



XAPP584 (v1.0) July 10, 2012

# Spartan-6 FPGA Dual-Lockstep MicroBlaze Processor with Isolation Design Flow

Author: Trevor Hardcastle

## Summary

This application note describes the creation of a dual-lockstep MicroBlaze™ processor system on a Spartan®-6 LX150T device. This system is then implemented using the Xilinx Isolation Design Flow (IDF) to separate it into isolated functions and regions. Finally, design preservation is used to lock down the isolated regions and functions.

## Overview

This application note describes how to implement a dual-lockstep MicroBlaze processor system on a Spartan-6 LX150T device using the Embedded Design Kit (EDK) Platform Studio. EDK Platform Studio is included in the Xilinx ISE® Design Suite, version 13.4. An introduction to the IDF is provided, along with how to apply the rules and considerations of the IDF to separate the dual-lockstep MicroBlaze processor system into five physically isolated and independent functions within the Spartan-6 device. Finally, a method is presented for locking down the design using design preservation techniques available in the Xilinx tools.

This application note has many purposes, but primarily it describes:

1. Building a dual-lockstep MicroBlaze processor using the EDK Platform Studio.
2. Introducing the Xilinx IDF.
3. Applying the IDF to an EDK-based system.
4. Using the PlanAhead™ design tool to fully execute a bottom-up implementation.
5. Using design preservation techniques to lock down design elements.

## Reference Design Overview

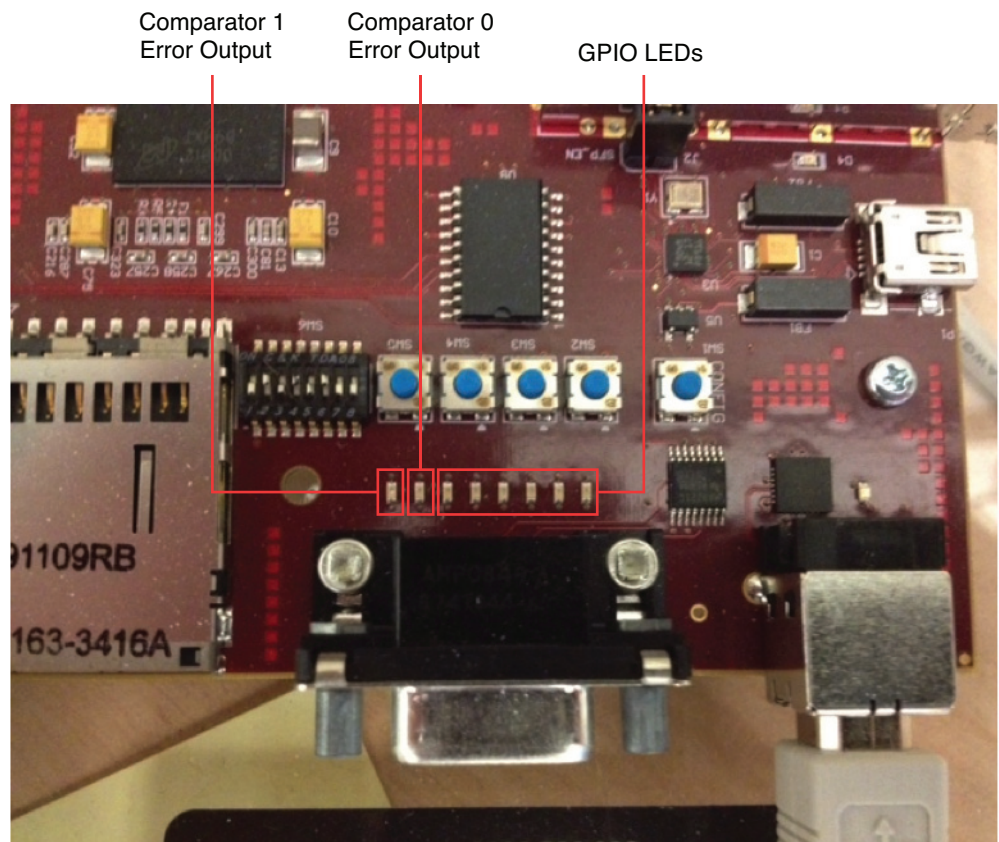
The dual-lockstep MicroBlaze processor system described in this application note is implemented on the Avnet Spartan-6 FPGA LX150T development board using the ISE Design Suite 13.4: System Edition. The system implements two MicroBlaze processors, each with their own local instruction and data memory. The system includes these peripherals:

- DDR3 SDRAM interface
- One RS-232 interface
- General-purpose I/O to interface to pushbuttons, DIP switches, and LEDs
- Linear flash interface
- Timer
- Timebase
- Peripheral block RAM
- Interrupt Controller
- One instance of the MicroBlaze Debug Module (MDM)

The system is implemented based on the AXI interconnect, which operates at 50 MHz. The DDR3 SDRAM operates at 650 MHz.

In the dual-lockstep system, both MicroBlaze processors get the same code loaded at start-up. The first MicroBlaze processor has full control over the AXI interconnect, driving the

instructions to the peripherals, and monitoring the responses through the AXI interconnect. The redundant MicroBlaze processor only monitors the response on the AXI interconnect so that it can respond as if it were fully controlling the AXI interconnect, but it does not physically drive the interconnect. This makes the first processor the system's master processor and the second processor the redundant checker. Redundant comparators monitor the outputs of both MicroBlaze processors to make sure that the processors are executing the exact same code, cycle for cycle. If an error is detected on a comparator, an LED is lit on the Avnet Spartan-6 FPGA LX150T development board. One LED is connected to each comparator. [Figure 1](#) shows a picture of the Avnet Spartan-6 FPGA LX150T development board and highlights the LEDs that are illuminated when the comparators detect an error.



X584\_01\_040412

*Figure 1: LED Distribution*

For debug support, the first MicroBlaze processor's LOCKSTEP\_MASTER port is initially connected to the second MicroBlaze processor. The MDM is also included in the system. Both of these debug features are disconnected at the end of this application note to show how design preservation can be used on a design to lock down isolated regions when making design changes at the top level.

When IDF is applied, the dual-lockstep system is partitioned into five isolated functions:

1. MB0\_TOP
2. MB1\_TOP
3. PERIPHERALS\_TOP
4. MB0\_COMPARATOR\_TOP
5. MB1\_COMPARATOR\_TOP

Each MB<sub>x</sub>\_TOP function implements a MicroBlaze processor instance and its local instruction and data memory. Each MB<sub>x</sub>\_COMPARATOR\_TOP function implements an instance of the



MicroBlaze Comparator (see [MicroBlaze Comparator User Guide](#)). The U4\_PERIPHERALS\_TOP function implements all the AXI interconnects, system peripherals, and the MDM. A block diagram of the final reference design is provided in [Figure 2](#).

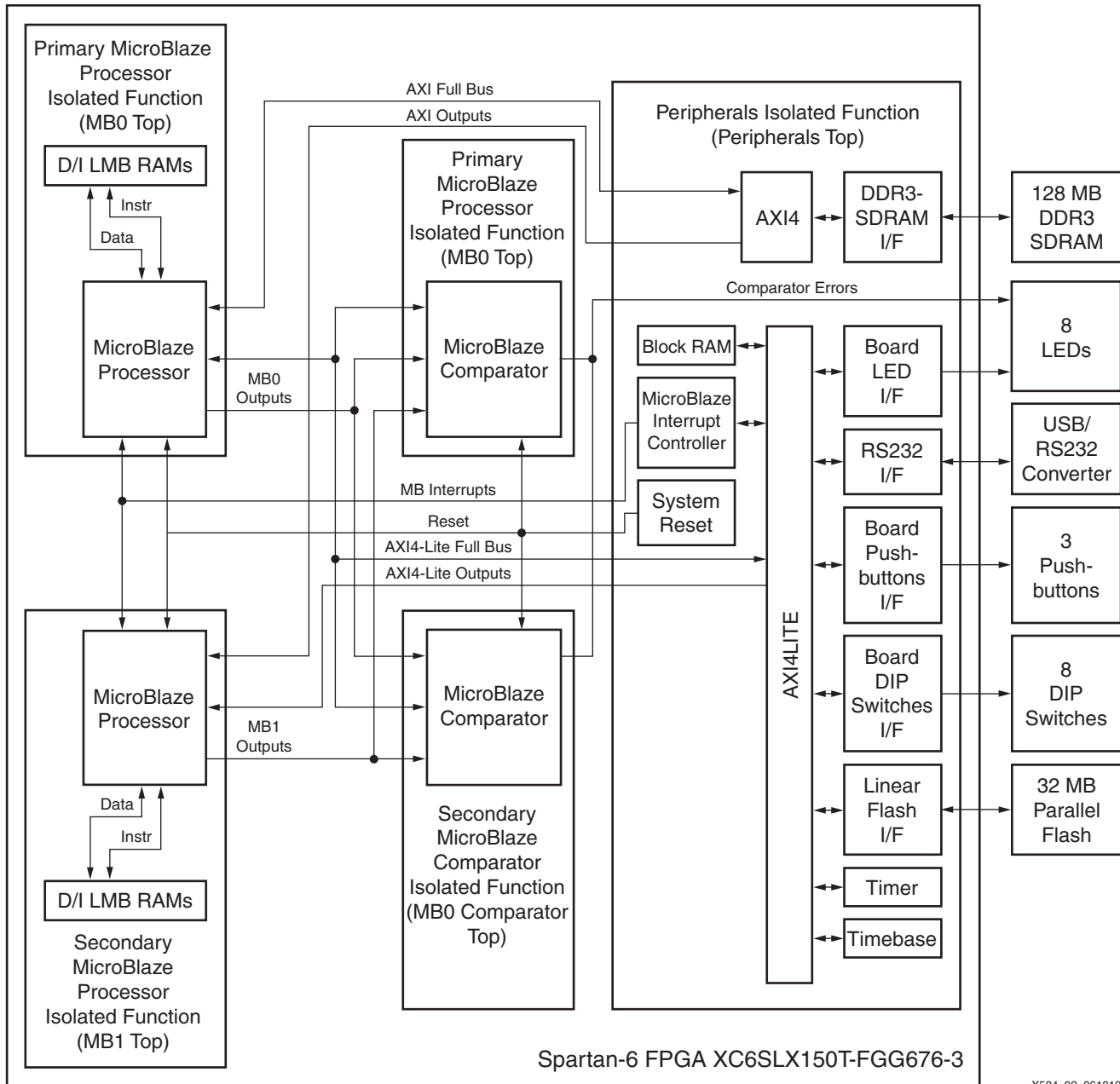


Figure 2: Final Reference Design Block Diagram

Further discussions of use cases for the lockstep MicroBlaze processor and descriptions of the lockstep output can be found in *MicroBlaze Processor Reference Guide Embedded Development Kit* [Ref 1].

### MicroBlaze Comparator User Guide

The dual-lockstep MicroBlaze processor system uses dual-redundant comparators to compare the outputs of each of the MicroBlaze processors, cycle for cycle. This comparator, formally known as the *MicroBlaze Comparator*, is delivered as a pcore in this application note. The MicroBlaze Comparator is addressable through either the AXI or processor local bus (PLB)

interconnects. The comparator provides a single bit error output along with an error bus identification output. The comparator is fully synchronous, and connects to each of the MicroBlaze processors through their lockstep output ports.

The MicroBlaze Comparator is primarily controlled through the AXI or PLB register interface. A status register is implemented to indicate which MicroBlaze processor interface had a comparison mismatch or error. Test error insertion is also enabled and controlled through the control registers, allowing a specific MicroBlaze processor, MicroBlaze processor interface, and interface bit to be targeted for error injection. After test error insertion is enabled, a test error is injected for one clock cycle. A control bit is provided to clear comparison errors. The comparison error indication remains asserted until the error is cleared either through the control bit or the Error\_Clear input. Figure 3 shows the MicroBlaze Comparator ports.



X584\_03\_041112

Figure 3: MicroBlaze Comparator Ports

Each MicroBlaze Comparator implements five registers described in [Table 1](#).

**Table 1: MicroBlaze Comparator Register Details**

Register Details	Description	
<b>Control Register</b>		
Address	Comparator base address + 0x00	
R/W	R/W	
Description	Control register for enabling fault injection and clearing of faults. All bits are active-High.	
Bit Descriptions	Bit 0 (msb) – Bit 29	Unused
	Bit 30	Enable fault inject.
	Bit 31 (lsb)	Clear fault (clears the status register and the error output).
<b>Status Register</b>		
Address	Comparator base address + 0x04	
R/W	R	
Description	Status register to indicate which interface had a miscompare since the last clearing of the fault. All bit indications are active-High.	
Bit Descriptions	<b>Bit</b>	<b>Description</b>
	0 (msb) – 3	Unused
	4	IPLB
	5	DPLB
	6	IXCL
	7	DXCL
	8	TRACE
	9	DEBUG
	10	AXI_IC
	11	AXI_DC
	12	AXI_IP
	13	AXI_DP
	14	ILMB
	15	DLMB
	16	FSL0/AXIS0
	17	FSL1/AXIS1
	18	FSL2/AXIS2
	19	FSL3/AXIS3
	20	FSL4/AXIS4
	21	FSL5/AXIS5
	22	FSL6/AXIS6
	23	FSL7/AXIS7
	24	FSL8/AXIS8
	25	FSL9/AXIS9

Table 1: MicroBlaze Comparator Register Details (Cont'd)

Register Details	Description	
Bit Descriptions (Cont'd)	26	FSL10/AXIS10
	27	FSL11/AXIS11
	28	FSL12/AXIS12
	29	FSL13/AXIS13
	30	FSL14/AXIS14
	31 (lsb)	FSL15/AXIS15
<b>Interface Fault Register 1</b>		
Address	Comparator base address + 0x08	
R/W	R/W	
Description	Control register that enables fault injection on the specified interface for MicroBlaze 1 (LOCKSTEP1). All bits are active-High.	
Bit Descriptions	<b>Bit</b>	<b>Description</b>
	0 (msb) – 3	Unused
	4	IPLB
	5	DPLB
	6	IXCL
	7	DXCL
	8	TRACE
	9	DEBUG
	10	AXI_IC
	11	AXI_DC
	12	AXI_IP
	13	AXI_DP
	14	ILMB
	15	DLMB
	16	FSL0/AXIS0
	17	FSL1/AXIS1
	18	FSL2/AXIS2
	19	FSL3/AXIS3
	20	FSL4/AXIS4
	21	FSL5/AXIS5
22	FSL6/AXIS6	
23	FSL7/AXIS7	
24	FSL8/AXIS8	
25	FSL9/AXIS9	
26	FSL10/AXIS10	
27	FSL11/AXIS11	



Table 1: MicroBlaze Comparator Register Details (Cont'd)

Register Details	Description	
Bit Descriptions (Cont'd)	28	FSL12/AXIS12
	29	FSL13/AXIS13
	30	FSL14/AXIS14
	31 (lsb)	FSL15/AXIS15
<b>Interface Fault Register 2</b>		
Address	Comparator base address + 0x0C	
R/W	R/W	
Description	Control register that enables fault injection on the specified interface for MicroBlaze 2 (LOCKSTEP2). All bits are active-High.	
Bit Descriptions	<b>Bit</b>	<b>Description</b>
	0 (msb) – 3	Unused
	4	IPLB
	5	DPLB
	6	IXCL
	7	DXCL
	8	TRACE
	9	DEBUG
	10	AXI_IC
	11	AXI_DC
	12	AXI_IP
	13	AXI_DP
	14	ILMB
	15	DLMB
	16	FSL0/AXIS0
	17	FSL1/AXIS1
	18	FSL2/AXIS2
	19	FSL3/AXIS3
	20	FSL4/AXIS4
	21	FSL5/AXIS5
22	FSL6/AXIS6	
23	FSL7/AXIS7	
24	FSL8/AXIS8	
25	FSL9/AXIS9	
26	FSL10/AXIS10	
27	FSL11/AXIS11	
28	FSL12/AXIS12	
29	FSL13/AXIS13	

Table 1: MicroBlaze Comparator Register Details (Cont'd)

Register Details	Description	
Bit Descriptions (Cont'd)	30	FSL14/AXIS14
	31 (lsb)	FSL15/AXIS15
<b>Fault Inject Bit Register</b>		
Address	Comparator base address + 0x10	
R/W	R/W	
Description	Control register that specifies which bit within an interface to inject the fault on.	
Bit Descriptions	Bit	Description
	0 (msb) – 20	Unused
	21 – 31 (lsb)	Specifies which bit within an interface to inject the fault on. The interfaces have the following number of bits: <ul style="list-style-type: none"> <li>• IPLB = 98 + (IPLB_WIDTH/8) + IPLB_WIDTH</li> <li>• DPLB = 98 + (DPLB_WIDTH/8) + DPLB_WIDTH</li> <li>• IXCL = 35</li> <li>• DXCL = 35</li> <li>• Trace = 209</li> <li>• AXI_IC = 133 + (AXI_IC_WIDTH/8) + AXI_IC_WIDTH</li> <li>• AXI_DC = 133 + (AXI_DC_WIDTH/8) + AXI_DC_WIDTH</li> <li>• AXI_IP = 158</li> <li>• AXI_DP = 158</li> <li>• ILMB = 34</li> <li>• DLMB = 71</li> <li>• FSLx/AXISx = 35</li> </ul>

A sample driver is provided as part of the demonstration software included with this application note for executing basic functions within the comparator. Table 2 lists the steps for executing these basic functions.

Table 2: Steps to Execute Basic Functions in the Comparator

Basic Function	Steps to Execute
Inject a comparator error	<ol style="list-style-type: none"> <li>1. Enable a fault injection by setting the Enable Fault Injection bit in the control register to 1.</li> <li>2. Set the Fault Inject Bit register to the bit within the interface to inject an error.</li> <li>3. Set Interface Fault Register 1 or Interface Fault Register 2 to the interface to inject an error.</li> </ol>
Clear previous comparator errors	Set the Clear Status Register bit in the control register to 1.
Read comparison error since last reset, Error_Clear, or clear status register	Read the status register.

## IDF Overview

The IDF allows for multiple physically isolated and independent functions to be implemented within a single FPGA, utilizing a fence of unused device components between each function. Each isolated function is separated by this fence, generating isolated regions within the device. The flow uses early floorplanning, modular design, modular and bottom-up synthesis, and adherence to a set of rules and considerations to guarantee isolation between functions. After a design is implemented, the Isolation Verification Tool (IVT) can be used to visualize the

modules and fence, and verify that the design rules for isolation have been successfully implemented. More details of the IDF as it applies to a Spartan-6 device can be found in *Developing Secure Designs with the Spartan-6 Family Using the Isolation Design Flow* [Ref 2] and *Implementation of a Fail-Safe Design in the Spartan-6 Family using ISE Design Suite 12.4* [Ref 3].

Figure 4 illustrates the conceptual implementation of a design of four isolated functions separated into four isolated regions by fences using the IDF.

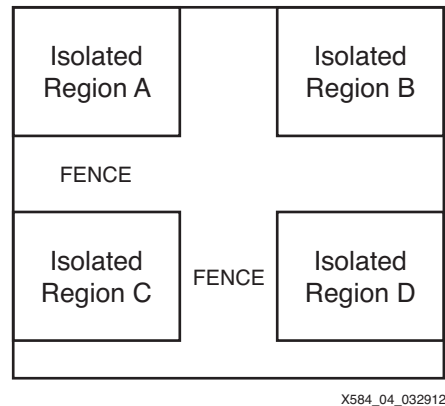


Figure 4: Isolation Design Flow Conceptual Design

The IDF is based on these rules and considerations:

1. Each isolated function must be at its own level of hierarchy in the hardware description language (HDL).

IDF uses design partitions that allow for the identification of isolated regions and functions. Isolated regions are floorplanned area groups of the device whose resources defined in the area group are physically reserved for implementing an isolated function. An isolated function is the HDL defined within a hierarchy that logically describes a function that is isolated from another HDL-defined function within the hierarchy. The design modules that exist below the “top” design file are the first level of modules that can be implemented as isolated functions (logical) into isolated regions (physical).

2. A fence must be used to separate isolated functions within a single device.

A *fence* is defined as one or multiple user tiles that exist between two isolated regions and does not contain routing or logic. The minimum required user tile number for a valid fence depends on the user tile type (i.e., DSP48, RAMB, SLICE, DCM, or PLL) that defines the fence and the device family (i.e., Spartan-6, Virtex®-5, or 7 series FPGAs). The results of the isolation analysis performed by Xilinx show that using one or sometimes two such user tiles placed between isolated regions guarantees that no single point of failure exists that would compromise the isolation between the two isolated regions. Fences are implemented with required manual floorplanning through the Xilinx tools.

3. Input/output buffers must be inferred or instantiated within the isolated function itself for proper isolation of the buffer.

Instantiating or inferring I/O buffers within the isolated function is required so that the buffer is considered part of the isolated function.

Global clock logic (i.e., DCMs, PLLs, or BUFGs) is the only type of logic allowed at the top level of the design’s hierarchy because, by definition, it spans the whole design. Global clock logic can be instantiated in its own module in the hierarchy to allow for cleaner design hierarchy in the HDL, but this HDL module cannot be included in its own isolated region so that global routing of the global clock is not prevented.

4. On-chip communication between isolated functions is achieved through the use of trusted routing.

Trusted routing is a set of resources that is selected automatically by the Xilinx tools to facilitate communication between isolated regions while maintaining the fence for physical separation. To use trusted routing, all components required for implementation of an isolated function must be part of the floorplanned isolated region, also referred to as the *partition* in Xilinx tools. Trusted routing requires that routing within an isolated region remain entirely within the isolated region. Communication and routing between multiple isolated regions is allowed based on these rules:

- Signals feeding through an isolated region are not allowed without buffering:
  - If a signal is directly connected to both an input port and an output port of an isolated function, it must be buffered.
- An isolated function's output port cannot connect to more than one isolated function or more than one isolated function's input port:
  - If a single output needs to connect to multiple isolated functions, the user must create separate output ports of the source output.
  - Each port must not violate the last rule.
- A single signal cannot drive two different output ports of the same module. Each port must have a unique driver:
  - The unique driver can be implemented using an instantiated look-up table (LUT) buffer or flip-flop.

