
wire block4_0_TX;
wire block4_0_Y; wire block4_0_Y 0;
wire REF_CLK_ibuf_Y; wire RESETN ibuf Y;
wire PF_CCC_C0_OUTO_FABCLK_0;
//
// Component instances
//block1
block1 block1_0(// Inputs
.CLK (PF_CCC_C0_0_OUT0_FABCLK_0),
.HRESETN (block4_0_Y_0), .PREADYS0 (block1_0 APBmslave0 PREADY),
.PSLVERRS0 (block1_0_APBmslave0_PSLVERR),
.RX (block4_0_TX), .PRDATASO (block1 0 APBmslave0 PRDATA),
.dataIn (block3_0_dataOut_0), // Outputs
.PENABLES (block1_0_APBmslave0_PENABLE),
.PSELS0 (block1_0_APBmslave0_PSELx), .PWRITES (block1_0_APBmslave0_PWRITE),
.TX (block1_0_TX),
.PADDRS (block1_0_APBmslave0_PADDR), .PWDATAS (block1_0_APBmslave0_PWDATA),
.dataOut (block1_0_dataOut)
);
//block2 block2 block2 0(
// Inputs
.PCLK (PF_CCC_C0_0_OUT0_FABCLK_0), .PENABLE in (block1 0 APBmslave0 PENABLE),
.PRESETN (block4_0_Y),
.PSEL_in (block1_0_APBmslave0_PSELx), .PWRITE in (block1 0 APBmslave0 PWRITE),
.PADDR in (block1 0 APBmslave0 PADDR), .PWDATA in (block1 0 APBmslave0 PWDATA),
// Outputs
.PREADY_in (block1_0_APBmslave0_PREADY), .PSVERR_in (block1_0_APBmslave0_PSLVERR),
.PRDATA_in (block1_0_APBmslave0_PRDATA)
);
//block3 block3 block3 0(
// Inputs
.CLK (PF_CCC_C0_0_OUT0_FABCLK_0), .DataIn0 (DataIn0),
.DataIn1 (DataIn1),
.DataIn2 (DataIn2), .DataIn3 (DataIn3),
.DataIn4 (DataIn4), .DataIn5 (DataIn5),
.DataIn6 (DataIn6),
.DataIn7 (DataIn7), .DataIn8 (DataIn8),
· Jacarno (Jacarno /)

Libero[®] SoC v2021.2

Example

.DataIn9	(DataIn9),
.DataIn10 .DataIn11	(DataIn10), (DataIn11),
.DataIn11 .DataIn12	(Datainii), (DataIn12),
.DataIn13	(DataIn13),
.DataIn14	(DataIn14),
.DataIn15	(DataIn15),
.DataIn16	(DataIn16),
.DataIn17	(DataIn17),
.DataIn18	(DataIn18),
.DataIn19	(DataIn19),
.DataIn20 .DataIn21	(DataIn20), (DataIn21),
.DataIn22	(DataIn22),
.DataIn23	(DataIn23),
.DataIn24	(DataIn24),
.DataIn25	(DataIn25),
.DataIn26	(DataIn26),
.DataIn27	(DataIn27),
.DataIn28 .DataIn29	(DataIn28), (DataIn29),
.DataIn30	(DataIn30),
.DataIn31	(DataIn31),
.RESETN	(RESETN ibuf Y),
.dataIn_0	(block1_0_dataOut),
// Outputs	
.dataOut	
—	(block3_0_dataOut_0)
);	
//block4	
block4 block4 0(
// Inputs	
.CLK	(PF_CCC_C0_0_OUT0_FABCLK_0),
.RESETN	(RESETN_ibuf_Y),
.RX .dataIn	(block1_0_TX), (block3 0 dataOut),
// Outputs	(DIOCKS_0_dataout),
.DataOut0	(DataOut0),
.DataOut1	(DataOut1),
.DataOut2	(DataOut2),
.DataOut3	(DataOut3),
.DataOut4	(DataOut4),
.DataOut5 .DataOut6	(DataOut5), (DataOut6),
.DataOut7	(DataOut7),
.DataOut8	(DataOut8),
.DataOut9	(DataOut9),
.DataOut10	(DataOut10),
.DataOut11	(DataOut11),
.DataOut12	(DataOut12),
.DataOut13 .DataOut14	(DataOut13), (DataOut14),
.DataOut14	(DataOut14), (DataOut15),
.DataOut16	(DataOut16),
.DataOut17	(DataOut17),
.DataOut18	(DataOut18),
.DataOut19	(DataOut19),
.DataOut20	(DataOut20),
.DataOut21	(DataOut21), (DataOut22),
.DataOut22 .DataOut23	(DataOut22), (DataOut23),
.DataOut23	(DataOut23),
.DataOut25	(DataOut25),
.DataOut26	(DataOut26),
.DataOut27	(DataOut27),
.DataOut28	(DataOut28),
.DataOut29	(DataOut29),
.DataOut30	(DataOut30),
.DataOut31 .TX	(DataOut31), (block4 0 TX),
. Y	(block4_0_Y),
.Y_0	(block4_0_Y_0)
);	
//INBUF	
//INDUE	