Further details about the IDF as it applies to a Spartan-6 FPGA can be found in *Developing Secure Designs with the Spartan-6 Family Using the Isolation Design Flow* [Ref 2] and *Implementation of a Fail-Safe Design in the Spartan-6 Family using ISE Design Suite 12.4* [Ref 3].

## Reference Design Files

The reference design files accompanying this application can be downloaded from:

<https://secure.xilinx.com/webreg/clickthrough.do?cid=189247>

The design checklist in Table 3 includes simulation, implementation, and hardware details for the reference design.

Table 3: Reference Design Checklist

Parameter	Description
<b>General</b>	
Developer name	Xilinx
Target device	Spartan-6 FPGA
Source code provided	Yes
Source code format	VHDL
Design uses code/IP from existing Xilinx application note/reference designs, the CORE Generator™ tool, or a third party	Yes
<b>Simulation</b>	
Functional simulation performed	Yes
Timing simulation performed	No
Test bench used for functional and timing simulations	Yes
Test bench format	VHDL/Verilog
Simulator software/version used	ISim 13.4
SPICE/IBIS simulations	No



Table 3: Reference Design Checklist (Cont'd)

Parameter	Description
<b>Implementation</b>	
Synthesis software tools/version used	XST 13.4
Implementation software tools/version used	ISE Design Suite, version 13.4
Static timing analysis performed	No
<b>Hardware Verification</b>	
Hardware verified	Yes
Hardware platform used for verification	Avnet Spartan-6 FPGA LX150T development board

Table 4 shows the device utilization table.

Table 4: Device Utilization Table

Device	Speed Grade	Package	Pre-Map	Post-Route	Slices
XC6SLX150T	-3	FGG676	88.37 MHz	51.65 MHz	8,527 (37%)

## Installing the Reference Design Files into the Target Directories

These steps describe installing the reference design files into a Windows environment:

1. Install the ISE Design Suite, version 13.4.
2. Copy `xapp584.zip` to the Windows desktop.
3. Extract the contents of `xapp584.zip` to the user drive.

Design files need to be unzipped into a directory without any spaces in the path.

**Note:** Throughout this application note, the use of `<reference design>\` in the file path refers to the root installation directory chosen as part of [step 3](#) as well as the application note directory `xapp584`.

## Setting Up the Environment

### Installing Development Board Files

The dual-lockstep MicroBlaze processor system described in this application note is implemented on the Avnet Spartan-6 FPGA LX150T development board using ISE Design Suite, version 13.4. Avnet provides board development files for use by the ISE design tools to automatically generate the design pinout and some of the system constraints. The board development files are available on the Avnet website [\[Ref 4\]](#).

1. On the Spartan-6 FPGA LX150T Development Kit page of the Avnet website [\[Ref 4\]](#), click **Support Files and Downloads**.
2. Enter your Avnet Express Login credentials to access the Support and Downloads page.
 

**Note:** The Avnet website does require a user to register with the site in order to download the files.
3. Under the XBD section, click **EDK 13.3 XBD/IP-XACT files**.
4. Save the ZIP file to the local computer and extract the files.
5. Follow the instructions in the Quick Install Readme file located within the ZIP file to install the files in the Xilinx 13.4 directory.

## Environmental Variables

The dual-lockstep MicroBlaze processor system requires the XIL\_EDK\_XST\_OPTIONS environment variable be set to **-shreg\_extract no**:

- XIL\_EDK\_XST\_OPTIONS = **-shreg\_extract no**

The XIL\_EDK\_XST\_OPTIONS environment variable adds the **-shreg\_extract no** option to EDK synthesis to prevent shift register inference of SRL16 and SRL32. SRL16s and SRL32s should not be used in Spartan-6 FPGA designs to prevent false single-event upset (SEU) detection.

A batch script can be developed to set the environment variable by copying this text into a file with a .bat extension and double-clicking the file within a Windows Explorer window:

```
rem #####
rem # SET ENVIRONMENT VARIABLES
rem #####
set XIL_EDK_XST_OPTIONS=-shreg_extract no
```

## Building the Reference Design

The reference design is executed in twelve high-level steps, illustrated in [Figure 5](#). The first four steps are executed using the EDK tools Platform Studio and Software Development Kit (SDK). In EDK, a dual MicroBlaze processor system is created and then converted to a dual-lockstep MicroBlaze processor system. Then, for the fifth step, modifications are made to the VHDL files that describe the dual-lockstep MicroBlaze processor system to apply hierarchy and the necessary changes required by IDF. From there, the PlanAhead tool is used to partition, isolate, implement, and preserve the design. The IVT is also used throughout to verify the isolation of the design as IDF is applied.

1. Generate Dual MicroBlaze System in EDK Platform Studio.
2. Modify Dual MicroBlaze System into Lockstep through EDK Platform Studio.
3. Build the Dual-Lockstep MicroBlaze System in EDK Platform Studio.
4. Perform a Quick Sanity Check of the Design.
5. Prepare Dual-Lockstep MicroBlaze Design Source for Isolated Design.
6. Synthesize and Floorplan Hierarchical Design.
7. Run IVT on the Design in UCF Mode.
8. Implement the Design.
9. Run IVT on the Design in NCD Mode.
10. Build Final Software in SDK.
11. Disconnect the MicroBlaze Processors and Debug Logic.
12. Re-verify the Re-implemented Design in IVT.

X584\_05\_041112

Figure 5: Reference Design Steps

## Generating the Dual MicroBlaze Processor Base System in the EDK Platform Studio

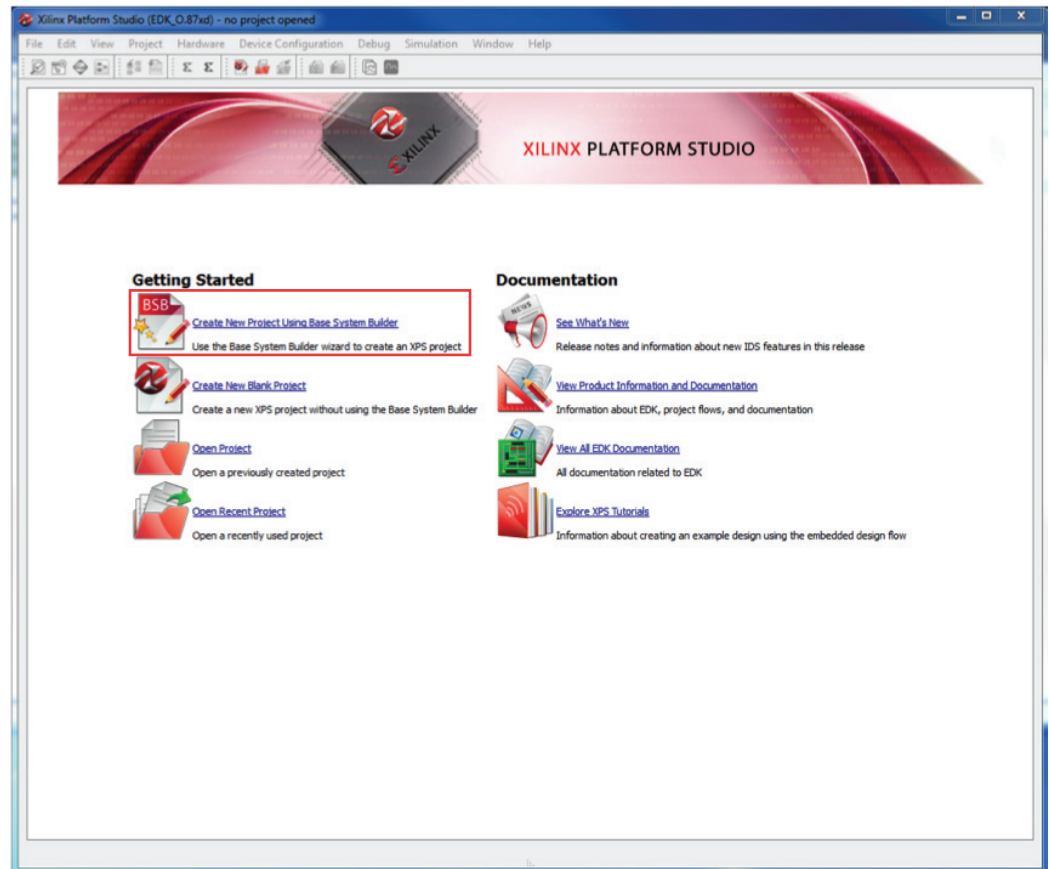
The steps in this section describe how to use the Platform Studio Base System Builder (BSB) to generate a dual MicroBlaze processor system with the AXI interconnect. This dual MicroBlaze processor system serves as the base system that is modified into the dual lockstep

MicroBlaze processor system. The peripheral selections and settings in this application note are chosen to keep the system simple and functional, but can be modified in accordance with user requirements for incorporation into a user design. The peripheral selections and settings are in no way locked down to make the dual-lockstep MicroBlaze processor.

1. Start EDK Platform Studio:

**Start > All Programs > Xilinx ISE Design Suite 13.4 > EDK > Xilinx Platform Studio**

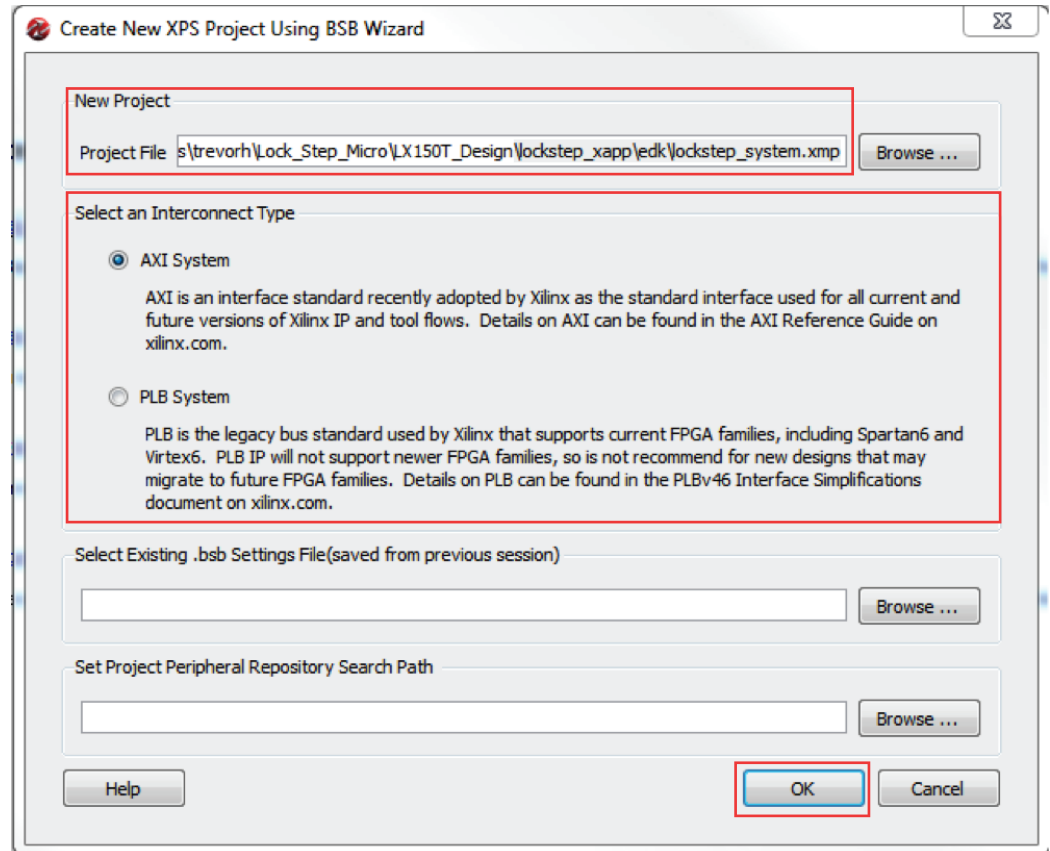
2. In the Platform Studio welcome screen ([Figure 6](#)), click **Create New Project Using Base System Builder** to start the Base System Builder wizard.



X584\_06\_041112

*Figure 6:* Platform Studio Welcome Screen

3. Make these selections in the Create New XPS Project Using BSB Wizard window to name the project lockstep\_system and base the system on the AXI interconnect (Figure 7), and then click **OK**:
  - Project File: <reference design>\edk\lockstep\_system.xmp
  - Interconnect Type: **AXI System**

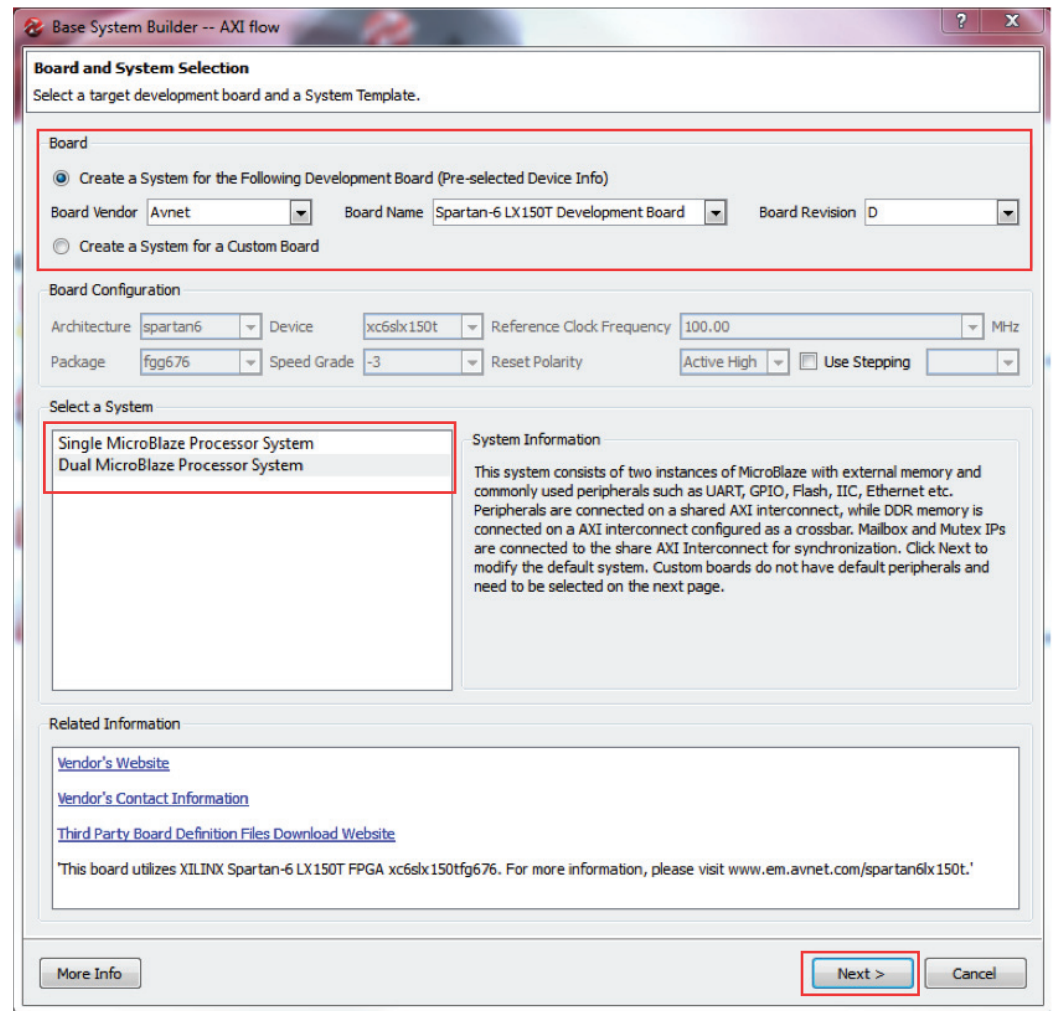


X584\_07\_040412

Figure 7: Create New XPS Project Using BSB Wizard Window in Platform Studio



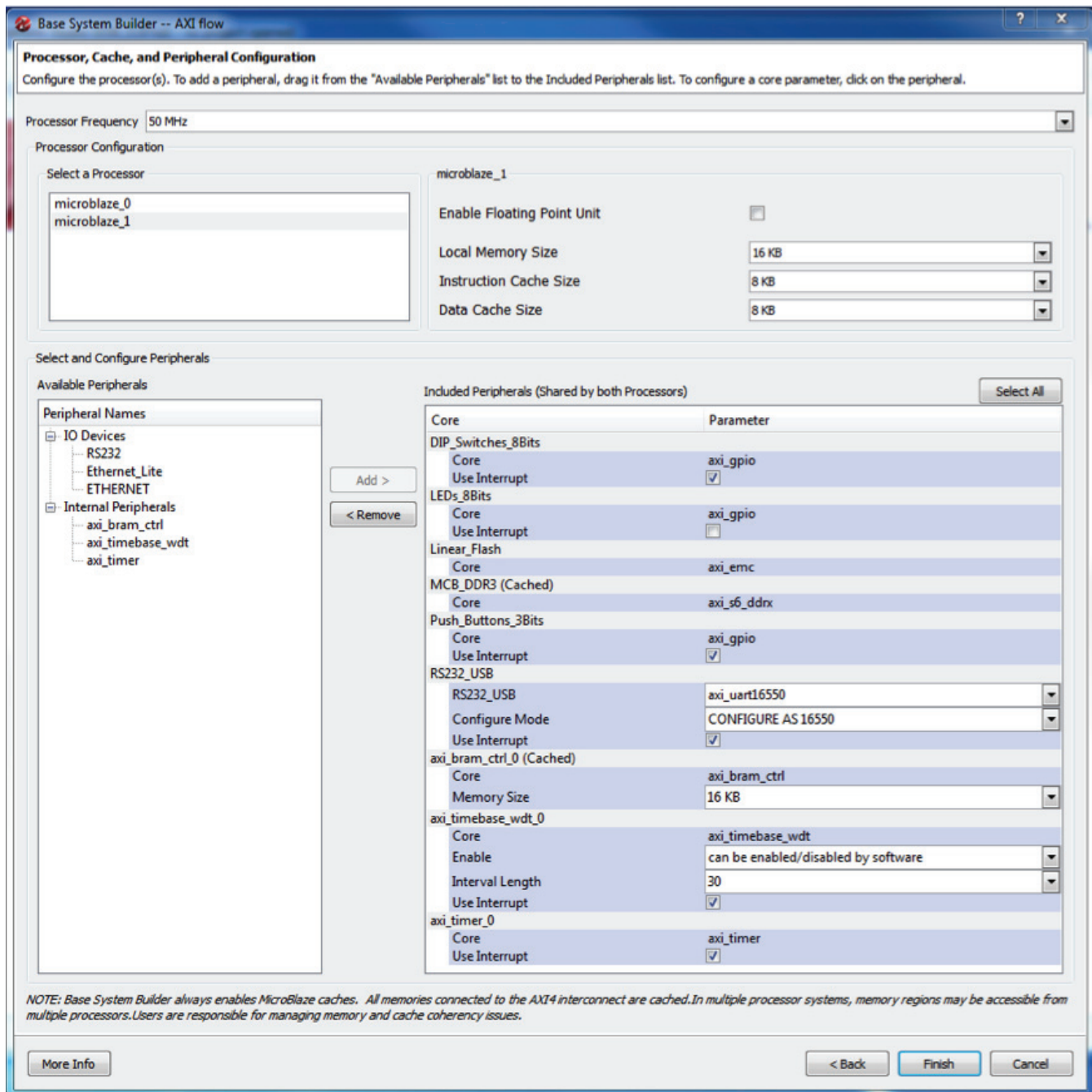
4. Make these selections in the Board and System Selection window (Figure 8) to select the Avnet Spartan-6 FPGA LX150T development board, then click **Next >**.
  - Board: **Create a System for the Following Development Board (Pre-Selected Device Info)**
  - Board Vendor: **Avnet**
  - Board Name: **Spartan-6 LX150T Development Board**
  - Board Revision: **D**
  - Select a System: **Dual MicroBlaze Processor System**



X584\_08\_040412

Figure 8: Board and System Selection Window

5. In the Processor, Cache, and Peripheral Configuration window (Figure 9) perform these steps to set the system clock frequency and to set up the memories and peripherals:



X584\_09\_040412

Figure 9: Processor, Cache, and Peripheral Configuration Window Settings

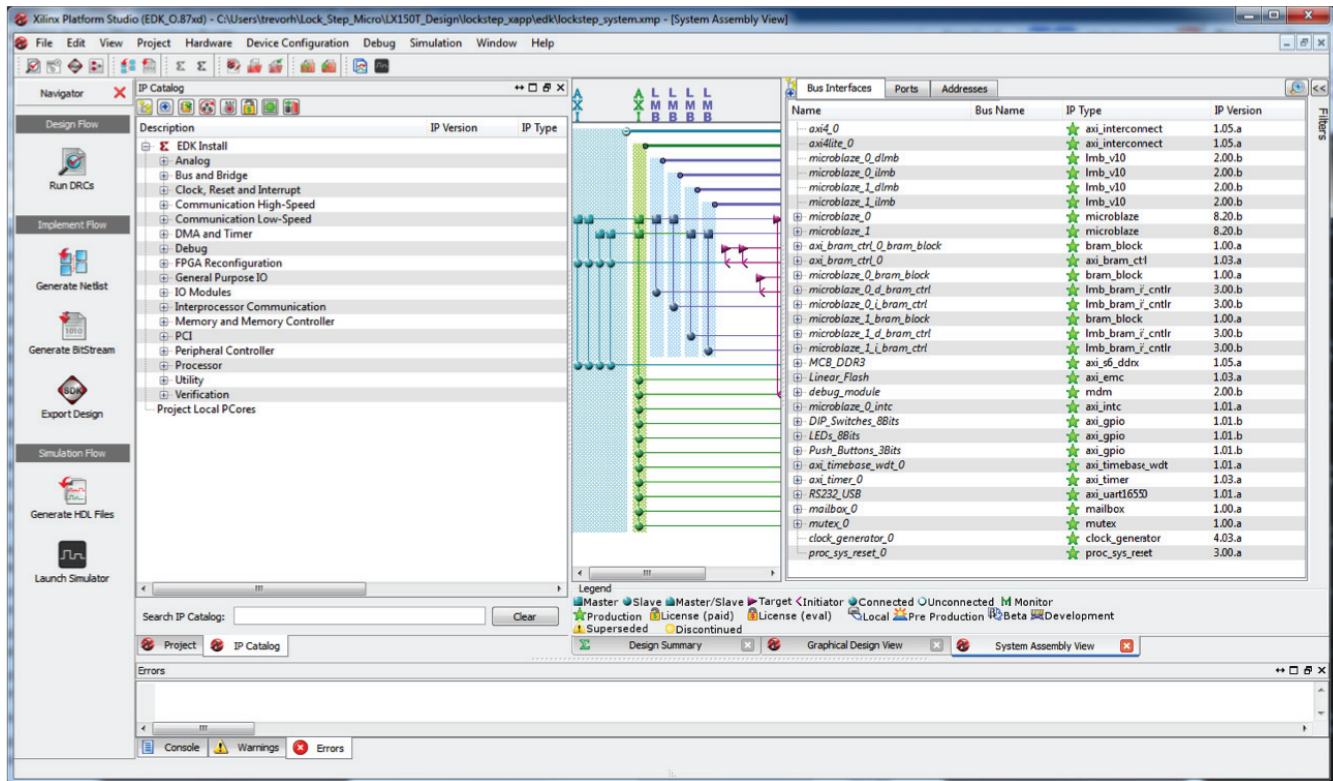
- a. Set the Processor Frequency to **50 MHz**.
- b. In the Processor Configuration section of the window, select **processor microblaze\_0** and make these selections:
  - Enable Floating Point Unit: **Unchecked**
  - Local Memory Size: **16 KB**
  - Instruction Cache Size: **8 KB**

- Data Cache Size: **8 KB**
- c. In the Processor Configuration section of the window, select **processor microblaze\_1** and make these selections to match **microblaze\_0**:
  - Enable Floating Point Unit: **Unchecked**
  - Local Memory Size: **16 KB**
  - Instruction Cache Size: **8 KB**
  - Data Cache Size: **8 KB**
- d. In the Select and Configure Peripherals > Available Peripherals section of the window, add one instance of each of these items:
  - **axi\_bram\_ctrl**
  - **axi\_timebase\_wdt**
  - **axi\_timer**
- e. In the **Select and Configure Peripherals > Included Peripherals** (shared by both processors) section of the window, perform these steps in sequence:
  - Remove the **Ethernet\_Lite** peripheral.
  - Remove the **RS232** peripheral, but keep the **RS232\_USB** peripheral.
  - Highlight **DIP\_Switches\_8bits** and check **Use\_Interrupt**.
  - Highlight **Push\_Buttons\_3bits** and check **Use\_Interrupt**.
  - Highlight **RS232\_USB** and set RS232\_USB to **axi\_uart16550, CONFIGURE AS 165550**, with **Use\_Interrupt** checked.
  - Highlight **axi\_bram\_ctrl\_0** and set the memory size to **16 KB**.
  - Highlight **axi\_timebase\_wdt\_0**. Set can be enabled and disabled by software, with an interval length of **30**, and **Use\_Interrupt** checked.
  - Highlight **axi\_timer\_0** and check **Use\_Interrupt**.

[Figure 9](#) shows the final settings for the Processor, Cache, and Peripheral Configuration.

- f. Click **Finish** to complete the generation of the system.

After Platform Studio generates the dual MicroBlaze processor system, the Platform Studio Main window appears showing the generated system ([Figure 10](#)).



X584\_10\_040412

Figure 10: Platform Studio Main Window Showing Generated Dual MicroBlaze Processor

## Modifying the Base Dual MicroBlaze Processor System into the Lockstep System

The dual MicroBlaze processor system that was created in Platform Studio serves as the starting point for generating the dual-lockstep MicroBlaze processor system (see Figure 11). The next steps describe the modifications that are made to the dual MicroBlaze processor system to convert it to the dual-lockstep MicroBlaze processor system. Some of the modifications are made through the Platform Studio GUI and some are made manually in the system's MHS file, which describes the system's characteristics and connections and is used by the Platform Studio GUI.



<input checked="" type="checkbox"/>	1. Generate Dual MicroBlaze System in EDK Platform Studio.
	2. Modify Dual MicroBlaze System into Lockstep through EDK Platform Studio.
	3. Build the Dual-Lockstep MicroBlaze System in EDK Platform Studio.
	4. Perform a Quick Sanity Check of the Design.
	5. Prepare Dual-Lockstep MicroBlaze Design Source for Isolated Design.
	6. Synthesize and Floorplan Hierarchical Design.
	7. Run IVT on the Design in UCF Mode.
	8. Implement the Design.
	9. Run IVT on the Design in NCD Mode.
	10. Build Final Software in SDK.
	11. Disconnect the MicroBlaze Processors and Debug Logic.
	12. Re-verify the Re-implemented Design in IVT.

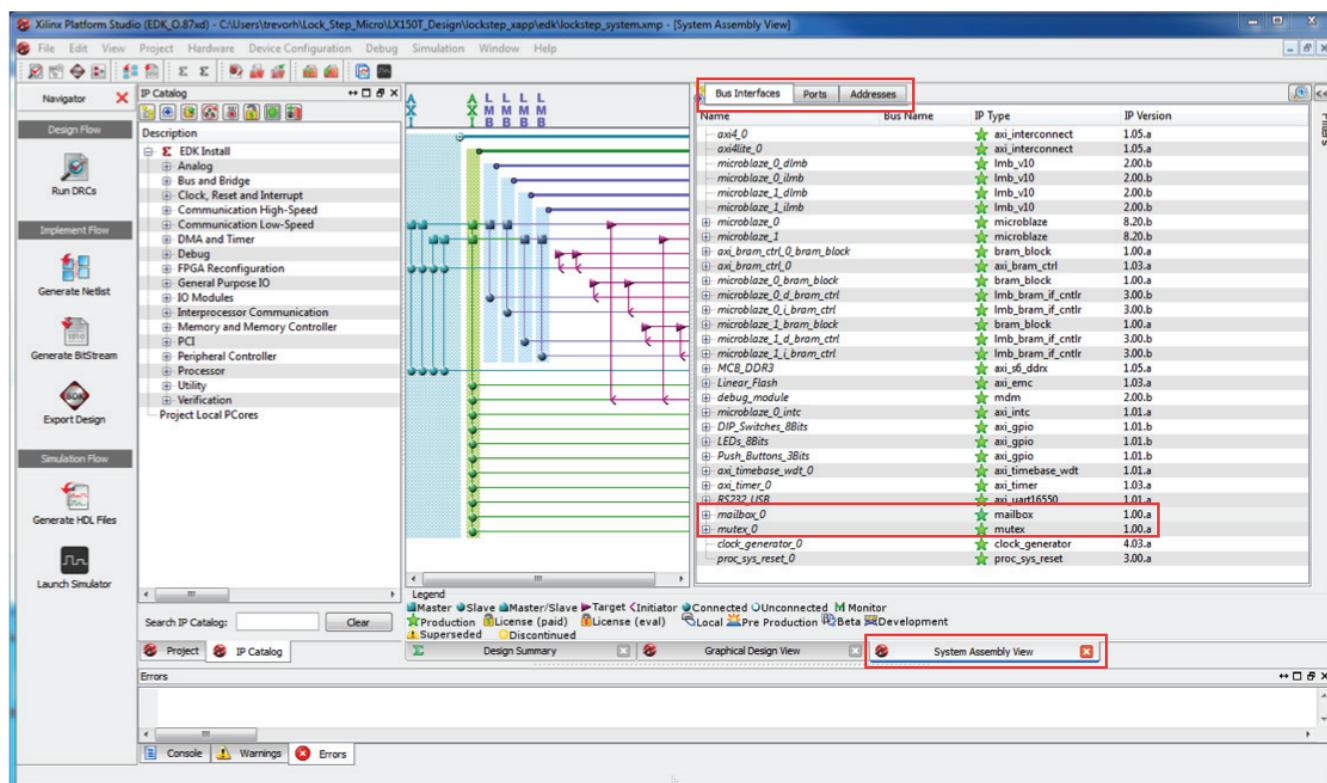
X584\_11\_041112

Figure 11: Reference Design Progress

### Removing Unused Cores from the System

Because the two MicroBlaze processors function in lockstep and not as separate processors that need to handshake with each other, the Mutex and Mailbox IPs included within the dual MicroBlaze processor system can be removed. These steps describe how to remove the two IPs (see [Figure 12](#)):

1. In the Platform Studio window under the System Assembly View Bus Interfaces tab, right-click the **mailbox\_0 IP** instance and then select **Delete Instance** in the pop-up menu.
2. At the Delete IP Instance prompt, select **Delete instance and all its connections**. Click **OK**.
3. Repeat [step 1](#) and [step 2](#) on the **mutex\_0 IP** instance.



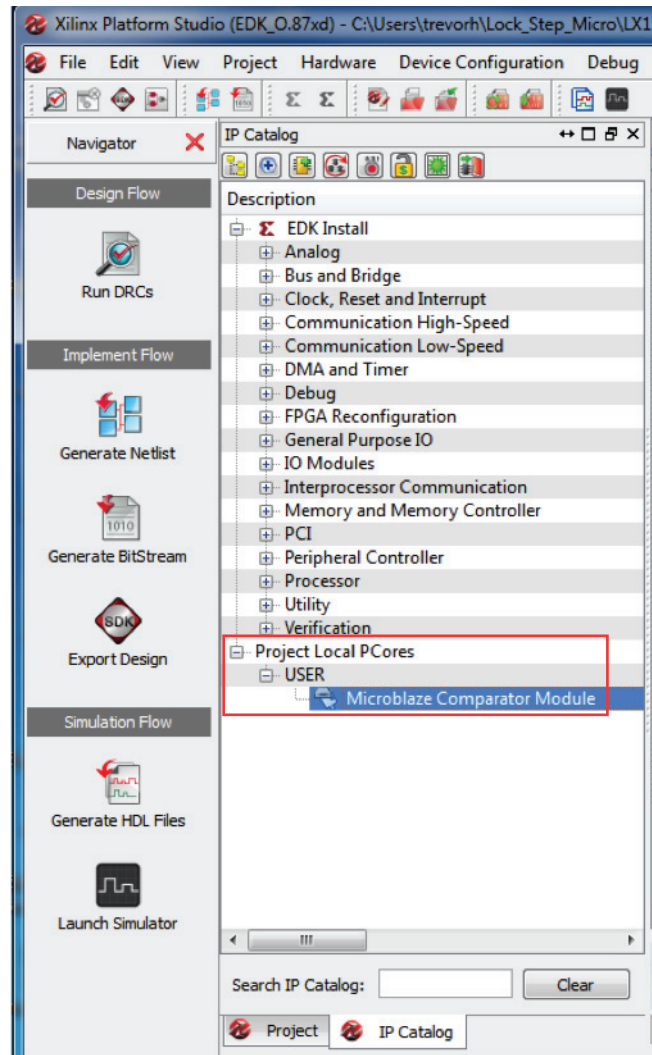
X584\_12\_040412

Figure 12: Mutex and Mailbox IPs in Platform Studio GUI

### Instantiating the MicroBlaze Comparator Cores IP

The dual-lockstep MicroBlaze processor system uses two redundant MicroBlaze Comparators to compare the outputs of each of the MicroBlaze processors. The comparator is provided by Xilinx to be instantiated as a local IP core. The files for the IP are provided as part of this reference design and are located in the <reference design>\external\_pcores directory. These steps describe how to include and connect the MicroBlaze Comparator cores into the system design:

1. Extract the `microblaze_comparator_v1_00_a.zip` file located at <reference design>\external\_pcores to <reference design>\edk\pcores so it can be found by the EDK Platform Studio tool.
2. In the Xilinx Platform Studio window, under the Project menu, select **Rescan User Repositories** to make Platform Studio find the MicroBlaze Comparator IP core.
3. With the MicroBlaze Comparator IP found by Platform Studio, in the Platform Studio IP Catalog pane, expand the **Project Local PCores > USER** tree to show the **MicroBlaze Comparator Module** (see Figure 13).

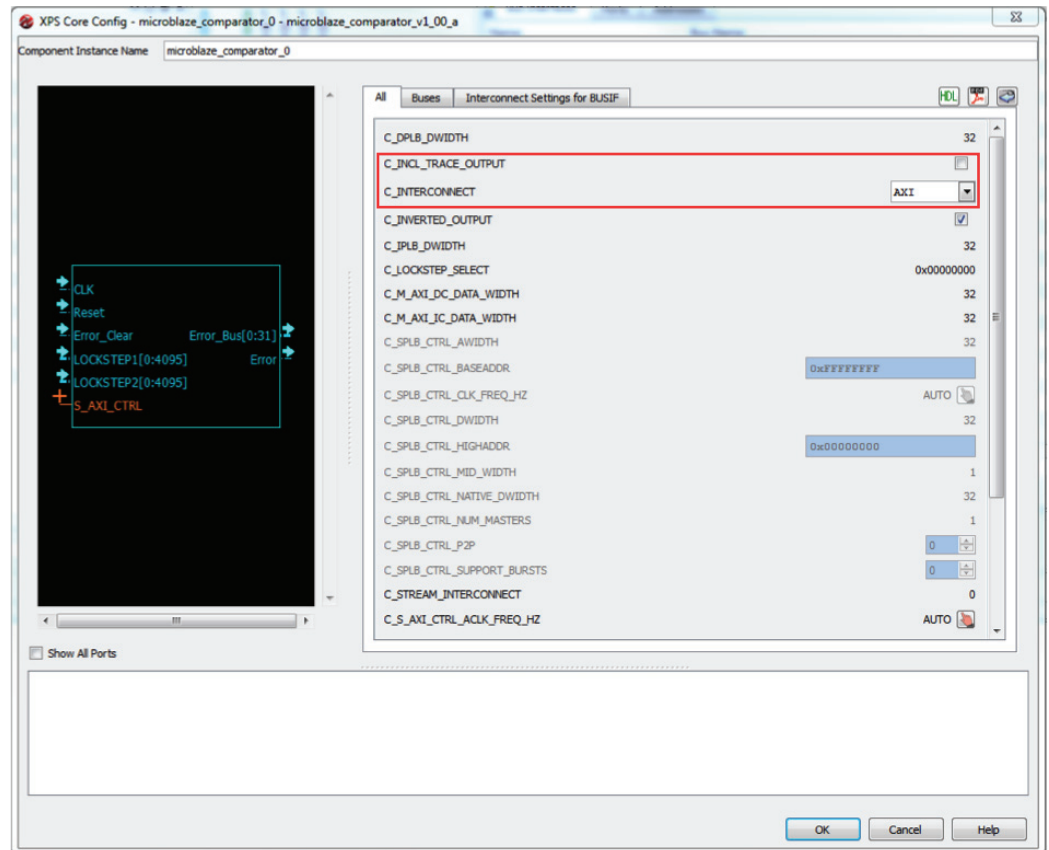


X584\_13\_040412

Figure 13: MicroBlaze Comparator Module in IP Catalog Pane

4. Right-click the **MicroBlaze Comparator Module** in the IP Catalog pane and click **Add IP** to add an instance of the comparator to the dual MicroBlaze processor system.
5. In the XPS Core Config window that appears (Figure 14), uncheck **C\_INCL\_TRACE\_OUTPUT** and set **C\_INTERCONNECT** to **AXI**.

**C\_INCL\_TRACE\_OUTPUT** is unchecked for this reference design because the design does not utilize the MicroBlaze processor Trace outputs. The MicroBlaze processor Trace outputs export a number of internal state signals for performance monitoring, analysis, and debugging. Including the Trace ports in the MicroBlaze comparator can be done in a design where the Trace port is not used to detect errors earlier before they reach the processor outputs. However, including the Trace ports increases the size of the comparator instance. For this reference design, it was decided to minimize the comparator logic footprint.



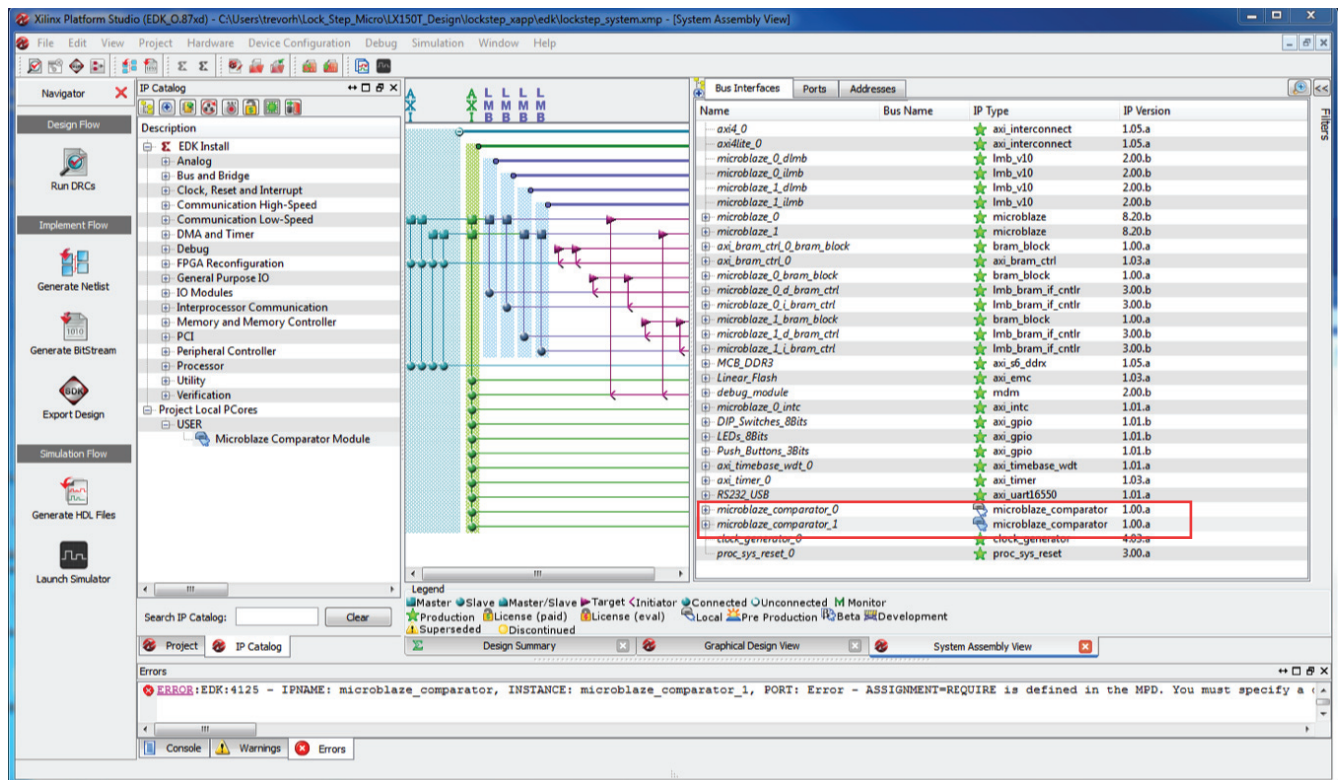
X584\_14\_040412

Figure 14: MicroBlaze Comparator IP Core Settings

6. Click **OK**.
7. In the Instantiate and Connect IP window that pops up, select **Select MicroBlaze instance to connect to...** and select **microblaze\_0**.
8. Click **OK**.
9. Repeat [step 4](#) through [step 8](#) to add a second instance of the MicroBlaze comparator core with the same configuration and MicroBlaze processor connection settings. The name of the second MicroBlaze comparator core is **microblaze\_comparator\_1**.

After the MicroBlaze comparators are added to the system, they appear in the Platform Studio System Assembly View, as shown in [Figure 15](#).

**Note:** Errors that appear in the Platform Studio transcript window are corrected in later steps in this application note.



X584\_15\_040412

Figure 15: MicroBlaze Comparator Cores Instantiated in the System

**Note:** At any time, the Platform Studio Project can be saved by selecting **Save Project...** under the File menu within Platform Studio.

## Manually Completing Port Connections

As part of the conversion of the dual MicroBlaze processor system to the dual-lockstep MicroBlaze processor system, interconnects need to be manually implemented in the system. This is done through the Bus Interfaces and Ports tab of the System Assembly View within Platform Studio. There are also some IP module configurations that need to be updated using the Platform Studio GUI. The steps in the following sections describe the manual connections and configuration changes that are required to transform to the dual-lockstep system.

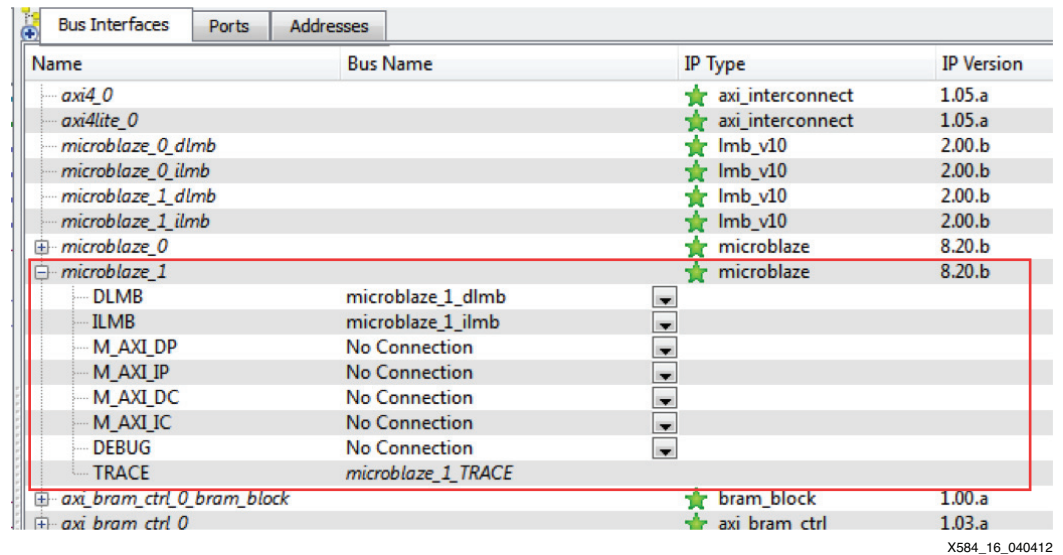
### Removal of microblaze\_1 AXI Bus Connections

In the dual MicroBlaze processor system, microblaze\_1 has its own AXI and MDM connections to function as an independent processor in the system. In the dual-lockstep MicroBlaze processor system, microblaze\_1 needs to see the same AXI inputs as microblaze\_0, and the MDM connection to microblaze\_1 needs to be completely removed. To start this process, the AXI and debug buses of microblaze\_1 are disconnected. This is done using these steps:

1. Within the Bus Interfaces tab of the System Assembly View, expand the bus list for microblaze\_1.
2. Click the down arrow in the Bus Name cell next to the M\_AXI\_DP bus and select **No Connection**.
3. Repeat [step 1](#) and [step 2](#) for the buses M\_AXI\_IP, M\_AXI\_DC, M\_AXI\_IC, and DEBUG.

The updated bus connections for microblaze\_1 are shown in [Figure 16](#).





Name	Bus Name	IP Type	IP Version
axi4_0		axi_interconnect	1.05.a
axi4lite_0		axi_interconnect	1.05.a
microblaze_0_dlmb		lmb_v10	2.00.b
microblaze_0_ilmb		lmb_v10	2.00.b
microblaze_1_dlmb		lmb_v10	2.00.b
microblaze_1_ilmb		lmb_v10	2.00.b
microblaze_0		microblaze	8.20.b
microblaze_1		microblaze	8.20.b
DLMB	microblaze_1_dlmb		
ILMB	microblaze_1_ilmb		
M_AXI_DP	No Connection		
M_AXI_IP	No Connection		
M_AXI_DC	No Connection		
M_AXI_IC	No Connection		
DEBUG	No Connection		
TRACE	microblaze_1_TRACE		
axi_bram_ctrl_0_bram_block		bram_block	1.00.a
axi_bram_ctrl_0		axi_bram_ctrl	1.03.a

X584\_16\_040412

Figure 16: microblaze\_1 Bus Connections

### microblaze\_0 Lockstep Port Connections

To support troubleshooting of the system, the Lockstep Master Output port of microblaze\_0 is connected to the Lockstep Slave port of microblaze\_1. This connection provides microblaze\_1 with all the control outputs that microblaze\_0 provides to the system so that microblaze\_1 can re-synchronize with microblaze\_0 if they get out of synchronization. Each MicroBlaze processor also has a lockstep output port that has all the processor outputs to connect to the MicroBlaze comparators for comparison. These steps describe connecting the lockstep and lockstep master outputs of microblaze\_0 (see Figure 17):

1. Within the Ports tab of the System Assembly View, expand the ports list for **microblaze\_0**.
2. Click the empty Connected Port cell next to the LOCKSTEP\_MASTER\_OUT port.
3. In the pop-up menu, select **microblaze\_1** on the left side and **LOCKSTEP\_SLAVE\_IN** on the right side to connect the microblaze\_0 LOCKSTEP\_MASTER\_OUT port to the microblaze\_1 LOCKSTEP\_SLAVE\_IN port.
4. Click anywhere outside of the pop-up menu to apply the connection. The new connection is made in the microblaze\_1 port definition automatically.
5. Click the empty Connected Port cell next to the LOCKSTEP\_OUT port.
6. In the pop-up menu, click the **New Connection** button to make two available connections.
7. In the first available connection for the LOCKSTEP\_OUT port, connect it to the microblaze\_comparator\_0 LOCKSTEP1 port by putting **microblaze\_comparator\_0** on the left side and **LOCKSTEP1** on the right side.
8. In the second available connection for the LOCKSTEP\_OUT port, connect it to the microblaze\_comparator\_1 LOCKSTEP1 port by putting **microblaze\_comparator\_1** on the left side and **LOCKSTEP1** on the right side.
9. Click anywhere outside of the pop-up menu to apply the connection. The new connection is automatically made in the microblaze\_comparator\_0 and microblaze\_comparator\_1 port definitions.

The updated port connections for microblaze\_0 are shown in Figure 17.

Name	Connected Port	Direction	Range	Class	Frequency(Hz)	Reset Polarity	Sensitivity	IP Type
External Ports								
axi4_0								axi_interconnect
axi4lite_0								axi_interconnect
microblaze_0_dlm								lmb_v10
microblaze_0_ilmb								lmb_v10
microblaze_1_dlm								lmb_v10
microblaze_1_ilmb								lmb_v10
microblaze_0								microblaze
MB_RESET	proc_sys_reset_0:MB_Reset	I		RST				
INTERRUPT	microblaze_0_intc:Irq	I		INTERRUPT			LEVEL_HIGH	
DBG_STOP		I						
MB_Halted		O						
MB_Error		O						
LOCKSTEP_MASTER_OUT	microblaze_1:LOCKSTEP_SLAVE_IN	O	[0:4095]					
LOCKSTEP_SLAVE_IN		I	[0:4095]					
LOCKSTEP_OUT	microblaze_comparator_0:LOCKSTEP1 microblaze_comparator_1:LOCKSTEP1	O	[0:4095]					
(BUS_IF) DLMB	Connected to BUS microblaze_0_dlm							
(BUS_IF) ILMB	Connected to BUS microblaze_0_ilmb							
(BUS_IF) M_AXI_DP	Connected to BUS axi4lite_0							
(BUS_IF) M_AXI_IP	Connected to BUS axi4lite_0							
(BUS_IF) M_AXI_DC	Connected to BUS axi4_0							
(BUS_IF) M_AXI_IC	Connected to BUS axi4_0							

Figure 17: microblaze\_0 Lockstep Port Connections

### microblaze\_1 Lockstep Port Connections

To connect the lockstep outputs of microblaze\_1 (Figure 18):

1. Within the Ports tab of the System Assembly View, expand the ports list for **microblaze\_1**.
2. Click the empty Connected Port cell next to the LOCKSTEP\_OUT port.
3. In the pop-up menu, click the **New Connection** button to make two available connections.
4. In the first available connection for the LOCKSTEP\_OUT port, connect it to the microblaze\_comparator\_0 LOCKSTEP2 port by putting **microblaze\_comparator\_0** on the left side and **LOCKSTEP2** on the right side.
5. In the second available connection for the LOCKSTEP\_OUT port, connect it to the microblaze\_comparator\_1 LOCKSTEP2 port by putting **microblaze\_comparator\_1** on the left side and **LOCKSTEP2** on the right side.
6. Click anywhere outside of the pop-up menu to apply the connection. The new connection is automatically made in the microblaze\_comparator\_0 and microblaze\_comparator\_1 port definitions.

The updated port connections for microblaze\_1 are shown in Figure 18.

Name	Connected Port	Direction	Range	Class	Frequency(Hz)	Reset Polarity	Sensitivity	IP Type
axi4lite_0								axi_interconnect
microblaze_0_dlm								lmb_v10
microblaze_0_ilmb								lmb_v10
microblaze_1_dlm								lmb_v10
microblaze_1_ilmb								lmb_v10
microblaze_1								microblaze
MB_RESET	proc_sys_reset_0:MB_Reset	I		RST				microblaze
INTERRUPT		I		INTERRUPT			LEVEL_HIGH	
DBG_STOP		I						
MB_Halted		O						
MB_Error		O						
LOCKSTEP_MASTER_OUT		O	[0:4095]					
LOCKSTEP_SLAVE_IN	microblaze_0:LOCKSTEP_MASTER_OUT	I	[0:4095]					
LOCKSTEP_OUT	microblaze_comparator_0:LOCKSTEP2 microblaze_comparator_1:LOCKSTEP2	O	[0:4095]					
(BUS_IF) DLMB	Connected to BUS microblaze_1_dlm							
(BUS_IF) ILMB	Connected to BUS microblaze_1_ilmb							
(BUS_IF) M_AXI_DP	Not connected to BUS or External Ports							
(BUS_IF) M_AXI_IP	Not connected to BUS or External Ports							
(BUS_IF) M_AXI_DC	Not connected to BUS or External Ports							
(BUS_IF) M_AXI_IC	Not connected to BUS or External Ports							

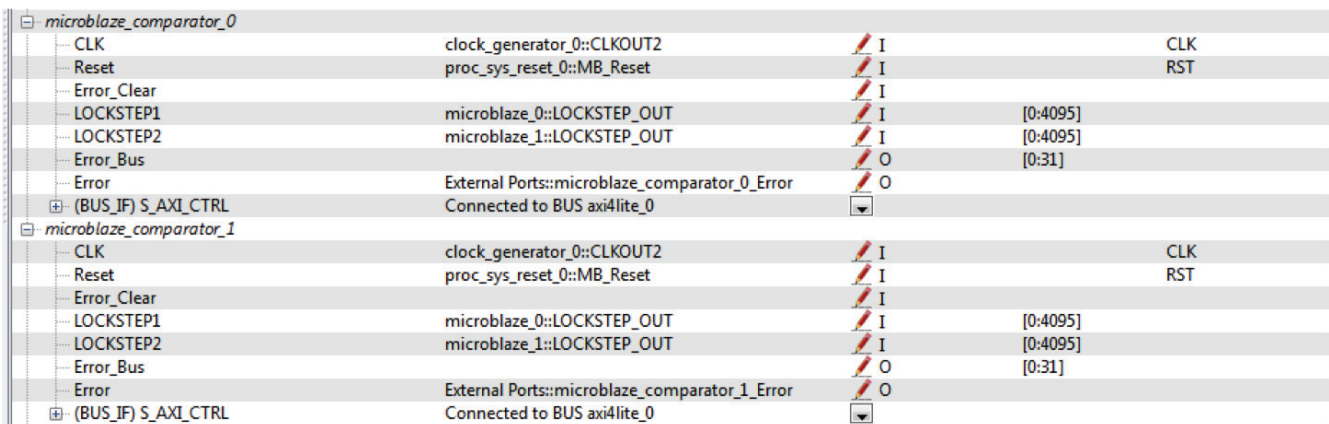
Figure 18: microblaze\_1 Lockstep Port Connections

### Comparator Port Connections

These steps describe how to connect the clock, reset, and error outputs of the MicroBlaze Comparators (see [Figure 19](#)):

1. Within the Ports tab of the System Assembly View, expand the ports list for **microblaze\_comparator\_0**.
2. Click the empty Connected Port cell next to the CLK port and connect it to the **CLKOUT2** port of **clock\_generator\_0**.
3. Click anywhere outside of the popup menu to apply the connection. The new connection connects the comparator clock to the 50 MHz clock used in the system.
4. Click the empty Connected Port cell next to the Reset port and connect it to the **MB\_Reset** port of **proc\_sys\_reset\_0**.
5. Click anywhere outside of the pop-up menu to apply the connection. The new connection connects the comparator reset to the reset used by the MicroBlaze processors.
6. Click the empty Connected Port cell next to the Error port. Select **External Ports** on the left and enter in **microblaze\_comparator\_0\_Error** on the right to route out the error signal to the system I/O with an output pin name of **microblaze\_comparator\_0\_Error**. This port is later connected to an LED.
7. Click anywhere outside of the pop-up menu to apply the connection.
8. Repeat [step 1](#) through [step 7](#) on **microblaze\_comparator\_1**. Note that [step 1](#) and [step 4](#) use the same port connections. For [step 6](#), the pin name should be **microblaze\_comparator\_1\_Error**.

The updated port connections for the comparators are shown in [Figure 19](#).



microblaze_comparator_0			
CLK	clock_generator_0::CLKOUT2	I	CLK
Reset	proc_sys_reset_0::MB_Reset	I	RST
Error_Clear		I	
LOCKSTEP1	microblaze_0::LOCKSTEP_OUT	I	[0:4095]
LOCKSTEP2	microblaze_1::LOCKSTEP_OUT	I	[0:4095]
Error_Bus		O	[0:31]
Error	External Ports::microblaze_comparator_0_Error	O	
(BUS_IF) S_AXI_CTRL	Connected to BUS axi4lite_0		
microblaze_comparator_1			
CLK	clock_generator_0::CLKOUT2	I	CLK
Reset	proc_sys_reset_0::MB_Reset	I	RST
Error_Clear		I	
LOCKSTEP1	microblaze_0::LOCKSTEP_OUT	I	[0:4095]
LOCKSTEP2	microblaze_1::LOCKSTEP_OUT	I	[0:4095]
Error_Bus		O	[0:31]
Error	External Ports::microblaze_comparator_1_Error	O	
(BUS_IF) S_AXI_CTRL	Connected to BUS axi4lite_0		

X584\_19\_040412

Figure 19: MicroBlaze Comparator Port Connections

### Modifying the LED Connections

With the addition of the MicroBlaze comparator Error ports to the system I/O, the existing LED connections that were generated by the Platform Studio BSB need to be modified to support connecting the new ports to the LEDs. The MicroBlaze comparators also need to have their configurations updated to invert the error output to interface to the LED so that it can be illuminated when an error is indicated. The LED needs an active-Low signal to turn on. These steps describe the modification:

1. Within the Bus Interfaces tab of the System Assembly View, right-click the **LEDs\_8Bits** instance of the **axi\_gpio** and select **Configure IP...**
2. In the XPS Core Config window, under the User tab, expand the **Channel 1** section and change the GPIO Data Channel Width setting to **6**.



3. Click **OK** to apply the change.
4. Within the Bus Interfaces tab of the System Assembly View, right-click the **microblaze\_comparator\_0** instance of the microblaze\_comparator and select **Configure IP...**
5. In the XPS Core Config window, check **C\_INVERTED\_OUTPUT**.
  - a. For microblaze\_comparator\_1 only, within the XPS Core Config window, set **C\_S\_AXI\_CTRL\_BASEADDR** to `0x7d220000` and **C\_S\_AXI\_CTRL\_HIGH\_ADDR** to `0x7d22ffff` to correct an address calculation error done by this version of Platform Studio.
6. Click **OK** to apply the change.
7. Repeat [step 4](#) through [step 6](#) for **microblaze\_comparator\_1**.
8. Click the Project tab located next to the callout for the IP Catalog tab on the bottom right of the Platform Studio window.
9. Under the Project Files tree, double-click the user constraints file (**UCF**) line to open the project's UCF.
10. Within the UCF, connect the MicroBlaze Comparator Error outputs to the removed LED ports by changing these lines from:

```
NET LEDs_8Bits_TRI_O[6] LOC = "C26" | IOSTANDARD = "LVCMOS33";
NET LEDs_8Bits_TRI_O[7] LOC = "F23" | IOSTANDARD = "LVCMOS33";
```

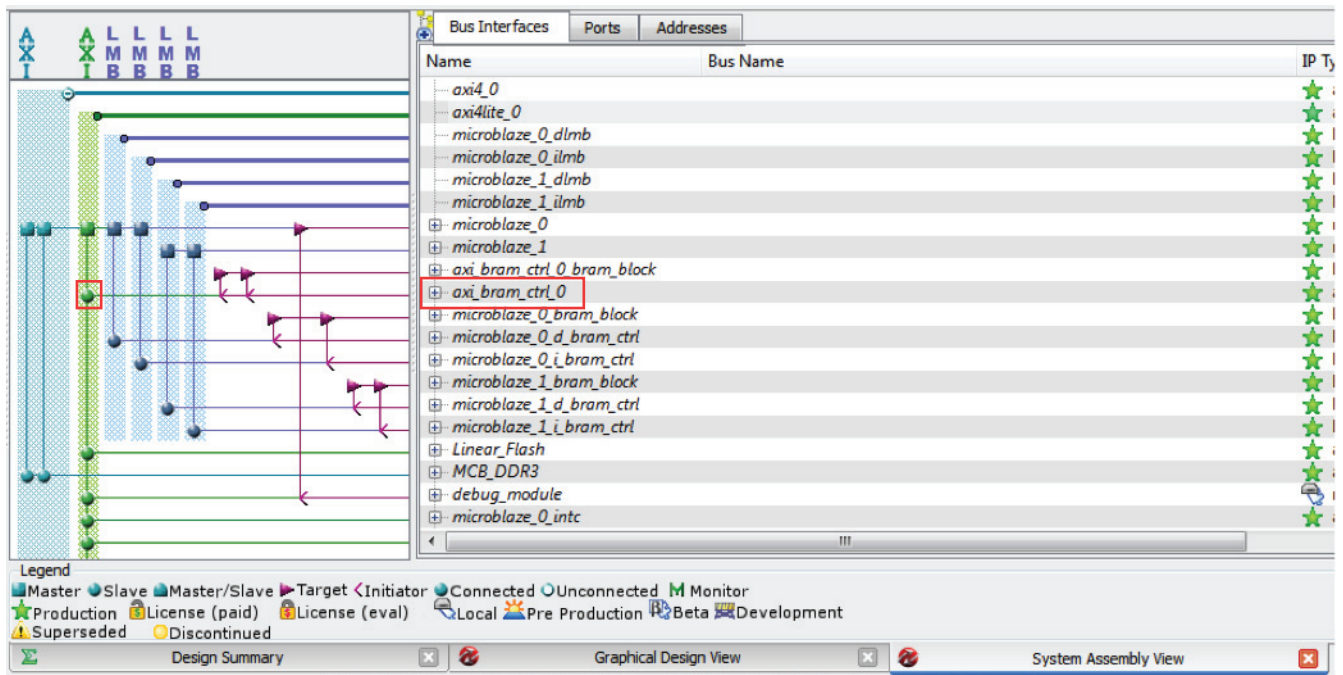
to:

```
NET microblaze_comparator_0_Error LOC = "C26" | IOSTANDARD = "LVCMOS33";
NET microblaze_comparator_1_Error LOC = "F23" | IOSTANDARD = "LVCMOS33";
```
11. Save the updated UCF by pressing **Ctrl+S**.

### ***Connecting the axi\_bram\_ctrl\_0 Instance to the AXI4-Lite Interconnect***

The Platform Studio BSB connected the axi\_bram\_ctrl\_0 instance to the AXI interconnect. For this reference design, axi\_bram\_ctrl\_0 should be connected to the AXI4-Lite interconnect. To do this, execute these steps (see [Figure 20](#)):

1. Within the Bus Interfaces tab of the System Assembly View, find the axi\_bram\_ctrl\_0 instance callout.
2. With the mouse, click the empty green circle to the left of the axi\_bram\_ctrl\_0 row in the green AXI column. This moves the connection from the AXI to the AXI4-Lite interconnect.



X584\_20\_040412

Figure 20: Connecting axi\_bram\_ctrl\_0 to AXI4-Lite Interconnect

### Modifying the MicroBlaze Debug Module

The MDM configuration needs to be modified to connect only to microblaze\_0 for the dual-lockstep MicroBlaze processor system. A step is also executed to make the MDM a local IP core to prepare the design to be isolated using the IDF.

1. Within the Bus Interfaces tab of the System Assembly View, right-click the **debug\_module** instance of the MDM and select **Make This IP Local**.
2. When prompted whether to continue making the IP local, click **OK**.

Making IP local places a copy of the necessary IP files into the pcores directory used by the project. By making the IP local, its source code can be modified to customize it for special design flows such as IDF. When the debug\_module instance has been made local by Platform Studio, the icon next to the instance changes from a star and is added to the Project Local Cores tree in the IP catalog pane.

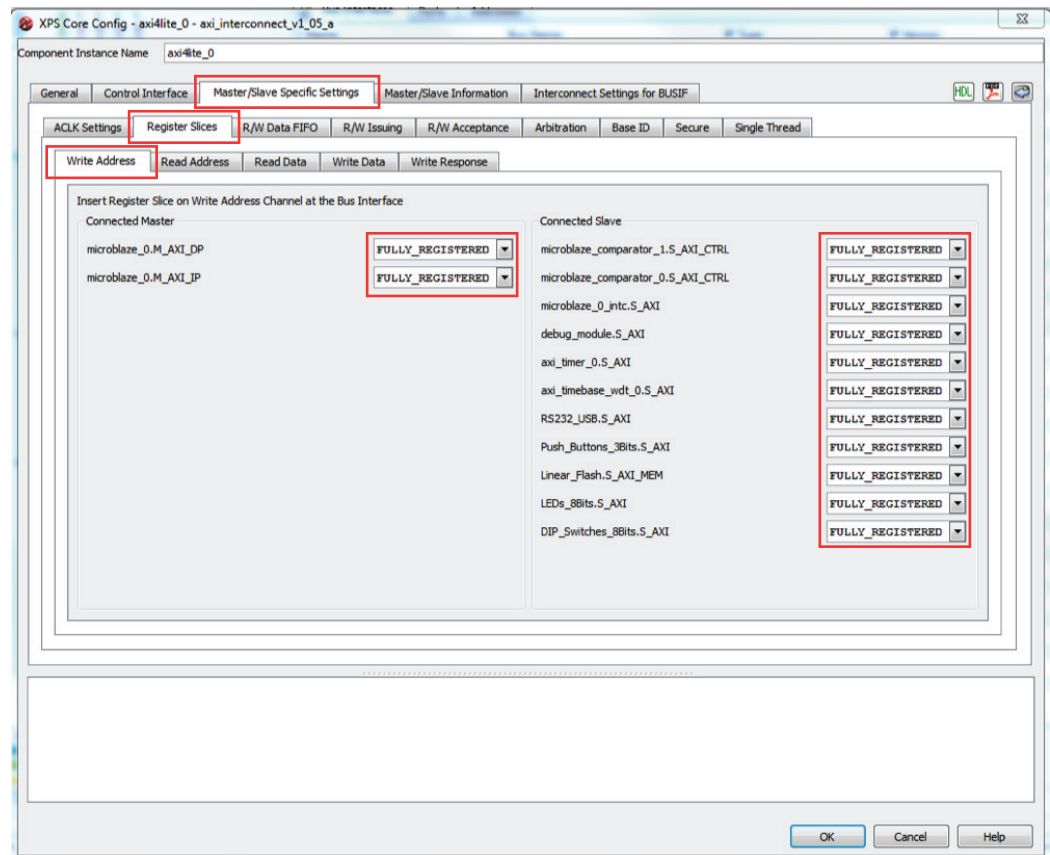
3. Begin to modify the debug\_module to support only one debug port by right-clicking the debug\_module instance in the Bus Interfaces tab of the System Assembly View, and select **Configure IP...**
4. In the XPS Core Config window, under the User tab, expand the Debug section and change the Number of MicroBlaze debug ports setting to **1**.
5. Click **OK** to apply the change.