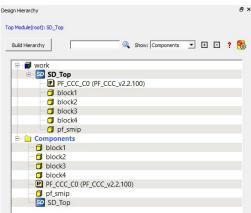
Libero[®] SoC v2021.2 Example

```
INBUF REF CLK ibuf(
        /7 Inputs
        .PAD ( REF CLK 0 ),
        // Outputs
             ( REF_CLK_ibuf_Y )
        .Υ
        );
//----INBUF
INBUF RESETN_ibuf(
        // Inputs
        .PAD ( RESETN ),
        // Outputs
        .Y (RESETN ibuf Y)
        );
//----PF_CCC_C0
PF_CCC_C0 PF_CCC_C0_0(
        // Inputs
        .REF_CLK 0
                        ( REF CLK ibuf Y ),
        // Outputs
        .OUTO FABCLK 0 ( PF CCC CO 0 OUTO FABCLK 0 ),
        .PLL LOCK 0
                       ()
        ):
//----pf smip
pf smip pf smip 0(
       // Inputs
.CLK
                        ( PF CCC C0 0 OUTO FABCLK 0 ),
        .pf_smip_reset ( pf_smip_reset ),
        // Outputs
        .pf_smip_out
                        ( pf_smip_out )
        );
```

endmodule

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

Figure 2-10. Top-level Design Hierarchy



2. For this module, uncheck the boxes for **Enable block creation** and **Enable synthesis** (in **Project Settings**) and check **Enable Design Separation Methodology**, as shown in the following figure.

Figure 2-11. Project Settings for the Top-level Design

	HDL source files language options	Save
Device settings Design flow nalysis operating conditi imulation options DO file	Libero SoC supports mixed-HDL language designs; you can import Verilog and VHDL in the same project. For Verilog files you can enable the System Verilog syntax option if your Verilog files contain System Verilog constructs. For VHDL files, you may choose between VHDL-2008 and VHDL-93.	Discard
Waveforms Vsim commands Timescale	Verlog © System Verlog © Verlog 2001]
mulation libraries PolarFire eneral Settings lobal Include Paths	VHDL	
	HDL generated files language options	
	C VHDL Block flow	
	Block flow enables you to publish a reusable component that can be instantiated into another design. A block component may not contain I/O cells and cannot be programmed by itself. It could include timing constraints, physical constraints, placement or routing.	
	Root SD_Top	
	Enable FPGA Hardware Breakpoint Auto Instantiation Design Separation	
	P Enable Design Separation Methodology	
	Reports Maximum number of high fanout nets to be displayed: 10	-
	Abort flow if errors are found in Physical Design Constraints (PDC) ✓ Abort flow if errors are found in Timing Constraints (PDC)	

3. Assign each Block of the design to a separation region.

2.5 Floorplanning Design with Separation Regions

Each Block of the design must be assigned to a separation region. You can define separation regions in Chip Planner or use a PDC file. For more information about floorplanning a design using Chip Planner, see the Libero online help.

This example defines a separation region using Chip Planner for instance block4_0, which corresponds to the block4 Block.

1. Open Chip Planner from **Manage Constraints > Floor Planner > Edit** in the **Design Flow** window, as shown in the following figure.

Figure 2-12. Opening Chip Planner

Design Flow		6 ×	SD_Top # ×	Constraint M	nager e ×	StartPage # ×				
Top Module(root): SD_Top	D Id	9	1/0 Attributes	Timing Floor Pl	mer Netist	Attributes				
Active Synthesis Implementation: synthesis			New	Import	Unk	Edt	Vev	Check	Help	1
Tool		-		in port		Modify the se	t of selecter			ng the Chip Planner Edito
B Create Design		_				Place and Rou	te.			
Create SmartDesign				lop\SD_Top.bl	ck.fp.pdc					
Create HDL			constraint\fp\	top.pdc		M				
Create SmartDesign Testbench										
Create HDL Testbench										
B Verify Pre-Synthesized Design										
Simulate										
B- > Constraints										
Manage Constraints										
V B > Implement Design										
- B Open Netlist Viewer										
V Synthesize										
B- Verify Post-Synthesized Design										
Generate Simulation File										
Simulate										
V Place and Route										
Edit Post Layout Design										
B Verify Post Layout Implementation										
 Generate Back Annotated Files 										
- Simulate										
- 🔍 Verify Timing										
Open SmartTime										
R Verify Power										
- to Open SSN Analyzer										
8- Configure Hardware										
Interface										
- & Configure Programmer		*								

Create a separation region for each Block according to the estimate obtained from the resource usage reports.
 Note: For more information, see Creating routing regions in the Chip Planner help.

1.7 FloorPlanning with Design Separation Regions shows a sample floorplan for the design. This example creates an exclusive routing region constraint for instance block4_0, as shown in the following figure. (You can define the region type to be inclusive if a top-level global instance needs to be placed within the same region).