### Modifying the AXI4-Lite Interconnect

To support better timing during design implementation, the axi4lite buses are fully registered for this reference design. These steps configure the axi4lite instance of the axi\_interconnect with registered signals:

1. Within the Bus Interfaces tab of the System Assembly View, right-click the **axi4lite\_0** instance of the axi\_interconnect and select **Configure IP...**
2. In the XPS Core Config window Master/Slave Specific Settings, Register Slices, and Write Address tabs for each Connected Master and Connected Slave, change the setting from

**BYPASS** to **FULLY\_REGISTERED** to add registers to the interconnect signals (see [Figure 21](#)).



X584\_21\_040412

Figure 21: AXI4-Lite Write Address Registering

3. Repeat [step 2](#) for each Connected Master and Connected Slave in the Read Address, Read Data, Write Data, and Write Response tabs.
4. Click **OK** to apply the changes.

### **Final System Modifications Done Manually to MHS File**

The final step in the conversion of the dual MicroBlaze processor system to the dual-lockstep MicroBlaze processor system is to modify the MHS file to add in parameters and port connections to the block RAM controllers and the MicroBlaze processors that could not be entered through the Platform Studio GUI. The updates are provided with the reference design in the `<reference design location>\file_mods\edk` directory. The steps in the following sections are provided to copy in the modifications to the active Platform Studio project MHS file.

#### **C\_MASK Parameter to microblaze\_1 Block RAM Controllers**

A `C_MASK` parameter has to be added to the `microblaze_1_i_bram_ctrl` and `microblaze_1_d_bram_ctrl` instances within the `lockstep_system.mhs` file. The parameter is required for Platform Studio to calculate the correct address decoding when building the dual-lockstep MicroBlaze processor system. The parameter is only required for the `microblaze_1` block RAM controllers because they are connected to the slave MicroBlaze processor, where automatic calculation of the parameter cannot be performed because the slave is isolated from the peripherals by only being connected to the AXI inputs.

1. With a text editor, open the files `<reference design>\edk\lockstep_system.mhs` and `<reference design>\file_mods\edk\mhs\bram_ctrl_mhs_mod.txt`.
2. Copy the parameter line located in the `bram_ctrl_mhs_mod.txt` file and paste it under the `PARAMETER C_HIGHADDR` line for both the `microblaze_1_i_bram_ctrl` and `microblaze_1_d_bram_ctrl` instances in the `lockstep_system.mhs` file.
3. Save the `lockstep_system.mhs` file.

The `C_MASK` value that is added to the `microblaze_1_i_bram_ctrl` and `microblaze_1_d_bram_ctrl` instances must match the calculated value for `microblaze_0_i_bram_ctrl` and `microblaze_0_d_bram_ctrl` or each of the MicroBlaze processors do not operate in lockstep. Because this application note lists the detailed steps to build the system, the value is not expected to differ from the value copied in the preceding section. If the MicroBlaze processors do not operate in lockstep, one thing to check is that the `C_MASK` values match for the `i_bram_ctrl` and `d_bram_ctrl` of the MicroBlaze processors. To do this, after generating the netlist in Platform Studio, the user can grep the `<reference design>\edk\hdl` directory for `C_MASK` and validate that the `C_MASK` generic setting provided in the `microblaze_0_d_bram_ctrl_wrapper.vhd`, `microblaze_0_i_bram_ctrl_wrapper.vhd`, `microblaze_1_d_bram_ctrl_wrapper.vhd`, and `microblaze_1_i_bram_ctrl_wrapper.vhd` files are all set to the same value. If the `C_MASK` settings do not match, the `microblaze_1_i_bram_ctrl` and `microblaze_1_d_bram_ctrl` instances `C_MASK` value in the `lockstep_system.mhs` file should be updated with the `C_MASK` generic value in the `microblaze_0_i_bram_ctrl_wrapper.vhd` file.

### microblaze\_1 Port and Parameter Updates

The `microblaze_1` port serves as the slave and checker in the dual-lockstep MicroBlaze processor system. It is the checker in that it does not drive the AXI interconnects directly, but it does execute the same instructions as `microblaze_0` and monitors AXI interconnect signals to ensure that it is getting the same responses as `microblaze_0`. The AXI interconnect signals that are inputs to `microblaze_0` need to be manually connected to `microblaze_1` in the MHS file so that both MicroBlaze instances see the same inputs from the AXI interconnects.

The interrupt signal that connects to `microblaze_1` needs to be the same signal that connects to `microblaze_0`, so a port connection is added to make the connection.

For this reference design, the dual-lockstep MicroBlaze processor does not allow the use of LUTRAMs, SRL16s, and SRL32s to prevent false SEU detection in Spartan-6 devices. A `C_AVOID_PRIMITIVES` parameter is added to prevent these primitives from being utilized by each MicroBlaze processor instance.

The `C_LOCKSTEP_SLAVE` parameter is also added to `microblaze_1` to differentiate it as the lockstep slave. This addition is done here because it is not automatically added by Platform Studio when the lockstep master port connection is made.

Finally, `C_INTERCONNECT_*_REGISTER` parameters are added to `microblaze_1` to give it the same registering that is applied to `microblaze_0` on the AXI4-Lite interconnect.

Adding these updates is done using these steps:

1. With a text editor, open the files `<reference design>\edk\lockstep_system.mhs` and `<reference design>\file_mods\edk\mhs\microblaze_1_mhs_mod.txt`.
2. Copy the parameter and port lines located in the `microblaze_1_mhs_mod.txt` file and paste it under the `BUS_INTERFACE DLMB` line for the `microblaze_1` instance in the `lockstep_system.mhs` file.
3. Save the `lockstep_system.mhs` file.

### microblaze\_0 Port and Parameter Updates

The AXI interconnect signals that are inputs to `microblaze_0` need to be redundantly called out in the `microblaze_0` instance as they were for `microblaze_1`. Also, as was done for

microblaze\_1, a C\_AVOID\_PRIMITIVES parameter is added to prevent LUTRAM and SRL primitives from being utilized by the MicroBlaze processor.

These are the steps to add the updates:

1. With a text editor, open the files `<reference design>\edk\lockstep_system.mhs` and `<reference design>\file_mods\edk\mhs\microblaze_0_mhs_mod.txt`.
2. Copy the parameter and port lines located in the `microblaze_0_mhs_mod.txt` file and paste it under the `BUS_INTERFACE DEBUG` line for the `microblaze_0` instance in the `lockstep_system.mhs` file.
3. Save the `lockstep_system.mhs` file.
4. Return to the Platform Studio window. When prompted to reload the project, click **Reload** to update the Platform Studio GUI with the updated MHS file.

## Building the Dual-Lockstep MicroBlaze Processor System in the EDK Platform Studio

With the dual MicroBlaze processor system converted to the *dual-lockstep* MicroBlaze processor system (Figure 22), the design is synthesized and a bitstream is generated to perform a design sanity check that all the updates are functioning correctly.

<input checked="" type="checkbox"/>	1. Generate Dual MicroBlaze System in EDK Platform Studio.
<input checked="" type="checkbox"/>	2. Modify Dual MicroBlaze System into Lockstep through EDK Platform Studio.
	3. Build the Dual-Lockstep MicroBlaze System in EDK Platform Studio.
	4. Perform a Quick Sanity Check of the Design.
	5. Prepare Dual-Lockstep MicroBlaze Design Source for Isolated Design.
	6. Synthesize and Floorplan Hierarchical Design.
	7. Run IVT on the Design in UCF Mode.
	8. Implement the Design.
	9. Run IVT on the Design in NCD Mode.
	10. Build Final Software in SDK.
	11. Disconnect the MicroBlaze Processors and Debug Logic.
	12. Re-verify the Re-implemented Design in IVT.

X584\_22\_041112

Figure 22: Reference Design Progress

To generate the bitstream:

1. Click the **Generate Bitstream** button on the right side of the Platform Studio window. This starts the Platgen function within Platform Studio to synthesize and implement the defined system.

**Note:** Generation of the bitstream takes some time.

A completed `lockstep_system.mhs` file that fully implements the dual-lockstep MicroBlaze processor system is archived in the `<reference design>\reference_files\completed_mhs` directory. This file can be copied into the `<reference design>\edk` directory to build the dual-lockstep MicroBlaze processor system after generating the dual MicroBlaze processor system. If this file is used, the user still needs to make the UCF change noted in [step 10 of Modifying the LED Connections, page 26](#).

## Performing a Quick Sanity Check of the Design

With the design netlist and bitstream built, SDK, included within EDK, is used to create a software project, import the provided source files, compile the provided source files, and generate an executable file to perform a quick sanity check to validate operation of the system on the Avnet Spartan-6 FPGA LX150T development board (Figure 23).

<input checked="" type="checkbox"/>	1. Generate Dual MicroBlaze System in EDK Platform Studio.
<input checked="" type="checkbox"/>	2. Modify Dual MicroBlaze System into Lockstep through EDK Platform Studio.
<input checked="" type="checkbox"/>	3. Build the Dual-Lockstep MicroBlaze System in EDK Platform Studio.
	4. Perform a Quick Sanity Check of the Design.
	5. Prepare Dual-Lockstep MicroBlaze Design Source for Isolated Design.
	6. Synthesize and Floorplan Hierarchical Design.
	7. Run IVT on the Design in UCF Mode.
	8. Implement the Design.
	9. Run IVT on the Design in NCD Mode.
	10. Build Final Software in SDK.
	11. Disconnect the MicroBlaze Processors and Debug Logic.
	12. Re-verify the Re-implemented Design in IVT.

X584\_23\_041112

Figure 23: Reference Design Progress

### Exporting the Files from Platform Studio to SDK

These steps describe exporting the Platform Studio design information to SDK, launching SDK, and creating a software project:

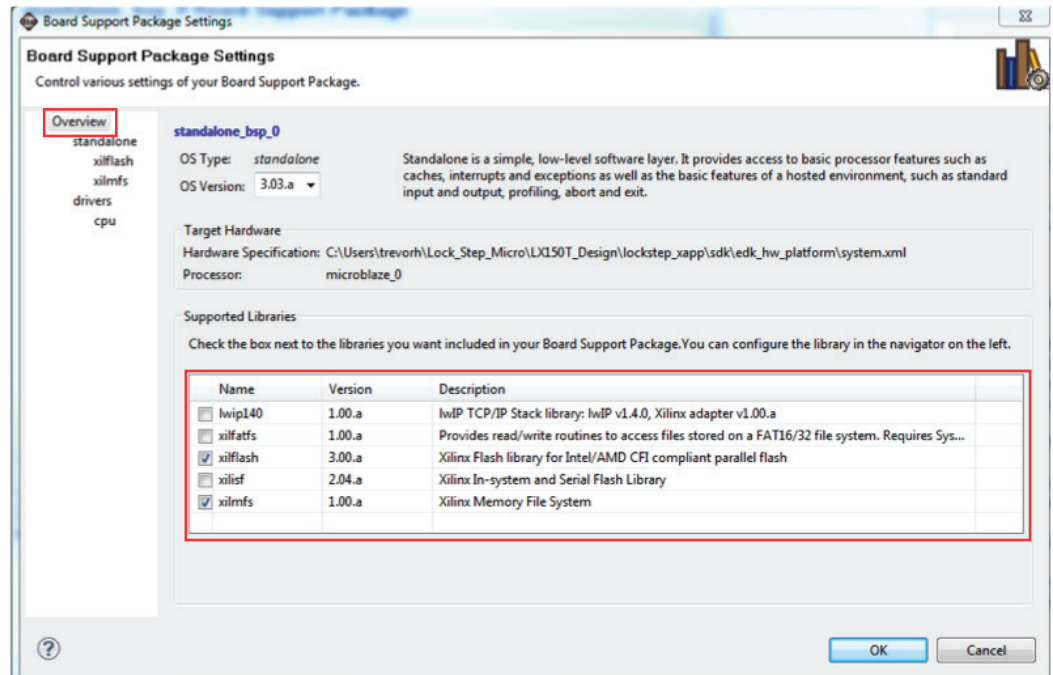
1. On the left side of the Platform Studio window, click the **Export Design** button.
2. In the Export to SDK/Launch SDK window, check **Include bitstream and BMM file** and then click **Export & Launch SDK** to export the design files to the `<reference design>\edk\SDK\SDK_Export\hw` directory and launch the SDK software.
3. Xilinx SDK launches and asks where to set up an SDK workspace to hold all the files for the SDK project. For this design, the SDK project is set up in `<reference design>\sdk`. In the Workspace Launcher window, set the workspace location to `<reference design>\sdk`. Click **OK**.

### Building the Board Support Package

These steps describe how to create a new standalone Board Support Package (BSP) for the dual-lockstep MicroBlaze processor system.

1. After SDK finishes loading, in the SDK window, select **File > New > Xilinx Board Support Package** to start setting up a BSP for the imported design.
2. In the New Board Support Package Project window, click **Finish** to build the new board support package with the default name `standalone_bsp_0` for the `edk_hw_platform` and `CPU_microblaze_0` of BSP OS type `standalone`. The hardware platform `edk_hw_platform` represents the system exported by Platform Studio.
3. In the Board Support Package Settings window (Figure 24), with **Overview** selected on the left, select **xilflash** and **xilmfs** under supported libraries to include the libraries in the BSP.

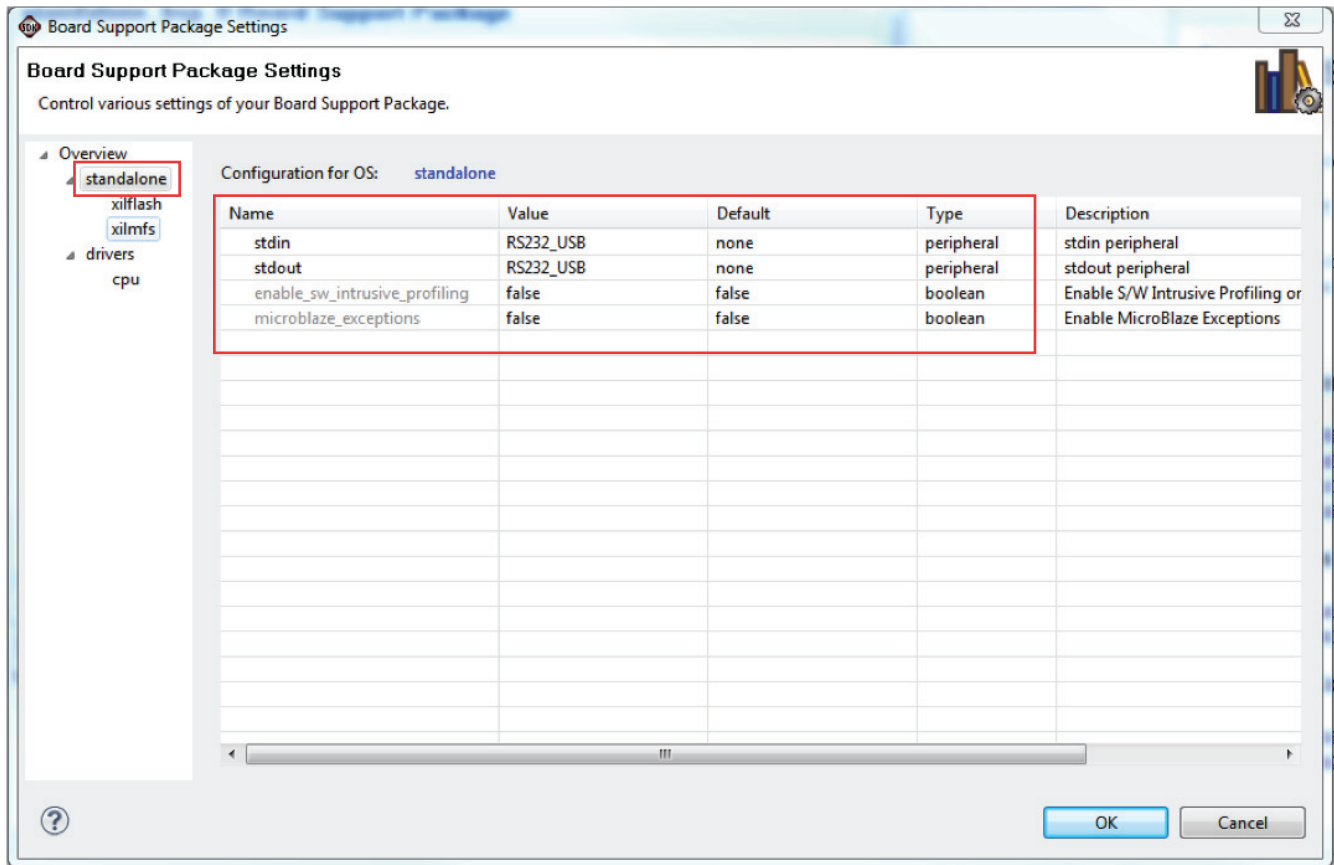




X584\_24\_040412

Figure 24: Selecting xilflash and xilmfs Libraries for Inclusion in the BSP

- In the Board Support Package Settings window (Figure 25), with **standalone** selected on the left, verify for **stdin** and **stdout** that the value is set to **RS232\_USB** to connect the RS-232 instance that is in the dual-lockstep MicroBlaze processor system as the standard-in and standard-out device. Click **OK**.



X584\_25\_040412

Figure 25: Verify stdin and stdout Settings

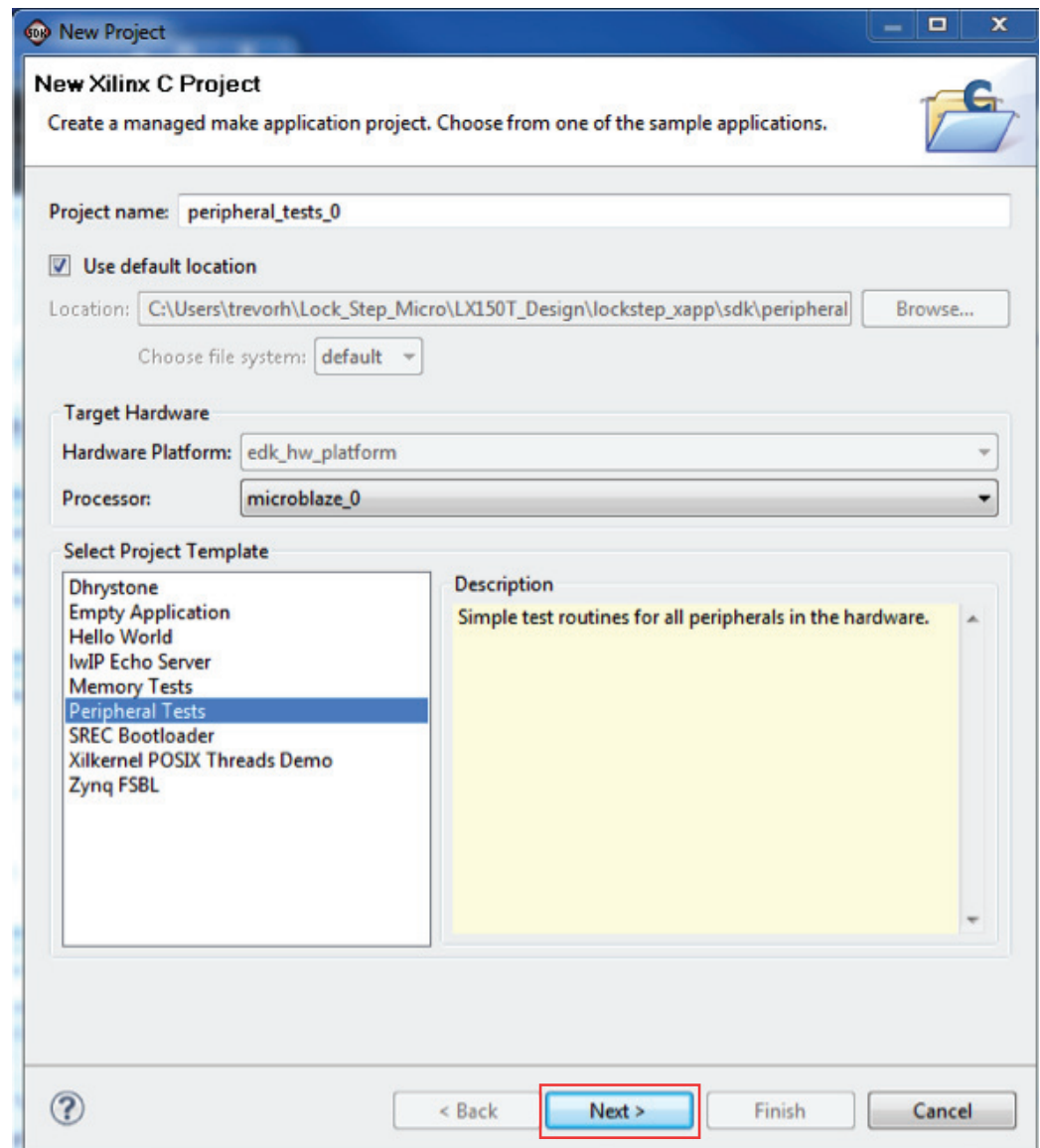
- After clicking **OK**, SDK compiles the BSP. After the compilation is done, verify that Finished building libraries can be seen in the SDK console, indicating the compilation was successful.