Properties				×	
Region	ame: Bookkregion				
0 0 C2_0	Snckalve Exclusive Exclusive Enpty				
Routing	orebain routing				IBR1_2 Block2region
	Reg	ion Extents		1region	Block2region
	rigin endpt	width	height		
1 (1200.		1152 sources	159	IBR1_4	IBR2_4
	esource Assigne	d Capacity	% Used		
	CARRY_CHAIN 6241	59280	10.53		akáragian
CARRY	CHAIN 644	38760	1.66	IBR3 III C. I C	
CC_CO		38760	0.17	ion in the second s	AIN IT SUUL
DFF	2120	59280 33	3.58		Received and the second s
MATH	0	192	0.00		
SRAM	16	192	8.33		
URAM	6	570	1.05	7	

- Create separation regions for all blocks of the design. Then assign Block instances to the respective
- Create separation regions for all blocks of the design. Then assign Block instances to the respective separation region. For this example, the floorplan resembles the following figure.

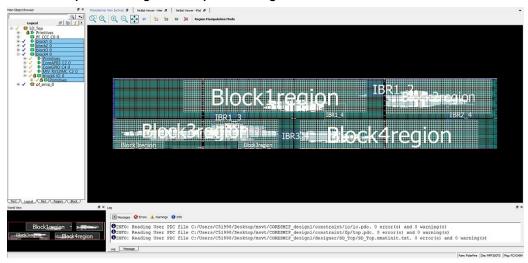


Figure 2-14. Floorplan of Design with Separation Regions Defined

4. After you create separation regions corresponding to the Blocks, create IRS regions for each set of connections between the Blocks. In the example, block1_0 connects to IBR1_2 and IBR1_2 connects to block2_0. Consequently, define the remaining three sets of IRS regions.

IRS region is an inclusive routing region that is created in a similar way as the separation regions. The following figure show an example of an IRS region for instances block1_0 to IBR1_2.

Libero[®] SoC v2021.2 Example

turag [™] Createan multing for Createan	<u>IBR1</u> 2 Block2region
turag [™] Createan multing for Createan	<u>IBR1</u> 2 Block2region
turag [™] Createan multing for Createan	Block2region
turag [™] Createan multing for Createan	DIOCKZTEGION
104/07 [™] Cremteinnutry 0 cremteinnutry 1544-2821 2027-3522 444 81 1544-2824 81 Resource Assigned Capacity % Used 1016 - CARPC 444 309	
Region Extents origin endpt width height 1584-2820 2027 3620 81 Resources Resigned Capacity % Used UUT - CARPC HAN 300 11664 3.09	
origin endpt width height (1584.282) (2027.362) 444 81 Resources Resource Assigned Capacity % Used UP: CARPC (HAN 360 11664 3.09	
Resource Add B1 Resource Assigned Capacity % Used U1 + CARPC HANN 360 11664 3.09	
Resource Image: Comparison of the comparison	
Resource Assigned Capacity % Used IUT + CARRY CHAIN 360 11664 3.09	
LUT + CARRY CHAIN 360 11664 3.09	
ARRY_CHAIN 0 7776 0.00	
CC_CONFIG 0 7776 0.00	
DFF 0 11664 0.00	
AATH 0 36 0.00	
RAM 0 36 0.00	
RAM 0 36 0.00	
RAM 0 36 0.00	
RAM 0 36 0.00	
RAM 0 36 0.00	

Figure 2-15. Creating an IRS Region Between Two Blocks

Assign valid IRS net macros to the respective IRS regions.
 A complete floorplan of the example design resembles the following figure.

Figure 2-16. Complete Floorplan of Design



The separation between each region should be at least equal to the required number of clusters to satisfy the separation criteria.

The following shows a sample PDC file that can be used to implement the preceding floorplan. Note the regions are defined with -route true to constrain routing. Separation regions are assigned by their highest level hierarchy name using the assign_region command. IRS nets are assigned with wildcards using the assign net macros command.

```
define region -region name Block1region -type exclusive -color 2143338688 -route true -
push place true -x1 456 -y1 195 -x2 1631 -y2 371
define region -region name Block2region -type exclusive -color 2143338688 -route true -
push_place true -x1 1752 -y1 189 -x2 2435 -y2 377
define_region -region_name Block3region -type exclusive -color 2143338688 -route true -
                          0 -y1
                                  0 -x2 335 -y2 41 ∖
push place true -x1
                  -x1
                          0 -y1 42 -x2 1067 -y2 161 \
                  -x1 804 -y1 0 -x2 1067 -y2 41
define region -region name Block4region -type exclusive -color 2143338688 -route true -
push place true -x1 1200 -y1
                                   0 -x2 2351 -y2 158
define_region -region_name SMIPregion -type exclusive -color 2143338688 -route
push_place true -x1 384 -y1 0 -x2 755 -y2 11
define_region -region_name IBR1_2 -type inclusive -color 2147442270 -route true -
                                             -type exclusive -color 2143338688 -route true -
push place false -x1 1584 -y1 282 -x2 2027 -y2 362
define region -region_name IBR1_3 -type inclusive -color 2147442270 -route true -
push_place false -x1 636 -y1 102 -x2 851 -y2 239
define_region -region_name IBR1_4 -type inclusive -color 2143338688 -route true -
push place false -x1 1356 -y1 126 -x2 1499 -y2 245
define region -region name IBR2 4 -type inclusive -color 2147442270 -route true -
```

Example

```
push place false -x1 2148 -y1 105 -x2 2327 -y2 266
define_region -region_name IBR3_4 -type inclusive -color 2147442270 -route true -
push_place false -x1 888 -y1 45 -x2 1463 -y2 98
assign_region -region_name Block1region -inst_name block1_0
assign_region -region_name Block2region -inst_name block2_0
assign region -region name Block3region -inst name block3 0
assign region -region name Block4region -inst name block40
assign_region -region_name Block4region -inst_name RESETN_ibuf
assign_region -region_name SMIPregion -inst_name pf_smip_0
assign_net_macros -region_name IBR1_2 -net_name block1_0_APBmslave0_PENABLE
include driver true
assign net macros -region name IBR1 2 -net name block1 0 APBmslave0 PSELx
include driver true
assign net macros -region name IBR1 2 -net name block1 0 APBmslave0 PWRITE
include driver true
assign net macros -region name IBR1 2 -net name block1 0 APBmslave0 PREADY
include driver true
assign net macros -region name IBR1 2 -net name {block1 0 APBmslave0 PADDR[*]}
include driver true
assign net macros -region name IBR1 2 -net name {block1 0 APBmslave0 PRDATA[*]} -
include driver true
assign net macros -region name IBR1 2 -net name {block1 0 APBmslave0 PWDATA[*]} -
include driver true
assign net macros -region name IBR1 3 -net name {block1 0 dataOut[*]} -include driver
true
assign net macros -region name IBR1 3 -net name {block3 0 dataOut 0[*]} -include driver
true
assign net macros -region name IBR1 4 -net name block4 0 TX -include driver true
assign_net_macros -region_name IBR1_4 -net_name block4_0_Y_0 -include_driver true assign_net_macros -region_name IBR1_4 -net_name block1_0_TX -include_driver true
assign net macros -region name IBR2 4 -net name block4 0 Y -include driver true
assign_net_macros -region_name IBR3_4 -net_name {block3_0_dataOut[*]} -include_driver
true
```

6. Complete place-and-route.

2.6 Complete Place-and-Route

After you complete the floorplan, edit the timing constraints and run place-and-route until you achieve timing closure on the design.

2.7 Configure Security Settings and Generate the Programming File

After you complete place-and-route, extract the design information to execute MSVT.

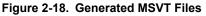
1. Navigate to **Configure Security > Configure Options > Debug Policy** and configure the security and programming options per system requirements, as shown in the following figure.

Libero[®] SoC v2021.2 Example

Figure 2-17. Security Settings Before Programming

	Debug Policy	
PK has not been provided and		
martDebug User debug access martDebug Live Probe debug a	and active probes are enabled. ccess is enabled.	
nartDebug sNVM debug is ena JTAG command through JTAG	oled.	
FAG (1149. 1) boundary scan is	enabled.	
FAG or SPI Slave reading of ter	mperature and voltage sensor is enabled.	
	Microsemi Factory Access	
llow Microsemi factory test mo	le access. This is required to perform Failure Analysis on the device.	
	JTAG/SPI Slave Commands Policy	
IE an inten through TTAC an	i CDT Chura ia suralisida	
ecurity key mode		
Bitstream encrypt	ion with default key Custom security options	
	Debugging is enabled by default at this stage for design and debug. Use FlashLock/UPK1 or DPK to temporarily enable access to disabled debug features during one debugging session.	
User Keys	de namedy i na v o n to conjectiny chale access to assere access retaines anny one accessing accessin	
	DPK (Debug Pass Key) (64 HEX chars):	
		- 0
'	Dx	ß
Update Policy	0x SmartDebug access control	ß
Update Policy	-SmartDebug access control	ß
Update Policy	SmartDebug access control	f
Update Policy	SmartDebug access control	f
	SmartDebug access control	
Update Policy Debug Policy	SmartDebug access control	
	Smart/Debug access control Disable user debug access and active probes Disable uive Probe Disable sNVM	
	Smithebug access control Smithebug access control Instable user debug access and active probes Instable user Arbote Disable #WM Disable UTAG command through JTAG interface	ß
Debug Policy Microsemi	Smart/Debug access control Disable user debug access and active probes Disable uive Probe Disable sNVM	ß
Debug Policy Microsemi Factory	Smithebug access control Smithebug access control Instable user debug access and active probes Instable user Arbote Disable #WM Disable UTAG command through JTAG interface	Ê
Debug Policy Microsemi	Smurtbebug access control	
Debug Policy Microsemi Factory	Smurtbebug access control	
Debug Policy Microsemi Factory	Smittlebug access control	
Debug Policy Policy Access TrAcfory Access TrAcfort Slave Commands	Smittlebug access control	
Debug Policy Microsemi Factory Access	Smittlebug access control	
Debug Policy Policy Access TrAcfory Access TrAcfort Slave Commands	Smittlebug access control	