### Generating a Test Design to Test the Peripherals

SDK has a built-in ability to generate pre-packaged applications. One of these pre-packaged applications is a test of the peripherals of the embedded system. These steps describe linking, building, and running a pre-packed C application to test the embedded system's peripherals:

- In the SDK window, select **File > New > Xilinx C Project** to bring up the available pre-packaged applications.
- In the New Xilinx C Project window (Figure 26), make these settings to build a peripheral test application to run on the microblaze\_0 processor. Then click **Next >**.
  - Use Default Location: **checked**
  - Target Platform: **edk\_hw\_platform**
  - Processor: **microblaze\_0**
  - Select Project Template: **Peripherals Tests**





X584\_26\_040412

Figure 26: New Xilinx C Project Settings

3. In the next window, select **Target an existing Board Support Package** and click **Finish** to link this to the existing BSP. The application then compiles.
4. Verify that in the SDK console window it says `elf check passed. Finished building: peripheral_tests_0.elfcheck` at the end of the compilation.

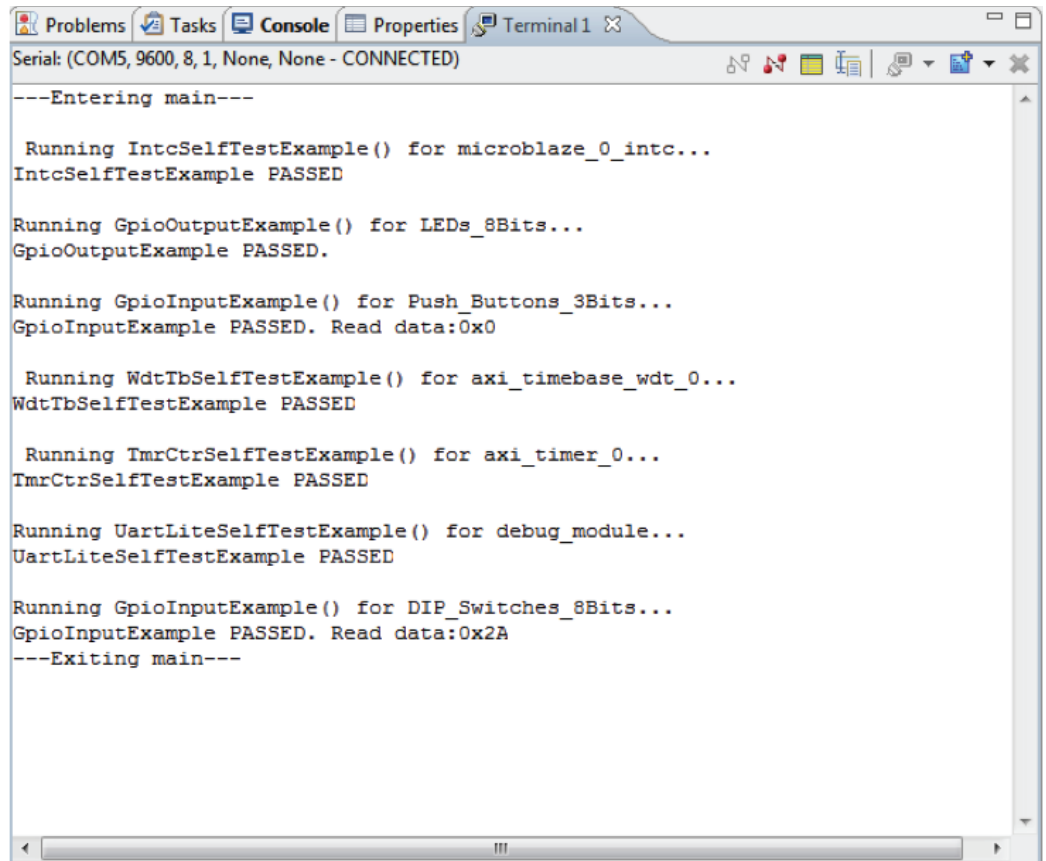
With the HW platform, BSP, and C Project all set up and compiled, the next step is to run the software on the board. The Avnet Spartan-6 FPGA LX150T development board connects to a computer through a USB to RS-232 converter located on the board to a USB port on the user's machine. Through this connection, the board can communicate through a terminal program such as HyperTerminal or the terminal embedded within the SDK. The driver software for the USB to RS-232 converter is provided with the Avnet Spartan-6 FPGA LX150T development board or is available for download on the Avnet site [Ref 5].

To run the Xilinx C project on the Avnet Spartan-6 FPGA LX150T development board:

1. Connect a Xilinx Platform Cable and a USB cable to the Avnet Spartan-6 FPGA LX150T development board and to the user computer. The Xilinx Platform Cable connects to J9 on

the board. There is only one USB port on the board, and it is located near the DB9 board-mounted socket.

2. Power on the Avnet Spartan-6 FPGA LX150T development board by setting SW11 to **ON**.
3. In the SDK window on the menu bar, select **Xilinx Tools > Program FPGA**.
4. In the Program FPGA window, the Bitstream and block RAM memory map (BMM) file locations point to the files exported by Platform Studio because it was selected to export the bitstream and BMM file in Platform Studio. The software configuration shows bootloop for both `microblaze_0` and `microblaze_1`. Click **Program** to program the FPGA through the Xilinx Platform Cable.
5. After the FPGA has programmed successfully, within the SDK window's Terminal pane, set up the terminal to have a serial connection with settings of **9600, 8, 1, none, and none**. Connect the terminal.
6. In the SDK window, right-click the `peripheral_tests_0` callout in the Project Explorer section and select **Run As > Launch on Hardware**.
7. Click **OK** in the Reset Status window.
8. The program now runs. In the Terminal (settings **9600, 8, 1, none, and none**), something like [Figure 27](#) appears where self-tests were run on each peripheral and they all passed.



```
Serial: (COM5, 9600, 8, 1, None, None - CONNECTED)
---Entering main---

Running IntcSelfTestExample() for microblaze_0_intc...
IntcSelfTestExample PASSED

Running GpioOutputExample() for LEDs_8Bits...
GpioOutputExample PASSED.

Running GpioInputExample() for Push_Buttons_3Bits...
GpioInputExample PASSED. Read data:0x0

Running WdtTbSelfTestExample() for axi_timebase_wdt_0...
WdtTbSelfTestExample PASSED

Running TmrCtrSelfTestExample() for axi_timer_0...
TmrCtrSelfTestExample PASSED

Running UartLiteSelfTestExample() for debug_module...
UartLiteSelfTestExample PASSED

Running GpioInputExample() for DIP_Switches_8Bits...
GpioInputExample PASSED. Read data:0x2A
---Exiting main---
```

X584\_27\_040412

Figure 27: C Project Peripherals Test Terminal Output

If desired, a memory test is also available in the pre-packaged C projects and can be executed similarly to the peripherals test.

9. When done running the applications in SDK, close SDK by selecting **File > Exit**.

## Preparing the Dual-Lockstep MicroBlaze Processor System for Isolated Design

Figure 28 shows the reference design progress to this point.

<input checked="" type="checkbox"/>	1. Generate Dual MicroBlaze System in EDK Platform Studio.
<input checked="" type="checkbox"/>	2. Modify Dual MicroBlaze System into Lockstep through EDK Platform Studio.
<input checked="" type="checkbox"/>	3. Build the Dual-Lockstep MicroBlaze System in EDK Platform Studio.
<input checked="" type="checkbox"/>	4. Perform a Quick Sanity Check of the Design.
	5. Prepare Dual-Lockstep MicroBlaze Design Source for Isolated Design.
	6. Synthesize and Floorplan Hierarchical Design.
	7. Run IVT on the Design in UCF Mode.
	8. Implement the Design.
	9. Run IVT on the Design in NCD Mode.
	10. Build Final Software in SDK.
	11. Disconnect the MicroBlaze Processors and Debug Logic.
	12. Re-verify the Re-implemented Design in IVT.

X584\_28\_041112

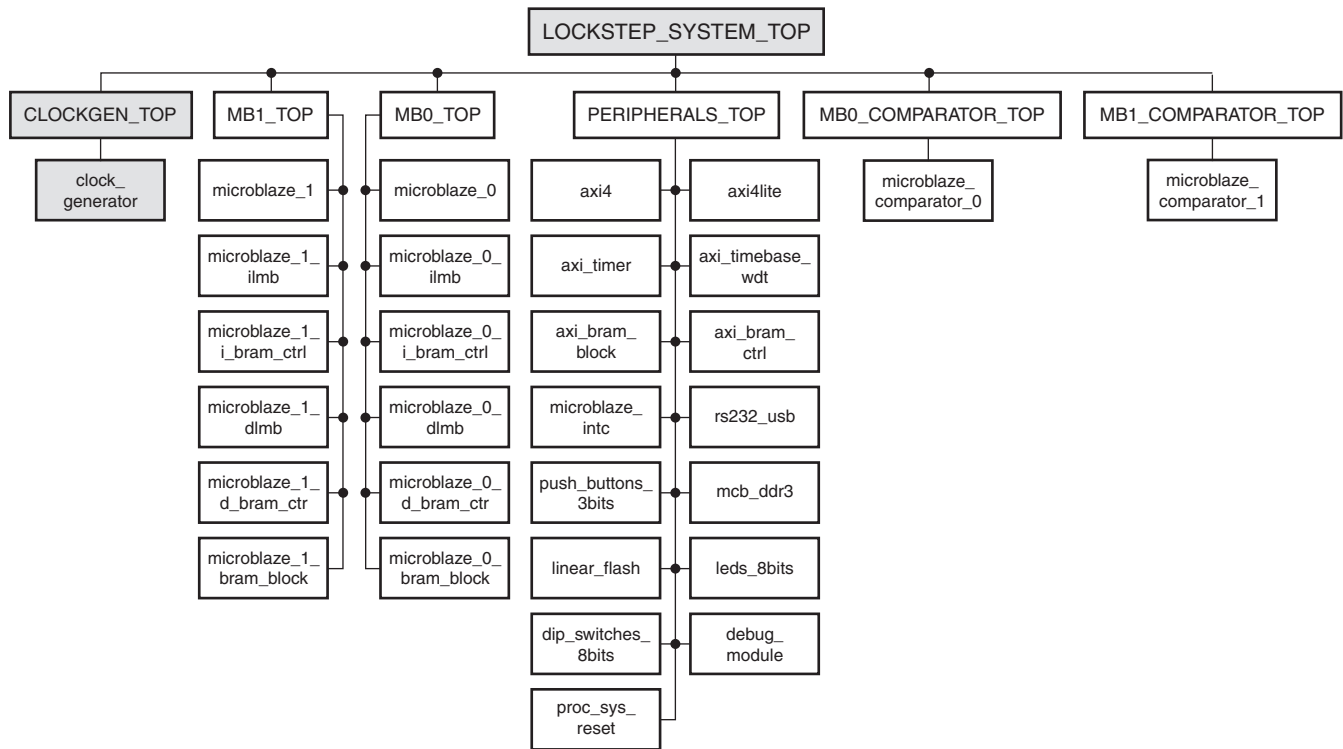
Figure 28: Reference Design Progress

In this application note, the IDF is applied to the dual-lockstep MicroBlaze processor to partition and floorplan the design into isolated functions. IDF requires that the design be hierarchical to identify the isolated functions. When the EDK Platform Studio generates a design, it is effectively flat, where all the design components are at the same level of the design hierarchy. The next sections describe what was done to change the flat design generated by Platform Studio into a hierarchical design that follows the rules and considerations for IDF. The finished hierarchical HDL files are in the `<reference design>\src\hdl` directory.

The design has the hierarchy illustrated in Figure 29. For the design, there are five isolated functions:

- MB0\_TOP
- MB1\_TOP
- PERIPHERALS\_TOP
- MB0\_COMPARATOR\_TOP
- MB1\_COMPARATOR\_TOP

The clock generator is also included in its own hierarchical element as good coding practice for readability. This is allowed in IDF because the clock generator is not implemented as its own isolated function.



X584\_29\_041112

Figure 29: Dual-Lockstep MicroBlaze Processor System Hierarchy

These changes require that the contents of the BMM and UCF files be updated to account for the new hierarchy. The updated files are provided in the `<reference design>\src\bmm` and `<reference design>\src\ucf` directories and are used in the implementation and floorplanning steps.

The use of IDF also requires a change to the EDK files that generate the MDM. Steps for the change are provided in the next section.

### Modifying the MicroBlaze Debug Module Generation Files

The files used to generate the MDM in EDK need to be modified to bring out a global clock buffer instantiation used to route the DRCK clock. For global clocks to be used in an isolated design, they must be instantiated at the design's top level or within a module that is not part of an isolated function. If the BUFG instantiation remains within the MDM, which is included within the PERIPHERALS\_TOP isolated function, global routing of the clock signal can be prevented. For this reference design, the DRCK BUFG instance is removed from the MDM and instantiated within the `LOCKSTEP_SYSTEM_TOP.vhd` file, the design's top.

The steps in this section describe the modifications required so that the EDK Platform Studio does not instantiate the BUFG within the MDM, but instead routes out the BUFG connections to the I/O of the MDM. The system's netlist is then rebuilt to incorporate the changes.

1. In the Platform Studio window, in the System Assembly View Bus Interfaces tab, right-click the **debug\_module** instance and select **Browse HDL Sources...**
2. In the file selection window, open the `mdm.vhd` file.
3. With the file open, make these modifications to the `mdm.vhd` file:
  - a. Add these two ports to the MDM entity port declarations to route out the connections to the DRCK BUFG:
 

```
drck: in std_logic; --DRCK BUFG output
```

`drck_i`: **out std\_logic**; --DRCK BUFG input

- b. At lines 527 and 534 of the `mdm.vhd` file, comment out the signal declarations for the `drck` and `drck_i` signals because they are now part of the entity port declarations.
  - c. At lines 743 to 747 of the `mdm.vhd` file, comment out the `BUFG_DRCK1` BUFG instance because it is not used for Spartan-6 device implementations.
  - d. At line 749, uncomment the signal assignment `drck1 <= drck1_i` to route through the assignment to `drck1`.
  - e. At lines 751 to 755 of the `mdm.vhd` file, comment out the `BUFG_DRCK` BUFG instance to remove it from the MDM and allow it to be instantiated at the `LOCKSTEP_SYSTEM_TOP` HDL file.
4. Save the updated `mdm.vhd` file by clicking **File > Save** in the Platform Studio menu bar.
  5. In the Platform Studio window, in the System Assembly View Bus Interfaces tab, right-click the **debug\_module** instance and select **View MPD**.
  6. With the addition of the `drck` input and `drck_i` output to the `mdm` module ports, the MPD file also needs to be updated so that Platform Studio knows of the updated module description. At line 105 of the `mdm_v2_1_0.mpd` file, below the port instance of `Debug_SYS_reset`, add these two lines to indicate the two new ports:
 

```
PORT drck = "", DIR = I
PORT drck_i = "", DIR = O
```
  7. Save the updated `mdm_v2_1_0.mpd` file by clicking **File > Save** in the Platform Studio menu bar.
  8. To apply the updated files to the `lockstep_system` project, rescan the user repositories within Platform Studio by selecting **Project > Rescan User Repositories** from the menu bar.
  9. Verify that the `drck` and `drck_i` ports are now applied to the project by going to the System Assembly View Ports tab and expanding the ports for the `debug_module`. The new ports `drck` and `drck_i` should be shown in the tab.
  10. Clean the `lockstep` system netlist and the implementation files by clicking **Project > Clean All Generated Files from the Platform Studio Menu** toolbar. Select **Yes** when prompted `Are you sure you want to delete all generated files?`
  11. After the netlist files have been deleted, regenerate the netlist to generate the system NGC and VHD files by clicking **Hardware > Generate Netlist**.
  12. The netlist generation takes some time. After the netlist generation completes, close the Platform Studio application.

## Comparing Flat Files between Designs

The `_TOP.vhd` files that define the hierarchical system and are located at `<reference design>\src\hdl` were derived from the `lockstep_system_refgen.vhd` file located within this same directory. The `lockstep_system_refgen.vhd` file is a copy of the `lockstep_system.vhd` file that was generated by the EDK Platform Studio and placed in the `<reference design>\edk\hdl` directory when the system was generated by EDK as part of writing this application note. The next sections describe how the hierarchical top files were generated from this `lockstep_system_refgen.vhd` file. Now the user should compare the file EDK generated for the user's system (located at `<reference design>\edk\hdl\lockstep_system.vhd`) with `lockstep_system_refgen.vhd`. If the files differ, the user might need to make minor modifications to the provided top files.

All the `_TOP.vhd` files capitalize the input and output ports and have a suffix of either `_IN` or `_OUT` to define the port direction. Input and output bus names also have the bit numbers included in the name to identify the portion of the larger bus they consume.

For example, `AXI4LITE_0_S_RID_0_IN` is an input that is bit 0 of the `axi4lite_0_s_rid` bus defined within the original `lockstep_system_refgen.vhd` file. Signal names that end in

*\_int* are internal signals that were added to the design as part of the hierarchical development to connect up buffers and registers. All other signal names remain unchanged from the `lockstep_system_refgen.vhd` file. This coding standard was utilized to make the code more readable and traceable given the large number of signals required to describe the system.

### Generating the CLOCKGEN\_TOP HDL File

The `CLOCKGEN_TOP.vhd` file instantiates the `clock_generator_0_wrapper` implemented by the EDK Platform Studio. The file was developed to receive the reset from the `proc_sys_reset` component instantiated in the `PERIPHERALS_TOP` isolated function and fanout the clocks and the `DCM_LOCKED` output to the dual-lockstep MicroBlaze processor system. The `clock_generator_0_wrapper` takes in the external 100 MHz clock, so the `EXTERN_CLK_IN_IBUF` has been added. The `black_box` attribute is applied to the `clock_generator_0_wrapper` so that the module's netlist is pulled from files implemented by Platform Studio.

### Generating the MB0\_TOP HDL File

The `MB0_TOP.vhd` file instantiates the `microblaze_0_wrapper`, `microblaze_0_ilmb_wrapper`, `microblaze_0_i_bram_ctrl_wrapper`, `microblaze_0_dlmb_wrapper`, `microblaze_0_d_bram_ctrl_wrapper`, and `microblaze_0_bram_block` implemented by Platform Studio.

The file implements I/O for these system connections:

- Receive the 50 MHz clock input from the `CLOCKGEN_TOP` module
- Receive the MicroBlaze and bus structure Resets
- Receive the MicroBlaze interrupt
- Connect to the MicroBlaze Debug Module's Debug bus
- Output the lockstep buses to `MB1_TOP` and each of the comparators
- Connect to the AXI and AXI4-Lite interconnect buses

The `black_box` attribute is applied to all the wrapper components so that the modules' netlists are pulled from the files implemented by Platform Studio.

An `equivalent_register_removal` attribute is applied to the fanout buses of the `lockstep_out` output. The `lockstep_out` output is fanned out to both of the MicroBlaze Comparators. In accordance with IDF rules and considerations for a single signal driving two different output ports of the same module, `lockstep_out` must be fanned out using active buffers. The `equivalent_register_removal` attribute is set to **no** to prevent the equivalent registers implemented in the `P_LOCKSTEP_FANOUT` process from being consolidated into one register set.

The lower 43 bits of the `LOCKSTEP_MASTER` output bus and various AXI and AXI4-Lite outputs not only drive loads located in other isolated functions of the design, but they also drive loads internally to the `MB0_TOP` isolated function. In accordance with IDF rules and considerations for signals that drive multiple isolated functions, LUT buffers have been added to the outputs, leaving the `MB0_TOP` isolated function. These LUT buffers prevent a failure in the external isolated function from affecting the `MB0_TOP` isolated function. Signals that drive multiple isolated functions, where one load was within the `MB0_TOP` isolated function and another load was in another isolated function, were indicated as isolation errors by running the IVT in native circuit description (NCD) mode on the design. These errors were then corrected and verified by re-implementing and re-running IVT. The errors were not directly determined by just reading the HDL code.

### Generating the MB1\_TOP HDL File

The `MB1_TOP.vhd` file instantiates the `microblaze_1_wrapper`, `microblaze_1_ilmb_wrapper`, `microblaze_1_i_bram_ctrl_wrapper`, `microblaze_1_dlmb_wrapper`,

microblaze\_1\_d\_bram\_ctrl\_wrapper, and microblaze\_1\_bram\_block implemented by Platform Studio.

The MB1\_TOP.vhd file, similar to MB0\_TOP, implements I/O for these system connections:

- Receive the 50 MHz clock input from the CLOCKGEN\_TOP module.
- Receive the MicroBlaze and bus structure Resets.
- Receive the MicroBlaze interrupt.
- Connect to the MicroBlaze Debug External Break inputs.
- Receive the Lockstep Master bus from MB1\_TOP.
- Output the Lockstep buses to each of the comparators.
- Connect to the AXI and AXI4-Lite interconnect buses.

Like all the other hierarchical components, the black\_box attribute is applied to all the wrapper components.

As was done for MB0\_TOP, and for the same reason, the lockstep\_output is fanned out using active buffers and has the equivalent\_register\_removal attribute applied and set to **no** for the fanout buses.

### Generating the MB0\_COMPARATOR\_TOP HDL File

The MB0\_COMPARATOR\_TOP.vhd file instantiates the microblaze\_comparator\_0\_wrapper implemented by Platform Studio.

The file implements I/O for these system connections:

- Receive the 50 MHz clock input from the CLOCKGEN\_TOP module.
- Receive the MicroBlaze processor reset.
- Receive the lockstep buses from each of the MicroBlaze processors.
- Output the comparator error signal to PERIPHERALS\_TOP for buffering and final output.
- Connect to the AXI4-Lite interconnect buses.

Like all the other hierarchical components, the black\_box attribute is applied to all the wrapper components.