2. Export the programming file from Program Design > Export FlashPro Express Job. A programming file and files required for the MSVT are generated. Libero exports these files into the <project_path>/designer/SD_Top/SD_Top.msvt.dtf directory and creates a parameter file in the <project_path>/designer/SD_Top/msvt.param file, as shown in the following figure. The msvt.param file contains a list of parameters that you can adjust before executing MSVT.



Clocklist.txt	
coverage_placeandroute	
🖳 🗀 export	
io_pcbit_info.ddf	
📑 🗎 msvt.param	
options.txt	
pinslacks.txt	
place_route.sdc	
🗎 rptstyle.xsl	
🖳 🗎 run_compile.tcl	
🖻 run_mapper.def	
🖻 run_pinrpt.tcl	
🗎 run_placer.def	
run_placer_st_shell_cmd.tcl	
run_placer_st_shell_des.tcl	
🖻 run_router.def	
🖻 run_tao.tcl	
🖻 run_tao_adl.tcl	
run_tao_adl_vt.tcl	
- 🗎 run_timrpt_st_shell_cmd.tcl	
run_timrpt_st_shell_des.tcl	
- 🗎 SD_Top.adl	
D_SD_Top.afl	
SD_Top.block.fp.pdc	
SD_Top.block.io.pdc	
- 🖹 SD_Top.cfrt	
D SD_Top.loc	
SD_Top.map SD_Top.msvt.dtf	

- esign Flow Design Hierarchy Stimulus Hierarchy Catalog Files
- 3. Modify the REQUIRED_SEPARATION parameter according to your system requirements before executing MSVT.

2.8 Execute MSVT

You can now run the MSVT to verify that the design adheres to the required separation criteria.

MSVT is invoked from <Libero Path>/bin64/msvt check pf. It is executed from the command line.

To verify the design using MSVT, run the following command:

```
<Libero_path>/bin64/msvt_check_pf -p <project_path>/designer/SD_Top/msvt.param -o msvt_check_pf.log
```

This command prints an exhaustive report to $msvt_check_pf.log$ file given with the -o argument or to *stdout* if -o is omitted. The argument -p is required, along with the path to the msvt.param file generated from Libero.

When this command completes successfully, the message MSVT Check failed appears if the design failed to meet one or more separation criteria and the message MSVT Check succeeded appears if the design met all separation criteria.

Because Microchip Design Separation methodology guidlines are followed in the example, the following output shows the conclusion of MSVT output indicating that the design was verified for the given separation criteria.

```
MSVT Check
Design: SD Top.msvt
                                        Started: Fri Jan 8 16:50:41 2021
Checking IRS connectivity against parameter file
_____
The following instances do not belong to any routing region:
   PF CCC CO 0/PF CCC CO 0/pll inst 0
   REF CLK 0 ibuf/U IOIN
The following IRS nets are not constrained by any routing region:
   block4 0 TX
  block1 0 TX
Analyzing floorplan ...
          ____
   block4 0 and block2 0 : Minimal floorplan separation = 9 clusters.
      block4_0 at cluster (144,62)
      block2_0 at cluster (144,52)
   block4_0 and block2_0 : Minimal placement separation = 21 clusters.
      (2148,156) containing cell block4 0/BUFD 1/U0
      (2148,225) containing cell block2 0/BUFD 0/U0
   block4 0 and block3 0 : Minimal floorplan separation = 11 clusters.
      block4 0 at cluster (99,27)
      block3 0 at cluster (87,27)
   block4_0 and block3_0 : Minimal placement separation = 11 clusters.
      (1211,82) containing cell block4_0/CoreGPIO_C4_0/CoreGPIO_C4_0/inData s2[6]
      (1057,81) containing cell block3 0/APB dp fp 1/U0/i post norm mul/s sh12 RNIS34841[4]
   block4 0 and block1 0 : Minimal floorplan separation = 11 clusters.
      block4_0 at cluster (99,64)
block1_0 at cluster (99,52)
   block4 0 and block1 0 : Minimal placement separation = 13 clusters.
      (1368,156) containing cell block4 0/BUFD 0/U0
      (1368,201) containing cell block1 0/BUFD 0/U0
   block4_0 and pf_{smip}_0: Minimal floorplan separation = 37 clusters.
      block4 0 at cluster (99,0)
   pf_smip_0 at cluster (61,0)
block4_0 and pf_smip_0 : Minimal placement separation = 38 clusters.
      (1219,2) containing cell block4_0/block4_IO_0/OUTBUF_31/U_IOTRI
      (746,2) containing cell pf smip 0/PF IO C1 0/PF IO C1 0/I IOD 0
   block4 0 and 'others' : Minimal floorplan separation = overlapping.
      block4 0 at cluster (99,0)
```

Libero[®] SoC v2021.2

Example

'others' at cluster (99,0) block4 0 and 'others' : Minimal placement separation = 0 clusters. (1219,2) containing cell block4_0/block4_IO_0/OUTBUF_31/U_IOTRI (1202,2) containing cell RESETN ibuf/U IOIN block2 0 and block3 0 : Minimal floorplan separation = diagonal. block2 0 and block3 0 : Minimal placement separation = diagonal. block2_0 and block1_0 : Minimal floorplan separation = 9 clusters. block2 0 at cluster (144,93) block1 0 at cluster (134,93) block2 0 and block1 0 : Minimal placement separation = 9 clusters. (1743,282) containing cell block2 0/BUFD 53/U0 (1620,282) containing cell block1_0/BUFD_87/U0 block2_0 and pf_smip_0 : Minimal floorplan separation = diagonal. block2_0 and pf_smip_0 : Minimal placement separation = diagonal. block2_0 and 'others' : Minimal floorplan separation = 9 clusters. block2 0 at cluster (144,62) 'others' at cluster (144,52) block2 0 and 'others' : Minimal placement separation = diagonal. block3 0 and block1 0 : Minimal floorplan separation = 10 clusters. block3 0 at cluster (38,64) block1 0 at cluster (38,53) block3_0 and block1_0 : Minimal placement separation = 22 clusters. (842,124) containing cell block3_0/CoreGPI0_C2_0/CoreGPI0_C2_0/dataOut[7] (842,196) containing cell block1_0/CoreGPI0_C0_0/CoreGPI0_C0_0/inData_s1[7] block3_0 and pf_smip_0 : Minimal floorplan separation = 4 clusters. block3_0 at cluster (66,0) pf_smip_0 at cluster (61,0) $block\overline{3}_0$ and pf_{smip}_0 : Minimal placement separation = 4 clusters. (811,2) containing cell block3 0/Block3 IO 0/INBUF 17/U IOIN (746,2) containing cell pf smip 0/PF IO C1 0/PF IO C1 0/I IOD 0 block3_0 and 'others' : Minimal floorplan separation = 11 clusters. block3 0 at cluster (99,0) 'others' at cluster (87,0) block3_0 and 'others' : Minimal placement separation = 15 clusters. (1010,2) containing cell block3_0/Block3_10_0/INBUF_19/U_IOIN (1202,2) containing cell RESETN_ibuf/U_IOIN block1_0 and pf_smip_0 : Minimal floorplan separation = 60 clusters. block1_0 at cluster (38,64) pf_smip_0 at cluster (38,3) block 0 and pf smip 0 : Minimal placement separation = diagonal. block1_0 and 'others' : Minimal floorplan separation = 11 clusters. block1_0 at cluster (99,64) 'others' at cluster (99,52) block1 0 and 'others' : Minimal placement separation = 66 clusters. (1204,204) containing cell block1 0/MIV RV32IMC C0 0/MIV RV32IMC C0 0/u opsrv 0/ u_core_0/u_lsu_0/un1_lsu_expipe_req_op_2 (1202,2) containing cell RESETN_ibuf/U_IOIN pf_smip_0 and 'others' : Minimal floorplan separation = 37 clusters. pf_smip_0 at cluster (61,0) 'others' at cluster (99,0) pf_smip_0 and 'others' : Minimal placement separation = 37 clusters. (746,2) containing cell pf_smip_0/PF_I0_C1_0/PF_I0_C1_0/I_IOD_0 (1202,2) containing cell RESETN ibuf/U IOIN Checking internal nets for block block4_0 ... _____ Checking IRS nets for block block4 0 ... ______ _____ Propagating IRS nets outgoing from block4 0 to block2 0 ______ Propagating IRS nets outgoing from block4 0 to block1 0 _____

Libero[®] SoC v2021.2 Example

Checking internal nets for block block2 0 ... _____ Checking IRS nets for block block2 0 ... _____ Propagating IRS nets outgoing from block2_0 to block1_0 _____ Checking internal nets for block block3 0 ... _____ Checking IRS nets for block block3_0 ... Propagating IRS nets outgoing from block3 0 to block4 0 ______ _____ Propagating IRS nets outgoing from block3_0 to block1_0 _____ Checking internal nets for block block1_0 ... _____ Checking IRS nets for block block1 0 ... _____ _____ Propagating IRS nets outgoing from block1_0 to block4_0 _____ Propagating IRS nets outgoing from block1 0 to block2 0 Propagating IRS nets outgoing from block1 0 to block3 0 Checking internal nets for block pf_smip_0 ... _____ Checking IRS nets for block pf_smip_0 ... _____ Design has met 2 switches separation requirement

MSVT Check succeeded. Number of errors: $\ensuremath{\texttt{0}}$

3. Revision History

Revision	Date	Description
В	08/2021	This document is released with Libero SoC Design Suite v2021.2 without changes from v2021.1.
А	04/2021	Initial Revision

4. Microchip FPGA Technical Support

Microchip FPGA Products Group backs its products with various support services, including Customer Service, Customer Technical Support Center, a website, and worldwide sales offices. This section provides information about contacting Microchip FPGA Products Group and using these support services.