The AXI4-Lite outputs not only drive loads located in other isolated functions of the design, but they also drive loads internally to the MB0\_COMPARATOR\_TOP isolated function. In accordance with IDF rules and considerations for signals that drive multiple isolated functions, LUT buffers have been added to the outputs leaving the MB0\_COMPARATOR\_TOP isolated function. These LUT buffers prevent a failure in the external isolated function from affecting the MB0\_COMPARATOR\_TOP isolated function.

**Note:** The signals that drive multiple isolated functions were indicated as isolation errors by running the IVT in NCD mode on the design. These errors were then corrected and verified by re-implementing and re-running the IVT. The errors were not directly determined by just reading the HDL code.

### Generating the MB1\_COMPARATOR\_TOP HDL File

The MB1\_COMPARATOR\_TOP.vhd file instantiates the microblaze\_comparator\_1\_wrapper implemented by Platform Studio. The highlights of generating the file are the same as MB0\_COMPARATOR\_TOP.vhd.

### Generating the PERIPHERALS\_TOP HDL File

The PERIPHERALS\_TOP.vhd file instantiates the AXI and AXI4-Lite interconnect buses. It also instantiates all the AXI peripherals, reset module, interrupt controller, and MDM.

The file implements I/O for these system connections:

- Receive the 50 MHz clock input from the CLOCKGEN\_TOP module.



- Receive the DDR3 clock inputs for the CLOCKGEN\_TOP module.
- Receive the DCM Locked input to feed the reset module.
- Output the MicroBlaze processor resets to both MicroBlaze processors and MicroBlaze comparators.
- Output the MicroBlaze interrupts to both MicroBlaze processors.
- Output the external breaks to both MicroBlaze processors.
- Receive the MicroBlaze Comparator error outputs, and then buffer and output the signal to the pins.
- Connect the MDM DRCK clock to the DRCK clock BUFG that was removed from the debug module generation files as discussed in [Modifying the MicroBlaze Debug Module Generation Files, page 38](#).
- Provide the AXI and AXI4-Lite interconnect signals to connect the MicroBlaze processors and MicroBlaze Comparators to the interconnect buses.
- Connect the MDM to MB0\_TOP.
- Provide external pin connections to LOCKSTEP\_SYSTEM\_TOP.

Like all the other hierarchical components, the `black_box` attribute is applied to all the wrapper components.

An `equivalent_register_removal` attribute is applied to the fanout buses of the MicroBlaze processor interrupt output. The interrupt output is fanned out to both of the MicroBlaze processors, and in accordance with IDF rules and considerations for a single signal driving two different output ports of the same module, must be fanned out using active buffers. The `equivalent_register_removal` attribute is set to `no` to prevent the equivalent registers implemented in the register process from being consolidated into one register set. A dual register set is used to help with cross clock domain signal generation.

Iobufs, Ibufs, and Obufs are instantiated within the PERIPHERALS\_TOP isolated function so that the I/O buffer is logically and physically owned by the isolated function in accordance with IDF. Directly instantiating the BUFs allows for the bottom-up synthesis to be accomplished within a single PlanAhead or ISE tool project. If the BUFs were not directly instantiated, the PlanAhead and ISE tools would instantiate the buffers at the top level, which would be against the rules of IDF. The I/O buffers for the DDR3 SDRAM were not instantiated in the PERIPHERALS\_TOP.vhd file because they were directly instantiated by Platform Studio in the mcb\_ddr3\_wrapper netlist.

**Note:** The user can tell which I/O buffers are owned by which hierarchical block by expanding the hierarchical blocks netlist in the PlanAhead tool Netlist window, and then expanding the Primitives tree. The OBUFs, IBUFs, and IOBUFs are listed in the primitives. The primitive type is listed in parentheses next to the primitive name.

Each of the MicroBlaze Comparators drive error outputs that go to PERIPHERALS\_TOP, where they are buffered with a LUT buffer to provide isolation buffering between the isolated functions. They are then forwarded to OBUFs for output. The MicroBlaze Comparator error outputs were handled this way to allow for easier floorplanning of the design, given the Avnet Spartan-6 FPGA LX150T development board pinout. Keeping ownership of the comparator error output buffers would have made the design unroutable due to the need to route the AXI interconnect throughout the chip.

The external reset fanout to the clock generator, along with the fanout of the MicroBlaze processor reset, the common AXI and AXI4-Lite outputs, and the JTAG signals that make up the MicroBlaze processor debug bus are all LUT buffered in accordance with IDF rules and considerations for signals with loads in two different isolated functions. As noted in [Generating the MB0\\_TOP HDL File, page 40](#) and [Generating the MB0\\_COMPARATOR\\_TOP HDL File, page 41](#), some of the signals that drive multiple isolated functions were indicated as isolation errors by the IVT running in NCD mode. These errors were then corrected and verified by



re-implementing and rerunning the IVT. The errors were not directly determined by just reading the HDL code.

### Generating the LOCKSTEP\_SYSTEM\_TOP HDL File

The `LOCKSTEP_SYSTEM_TOP.vhd` serves as the overall design top file, and instantiates the `CLOCKGEN_TOP`, `MB0_TOP`, `MB1_TOP`, `PERIPHERALS_TOP`, `MB0_COMPARATOR_TOP`, `MB1_COMPARATOR_TOP`, and the `BUFG` instance for the debug module's `DRCK`. The entity I/O names match the names generated by Platform Studio in the `lockstep_system_refgen.vhd` file to minimize changes to the UCF, which is also generated by Platform Studio. The `BUFG` instance is instantiated within this file so that it is logically owned at the top level in accordance with IDF. No black-box callouts are required because all the submodules generated by Platform Studio that have netlists are instantiated at least two levels down in the hierarchy.

### Porting the UCF

When generating the netlist, Platform Studio also generates a UCF for the design. When developing the UCF for the isolated design, this generated UCF was used as the base because it already contained the pinouts and timing constraints. There were some constraints that were added to this base UCF to support the isolated design. These changes are discussed in this section.

This constraint was added to allow for non-global clock routes to be utilized on the 50 MHz clock route:

```
PIN "U1_clkgen/clock_generator_0/clock_generator_0/PLL0_CLKOUT2_BUFG_INST.O"  
CLOCK_DEDICATED_ROUTE = FALSE;
```

This is required because the lockstep output bus from each of the MicroBlaze processors forwards the clock signal through bit 591. Without this constraint, Map would fail.

The constraint:

```
NET "*/mcb_wrapper_inst/selfrefresh_mcb_mode" TIG;
```

was changed to:

```
NET "U4_peripherals/MCB_DDR3/MCB_DDR3/mcb_ui_top_0/mcb_raw_wrapper_inst/  
selfrefresh_mcb_mode" TIG;
```

to account for the new hierarchy.

The `OUT_TERM = UNTUNED_50` was added to each of the `mcbx_dram` I/O to include a constraint included in the netlist constraints file (NCF) for the DDR3 SDRAM. The DDR3 NCF is not included in the PlanAhead tool project because it causes pinout conflicts.

### Porting the BMM File

When generating the netlist, Platform Studio also generates a BMM for the design. When developing the BMM for the isolated design, this generated BMM was used as the base because it contained the memory mappings. The memory mappings in the BMM only needed to be updated to account for the new design hierarchy. This was done with three simple replaces that placed the hierarchical instance name at the beginning of the memory path where it is instantiated. The three replaces executed were:

- `microblaze_1_bram_block/` changed to `U3_mb1/microblaze_1_bram_block/`
- `microblaze_0_bram_block/` changed to `U2_mb0/microblaze_0_bram_block/`
- `axi_bram_ctrl_0_bram_block/` changed to `U4_peripherals/ axi_bram_ctrl_0_bram_block/`

### Synthesize and Floorplan the Hierarchical Design

Figure 30 shows the reference design progress to this point.

<input checked="" type="checkbox"/>	1. Generate Dual MicroBlaze System in EDK Platform Studio.
<input checked="" type="checkbox"/>	2. Modify Dual MicroBlaze System into Lockstep through EDK Platform Studio.
<input checked="" type="checkbox"/>	3. Build the Dual-Lockstep MicroBlaze System in EDK Platform Studio.
<input checked="" type="checkbox"/>	4. Perform a Quick Sanity Check of the Design.
<input checked="" type="checkbox"/>	5. Prepare Dual-Lockstep MicroBlaze Design Source for Isolated Design.
	6. Synthesize and Floorplan Hierarchical Design.
	7. Run IVT on the Design in UCF Mode.
	8. Implement the Design.
	9. Run IVT on the Design in NCD Mode.
	10. Build Final Software in SDK.
	11. Disconnect the MicroBlaze Processors and Debug Logic.
	12. Re-verify the Re-implemented Design in IVT.

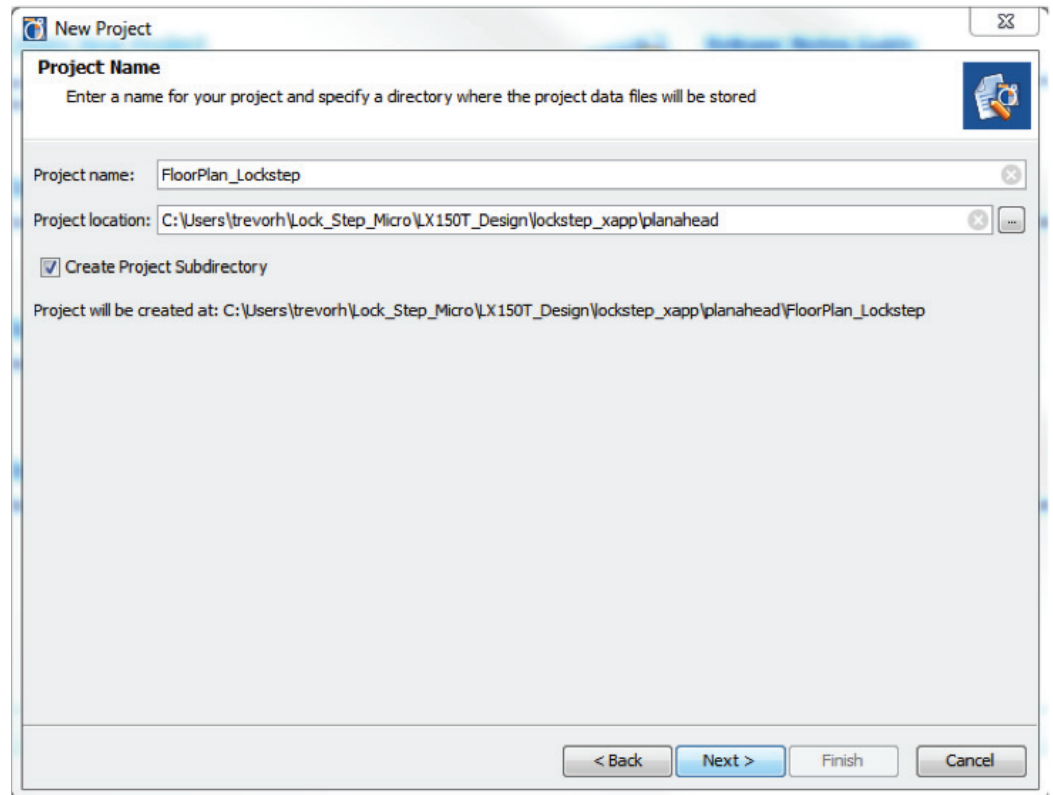
X584\_30\_041112

*Figure 30: Reference Design Progress*

The next sections define the steps to synthesize and floorplan the isolated functions into isolated regions using the PlanAhead tool.

### Building the PlanAhead Tool Project

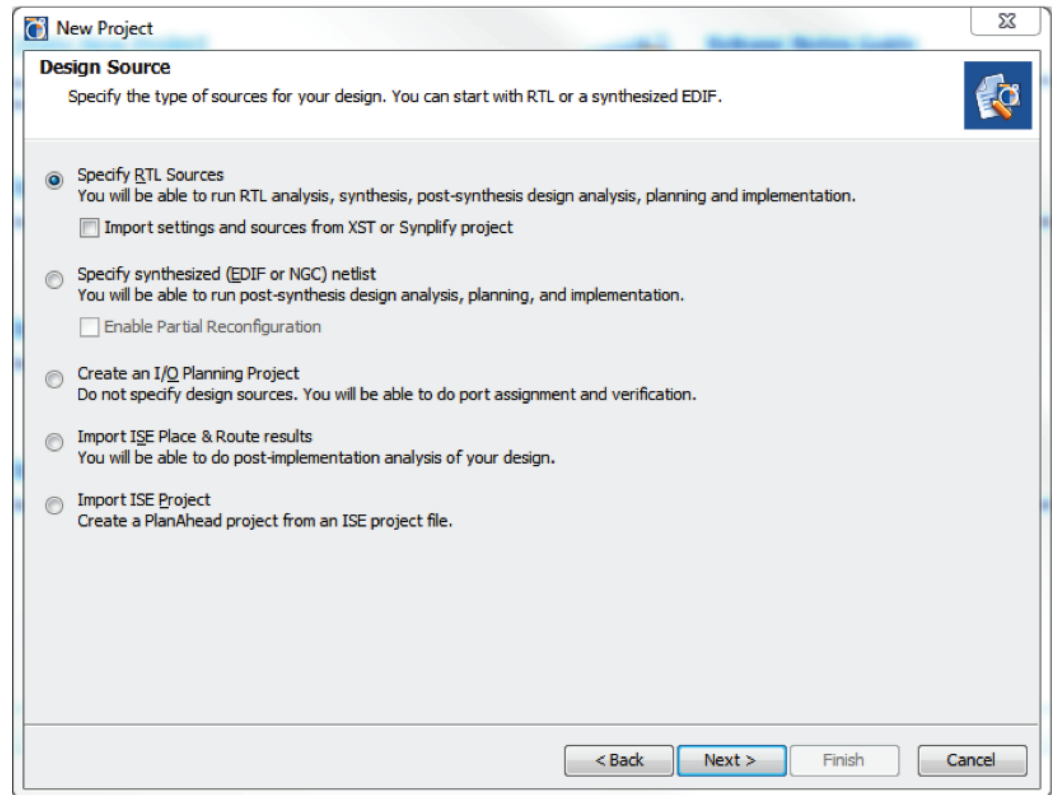
1. Launch the PlanAhead tool, version 13.4 by clicking **Start All Programs Xilinx ISE Design Suite 13.4 > PlanAhead > PlanAhead**.
2. In the Getting Started window, click **Create New Project**.
3. Click **Next** in the New Project wizard.
4. In the Project Name window ([Figure 31](#)), make these settings, then click **Next >**.
  - Project Name: **FloorPlan\_Lockstep**
  - Project Location: <reference design>\planahead
  - **Create Project Subdirectory**: checked



X584\_31\_040412

*Figure 31: PlanAhead Tool New Project - Project Name Window*

5. In the Design Source window ([Figure 32](#)), select **Specify RTL Sources**. Click **Next >**.  
Building the project with the Register Transfer Level (RTL) sources specified provides the PlanAhead tool with the ability to execute a bottom-up synthesis on a hierarchical design without having to perform synthesis externally.



X584\_32\_040412

Figure 32: PlanAhead Tool New Project - Design Source Window

6. In the Add Sources window (Figure 33), click **Add Files....** Select the files listed under each of these directories:

```
<reference design>\src\hdl:
```

- CLOCKGEN\_TOP.vhd
- LOCKSTEP\_SYSTEM\_TOP.vhd
- MB0\_COMPARATOR\_TOP.vhd
- MB0\_TOP.vhd
- MB1\_COMPARATOR\_TOP.vhd
- MB1\_TOP.vhd
- PERIPHERALS\_TOP.vhd

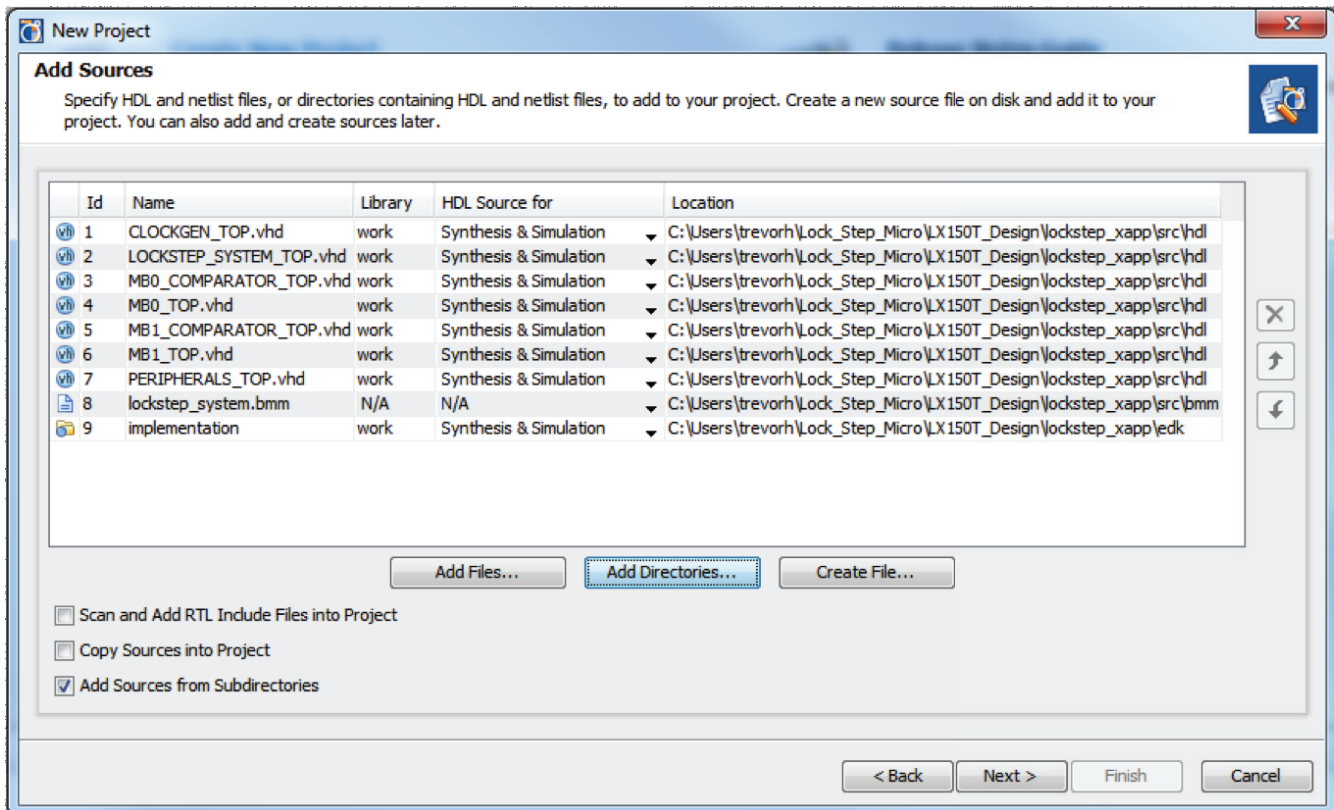
```
<reference design>\src\bmm:
```

- lockstep\_system.bmm

7. In the Add Sources window (Figure 33), click **Add Directories....** Navigate to <reference design>\edk\implementation. Select the directory.

Adding the directory allows the PlanAhead tool to pull netlist and HDL files into the project without having to select each file individually.

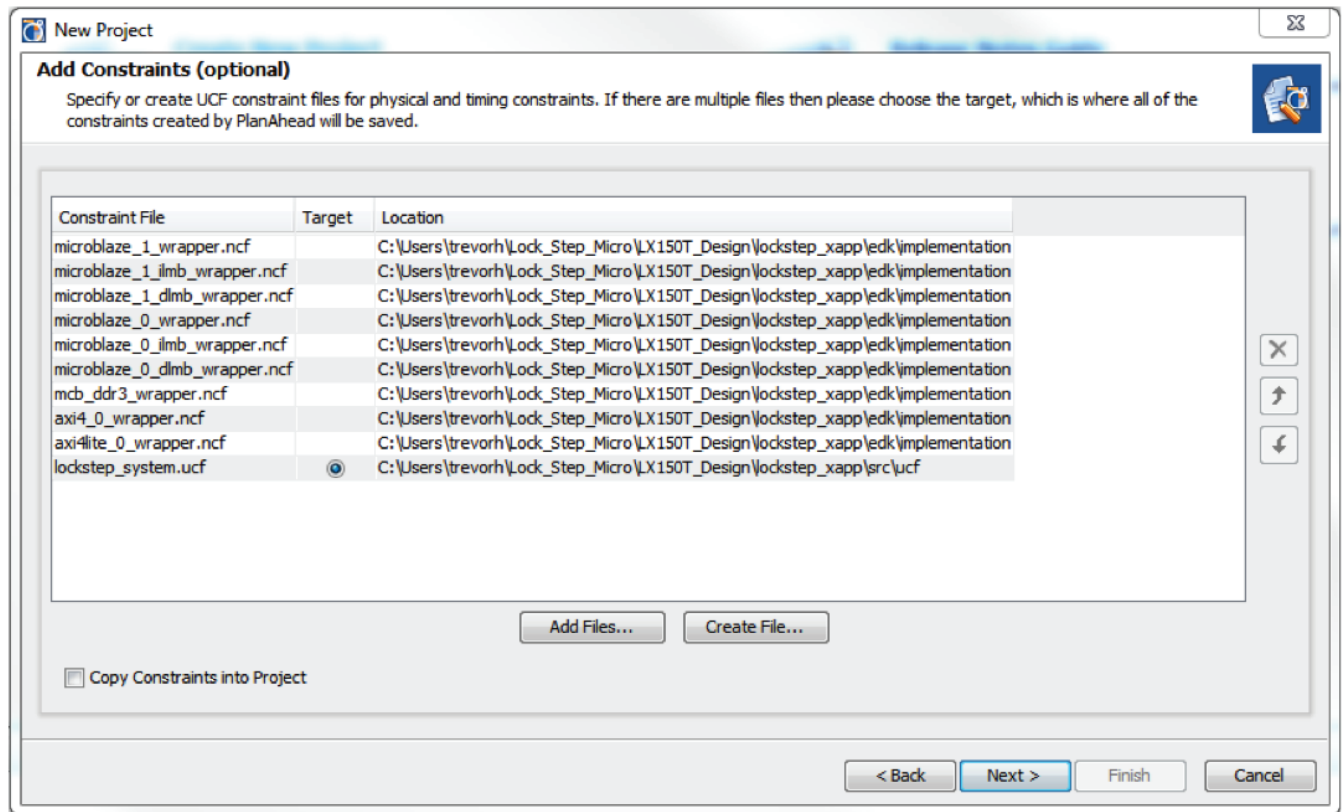
8. In the Add Sources window (Figure 33), verify that **Add Sources from Subdirectories** is checked and that **Copy Sources into Project** and **Scan and Add RTL Include Files into Project** are not checked.