4.1 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**

4.2 Customer Technical Support

Microchip FPGA Products Group staffs its Customer Technical Support Center with highly skilled engineers who can help answer your hardware, software, and design questions about Microchip FPGA 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.

You can communicate your technical questions through our Web portal and receive answers back by email, fax, or phone. Also, if you have design problems, you can upload your design files to receive assistance. We constantly monitor the cases created from the web portal 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.

Technical support can be reached at soc.microsemi.com/Portal/Default.aspx.

For technical support on RH and RT FPGAs that are regulated by International Traffic in Arms Regulations (ITAR), log in at soc.microsemi.com/Portal/Default.aspx, go to the **My Cases** tab, and select **Yes** in the ITAR drop-down list when creating a new case. For a complete list of ITAR-regulated Microchip FPGAs, visit the ITAR web page.

You can track technical cases online by going to My Cases.

4.3 Website

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

4.4 Outside the U.S.

Customers needing assistance outside the US time zones can either contact technical support at (https://soc.microsemi.com/Portal/Default.aspx) or contact a local sales office.

Visit About Us for sales office listings and corporate contacts.

The Microchip Website

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

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

Product Change Notification Service

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

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

Customer Support

Users of Microchip products can receive assistance through several channels:

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

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

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

Microchip Devices Code Protection Feature

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

- Microchip products meet the specifications contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is secure when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods being used in attempts to breach the code protection features of the Microchip devices. We believe that these methods require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Attempts to breach these code protection features, most likely, cannot be accomplished without violating Microchip's intellectual property rights.
- · Microchip is willing to work with any customer who is concerned about the integrity of its code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of its code. Code protection does not mean that we are guaranteeing the product is "unbreakable." Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Legal Notice

Information contained in this publication is provided for the sole purpose of designing with and using Microchip products. Information regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications.

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

IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE, INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE INFORMATION OR ITS USE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THE INFORMATION OR ITS USE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THE INFORMATION. Use of Microchip devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights unless otherwise stated.

Trademarks

The Microchip name and logo, the Microchip logo, Adaptec, AnyRate, AVR, AVR logo, AVR Freaks, BesTime, BitCloud, chipKIT, chipKIT logo, CryptoMemory, CryptoRF, dsPIC, FlashFlex, flexPWR, HELDO, IGLOO, JukeBlox, KeeLoq, Kleer, LANCheck, LinkMD, maXStylus, maXTouch, MediaLB, megaAVR, Microsemi, Microsemi logo, MOST, MOST logo, MPLAB, OptoLyzer, PackeTime, PIC, picoPower, PICSTART, PIC32 logo, PolarFire, Prochip Designer, QTouch, SAM-BA, SenGenuity, SpyNIC, SST, SST Logo, SuperFlash, Symmetricom, SyncServer, Tachyon, TimeSource, tinyAVR, UNI/O, Vectron, and XMEGA are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

AgileSwitch, APT, ClockWorks, The Embedded Control Solutions Company, EtherSynch, FlashTec, Hyper Speed Control, HyperLight Load, IntelliMOS, Libero, motorBench, mTouch, Powermite 3, Precision Edge, ProASIC, ProASIC Plus, ProASIC Plus logo, Quiet-Wire, SmartFusion, SyncWorld, Temux, TimeCesium, TimeHub, TimePictra, TimeProvider, WinPath, and ZL are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Adjacent Key Suppression, AKS, Analog-for-the-Digital Age, Any Capacitor, AnyIn, AnyOut, Augmented Switching, BlueSky, BodyCom, CodeGuard, CryptoAuthentication, CryptoAutomotive, CryptoCompanion, CryptoController, dsPICDEM, dsPICDEM.net, Dynamic Average Matching, DAM, ECAN, Espresso T1S, EtherGREEN, IdealBridge, In-Circuit Serial Programming, ICSP, INICnet, Intelligent Paralleling, Inter-Chip Connectivity, JitterBlocker, maxCrypto, maxView, memBrain, Mindi, MiWi, MPASM, MPF, MPLAB Certified logo, MPLIB, MPLINK, MultiTRAK, NetDetach, Omniscient Code Generation, PICDEM, PICDEM.net, PICkit, PICtail, PowerSmart, PureSilicon, QMatrix, REAL ICE, Ripple Blocker, RTAX, RTG4, SAM-ICE, Serial Quad I/O, simpleMAP, SimpliPHY, SmartBuffer, SMART-I.S., storClad, SQI, SuperSwitcher, SuperSwitcher II, Switchtec, SynchroPHY, Total Endurance, TSHARC, USBCheck, VariSense, VectorBlox, VeriPHY, ViewSpan, WiperLock, XpressConnect, and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

The Adaptec logo, Frequency on Demand, Silicon Storage Technology, and Symmcom are registered trademarks of Microchip Technology Inc. in other countries.

GestIC is a registered trademark of Microchip Technology Germany II GmbH & Co. KG, a subsidiary of Microchip Technology Inc., in other countries.

All other trademarks mentioned herein are property of their respective companies.

[©] 2021, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

ISBN: 978-1-5224-8522-3

Quality Management System

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



Worldwide Sales and Service

ASIA/PACIFIC		
Austrolia Sudrau	ASIA/PACIFIC	EUROPE
	-	Austria - Wels Tel: 43-7242-2244-39
		Fax: 43-7242-2244-39
		Denmark - Copenhagen
•		Tel: 45-4485-5910
		Fax: 45-4485-2829
•. •	•	Finland - Espoo
		Tel: 358-9-4520-820
•••		France - Paris
		Tel: 33-1-69-53-63-20
-		Fax: 33-1-69-30-90-79
		Germany - Garching
-		Tel: 49-8931-9700
		Germany - Haan
		Tel: 49-2129-3766400
		Germany - Heilbronn
		Tel: 49-7131-72400
		Germany - Karlsruhe
-		Tel: 49-721-625370
		Germany - Munich
China - Shanghai	Singapore	Tel: 49-89-627-144-0
		Fax: 49-89-627-144-44
China - Shenyang	Taiwan - Hsin Chu	Germany - Rosenheim
Tel: 86-24-2334-2829	Tel: 886-3-577-8366	Tel: 49-8031-354-560
China - Shenzhen	Taiwan - Kaohsiung	Israel - Ra'anana
Tel: 86-755-8864-2200	Tel: 886-7-213-7830	Tel: 972-9-744-7705
China - Suzhou	Taiwan - Taipei	Italy - Milan
Tel: 86-186-6233-1526	Tel: 886-2-2508-8600	Tel: 39-0331-742611
China - Wuhan	Thailand - Bangkok	Fax: 39-0331-466781
Tel: 86-27-5980-5300	Tel: 66-2-694-1351	Italy - Padova
China - Xian	Vietnam - Ho Chi Minh	Tel: 39-049-7625286
Tel: 86-29-8833-7252	Tel: 84-28-5448-2100	Netherlands - Drunen
China - Xiamen		Tel: 31-416-690399
Tel: 86-592-2388138		Fax: 31-416-690340
China - Zhuhai		Norway - Trondheim
Tel: 86-756-3210040		Tel: 47-72884388
		Poland - Warsaw
		Tel: 48-22-3325737
		Romania - Bucharest
		Tel: 40-21-407-87-50
		Spain - Madrid
		Tel: 34-91-708-08-90
		Fax: 34-91-708-08-91
		Sweden - Gothenberg
		Tel: 46-31-704-60-40
		Sweden - Stockholm
		Tel: 46-8-5090-4654
		UK - Wokingham
		Tel: 44-118-921-5800
		Fax: 44-118-921-5820
		1 a. ++-110-321-3020
	Tel: 86-21-3326-8000 China - Shenyang Tel: 86-24-2334-2829 China - Shenzhen Tel: 86-755-8864-2200 China - Suzhou Tel: 86-186-6233-1526 China - Wuhan Tel: 86-27-5980-5300 China - Xian Tel: 86-29-8833-7252 China - Xiamen Tel: 86-592-2388138 China - Zhuhai	Tel: 61-2-9868-6733 Tel: 91-80-3090-4444 China - Beijing India - New Delhi Tel: 86-10-8569-7000 Tel: 91-11-4160-8631 China - Chengdu India - Pune Tel: 86-28-8665-5511 Tel: 91-20-4121-0141 China - Chongqing Japan - Osaka Tel: 86-23-8980-9588 Tel: 81-6-6152-7160 China - Dongguan Japan - Tokyo Tel: 86-769-8702-9880 Tel: 81-3-6880-3770 China - Guangzhou Korea - Daegu Tel: 86-71-8792-8115 Tel: 82-2-554-7200 China - Hong Kong SAR Malaysia - Valau Lumpur Tel: 86-571-8792-8115 Tel: 60-3-7651-7906 China - Nanjing Malaysia - Penang Tel: 86-512-8502-7355 Tel: 60-3-2634-9065 China - Shanghai Singapore Tel: 86-21-3326-8000 Tel: 86-334-8870 Tel: 86-21-3326-8000 Tel: 86-3577-8366 China - Shenyang Taiwan - Hsin Chu Tel: 86-755-8864-2200 Tel: 886-3-771-830 Tel: 86-186-6233-1526 Tel: 886-7-213-7830 China - Suzhou Taiwan - Taipei Tel: 86-186-6233-1526 Tel: 886-2-2508-8600 China - Xian