X584\_33\_040412

Figure 33: PlanAhead Tool - Add Sources Window

9. Click **Next >**.
10. Click **Next>** in the Add Existing IP window without adding any files.
11. In the Add Constraints window (Figure 34), there should be a list of NCF files pulled in from the `<reference design>\edk\implementation` directory.
  - a. Add the system UCF by selecting **Add Files...** and then adding the `<reference design>\src\ucf\lockstep_system.ucf` file.
  - b. Select `lockstep_system.ucf` as the target.



X584\_34\_040412

Figure 34: PlanAhead Tool - Add Constraints Window

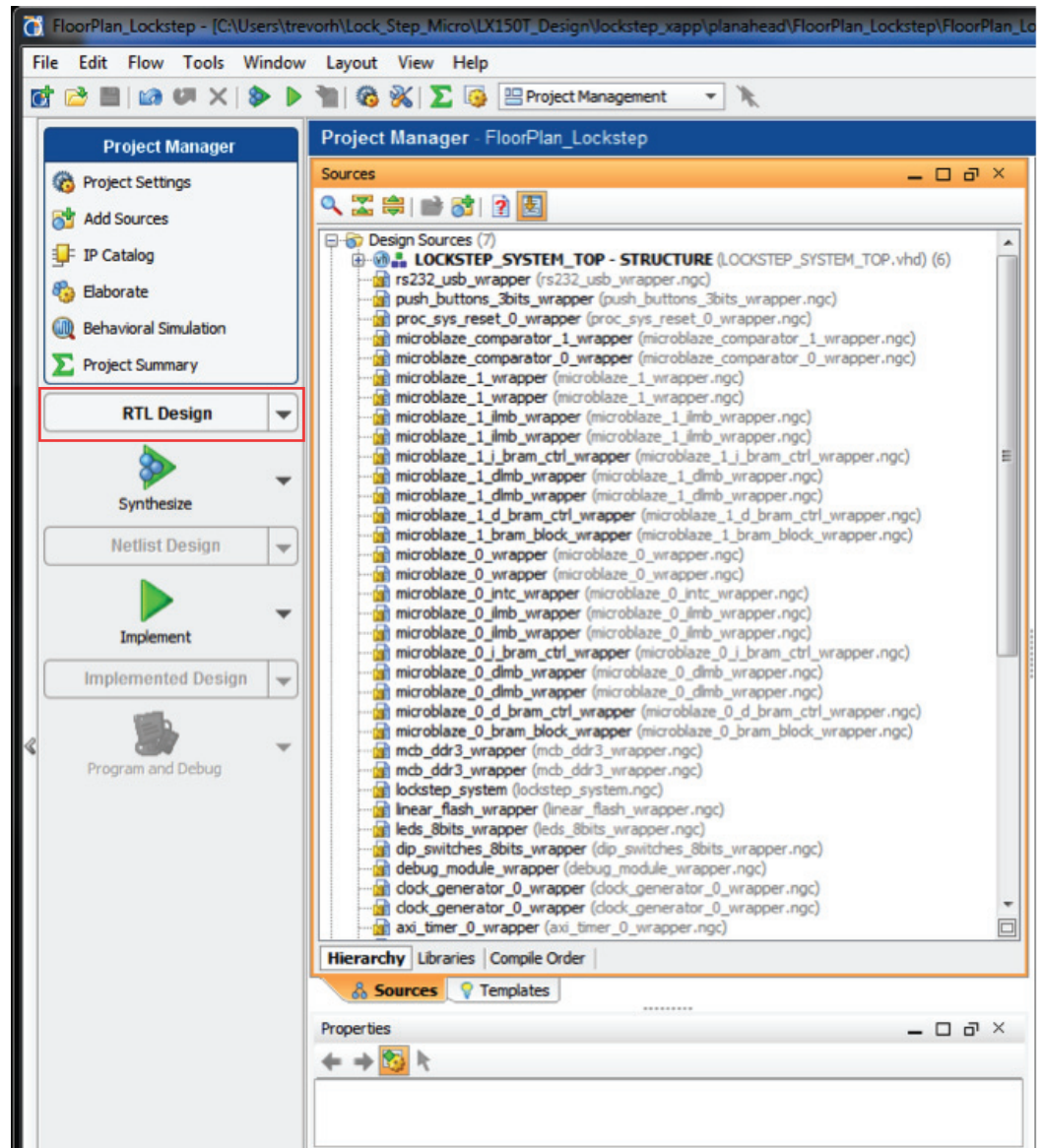
12. With **Copy Constraints into Project** not selected, click **Next >**.
13. In the Default Part window, select the device **xc6slx150tfgg676-3**. Click **Next >**.
14. Click **Finish** in the New Project Summary window. The tool creates the PlanAhead tool project (Figure 35).





These steps describe building the RTL netlist and using the netlist to indicate the partitions:

1. In the left of the PlanAhead tool window, click **RTL Design** to build the RTL netlist (Figure 36).

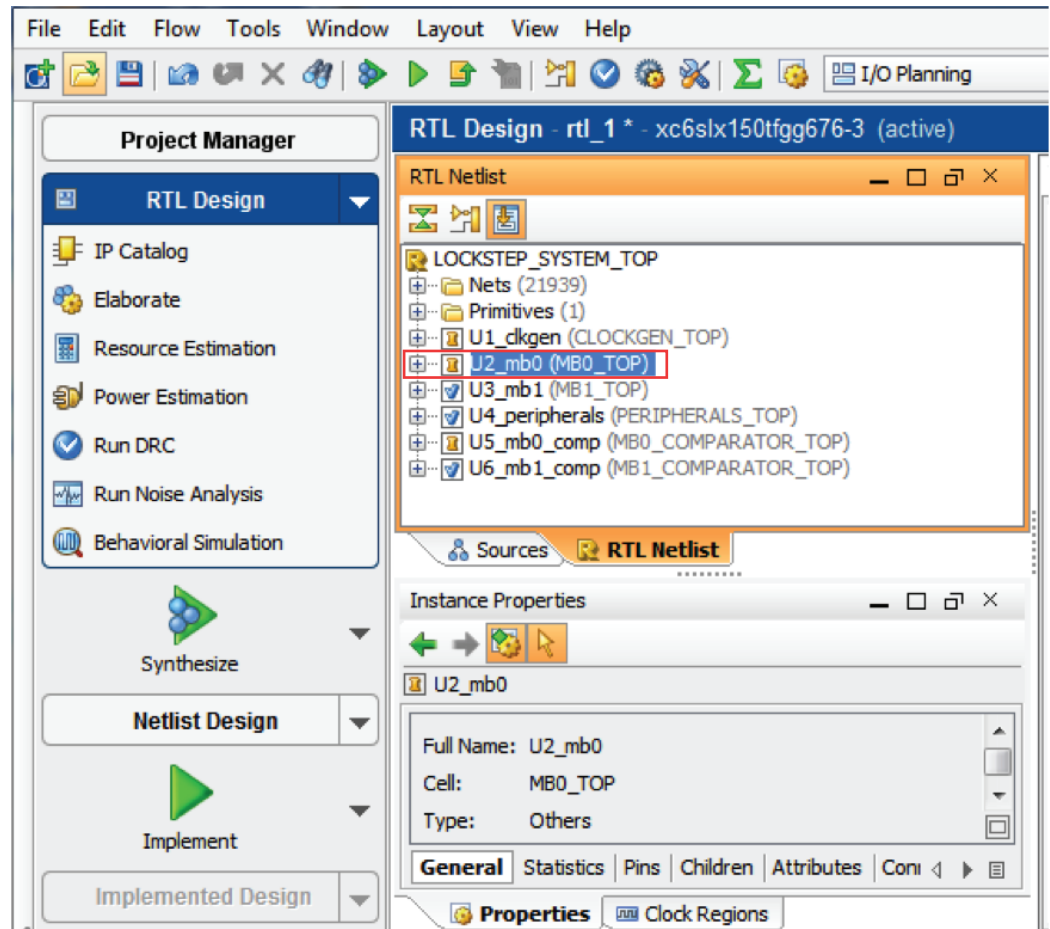


X584\_36\_040412

Figure 36: PlanAhead Tool Window - RTL Design Button

2. After the RTL netlist is built, click the RTL Netlist tab in the PlanAhead tool's window (Figure 37).
3. With the RTL netlist now loaded, the PlanAhead tool needs information about what the isolated functions are by initially associating each isolated function to a partition. In the RTL Netlist tab, left-click then right-click **U2\_mb0** and select **Set Partition** to indicate that the MBO\_TOP instance is a partition in the design (Figure 37).

**Note:** After a partition is set for a component, the square to the left of the instance callout in the RTL Netlist tab changes to a solid yellow. The U4\_peripherals instance has a blue check in the box because area group constraints are provided in the lockstep\_system.ucf file for that instance.

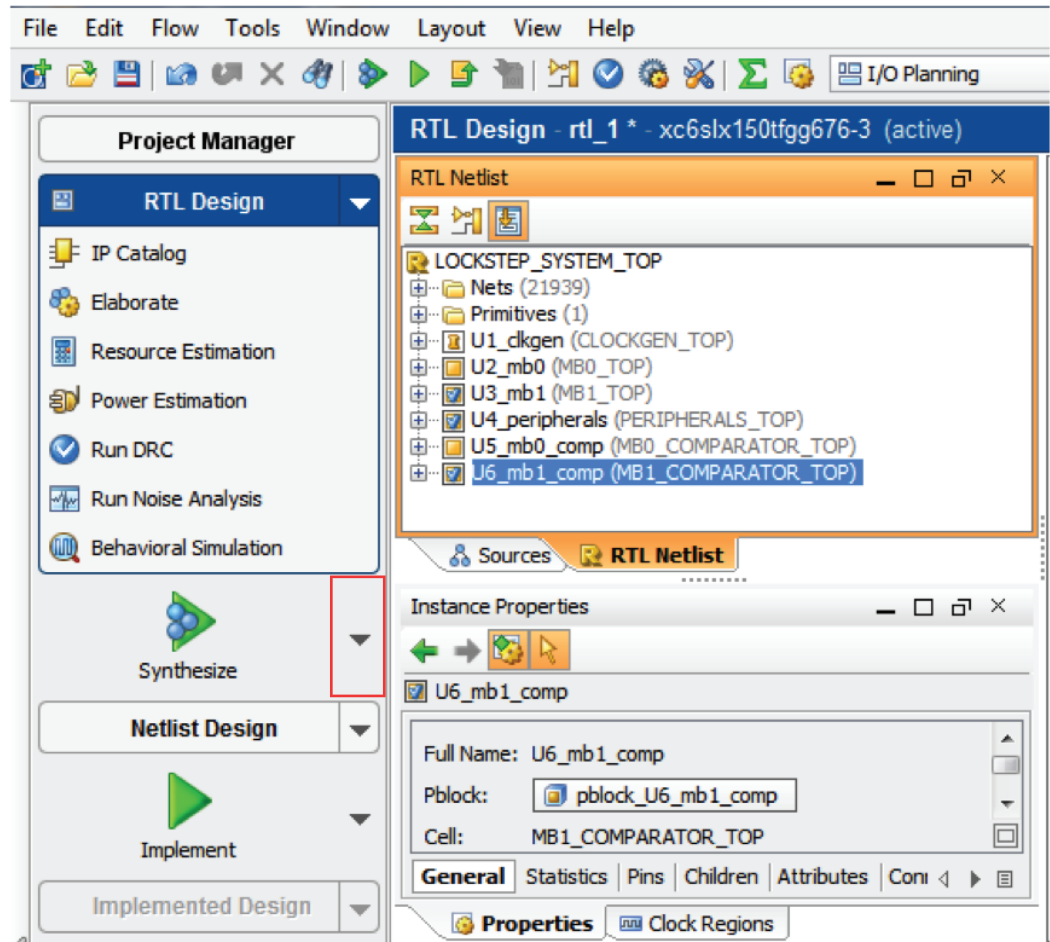


X584\_37\_040412

Figure 37: PlanAhead Tool RTL Netlist Tab

4. Repeat [step 3](#) for **U3\_mb1**, **U4\_peripherals**, **U5\_mb0\_comp**, and **U6\_mb1\_comp**.  
With the partitions indicated, the PlanAhead tool has enough information to automatically execute a bottom-up synthesis of the design. The **U1\_clkgen** instance is not set as a partition because it contains the global clocking components that cannot be included within a partition per IDF.
5. Prepare to synthesize the design by making the Synthesis settings. In the left of the PlanAhead tool window, click the down arrow next to Synthesize and select **Synthesis**

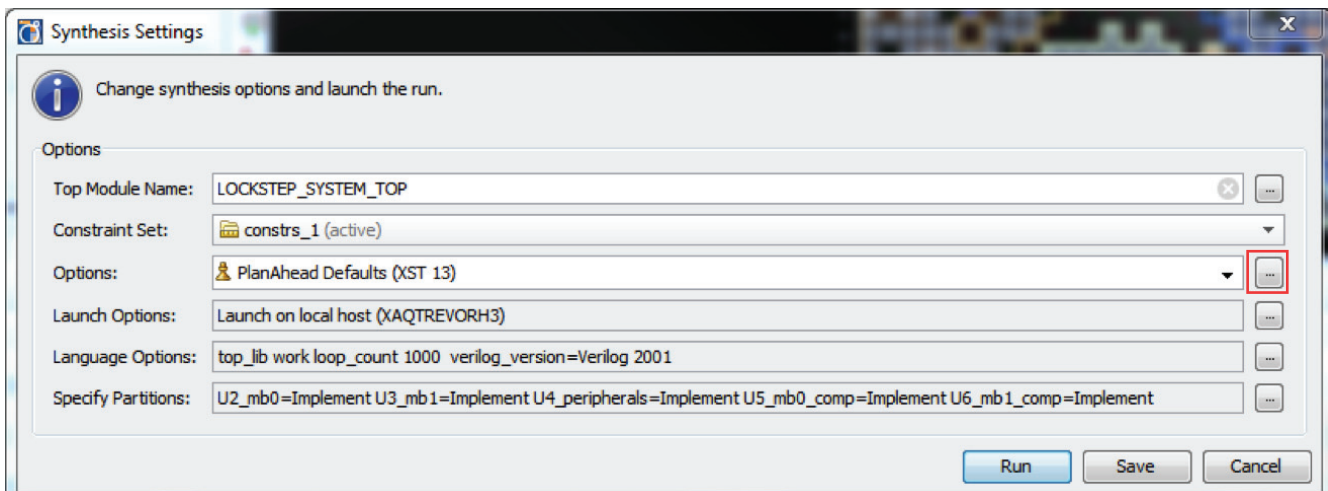
Settings...(Figure 38).



X584\_38\_040412

Figure 38: PlanAhead Tool Window - Synthesize Menu

- In the Synthesis Settings window (Figure 39), click the ... button next to the **Options:** line to open the Design Run Settings window.

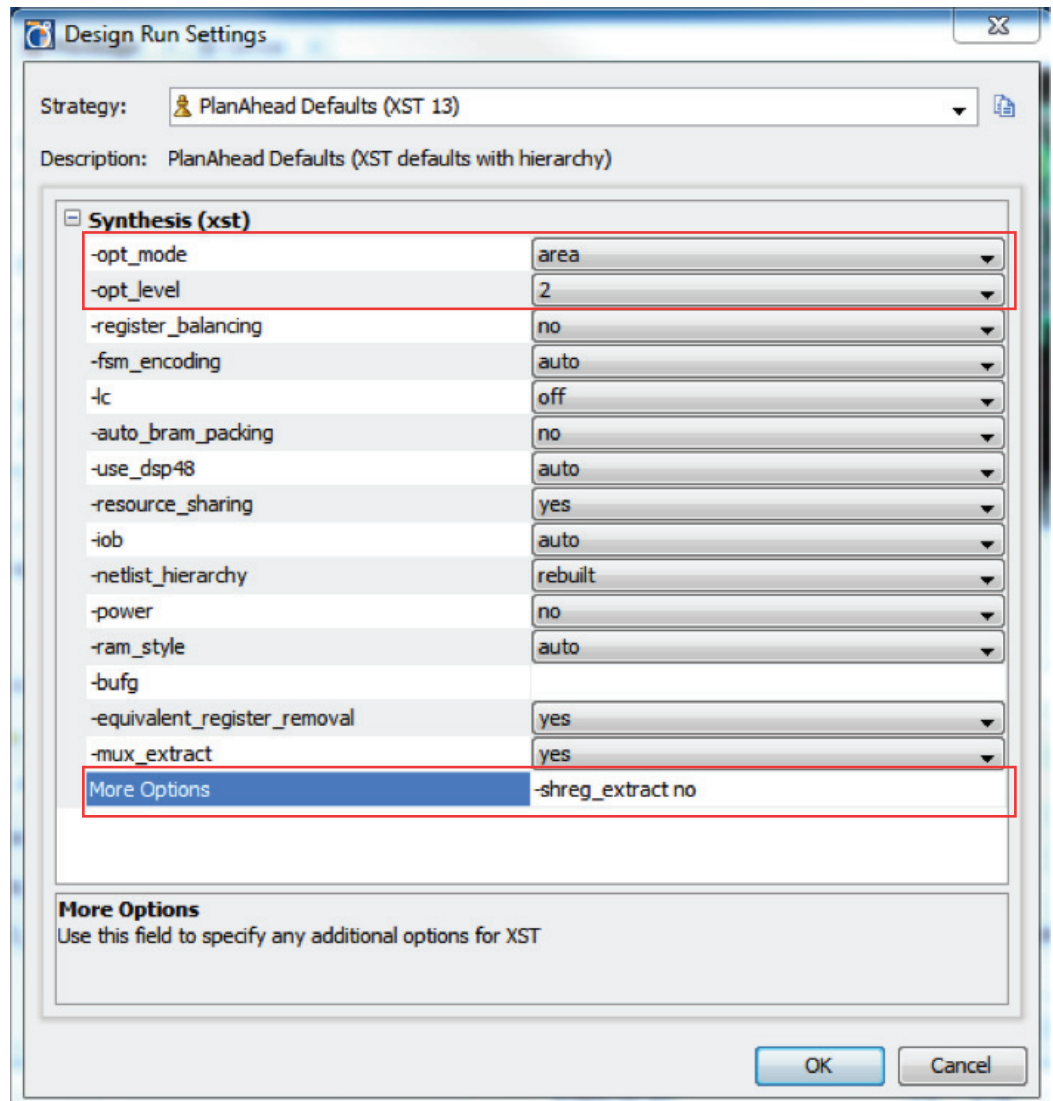


X584\_39\_040412

Figure 39: PlanAhead Tool Synthesis Settings Window

7. In the Design Run Settings window (Figure 40), change these settings:
  - -opt\_mode: **area**
  - -opt\_level: **2**
  - More options: **-shreg\_extract no**

The setting **-shreg\_extract no** prevents shift register inference.

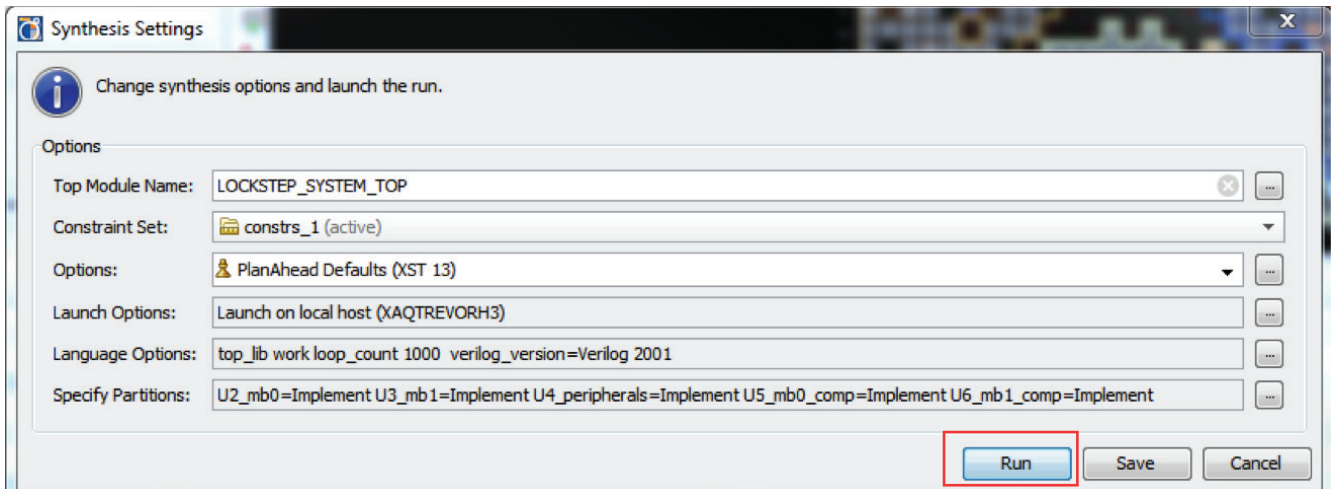


X584\_40\_040412

Figure 40: PlanAhead Tool Design Run Settings - Synthesis

8. Click **OK** to save the settings. Click **Run** in the Synthesis settings menu to start synthesis (Figure 41). If prompted to save the project, click **Save**.

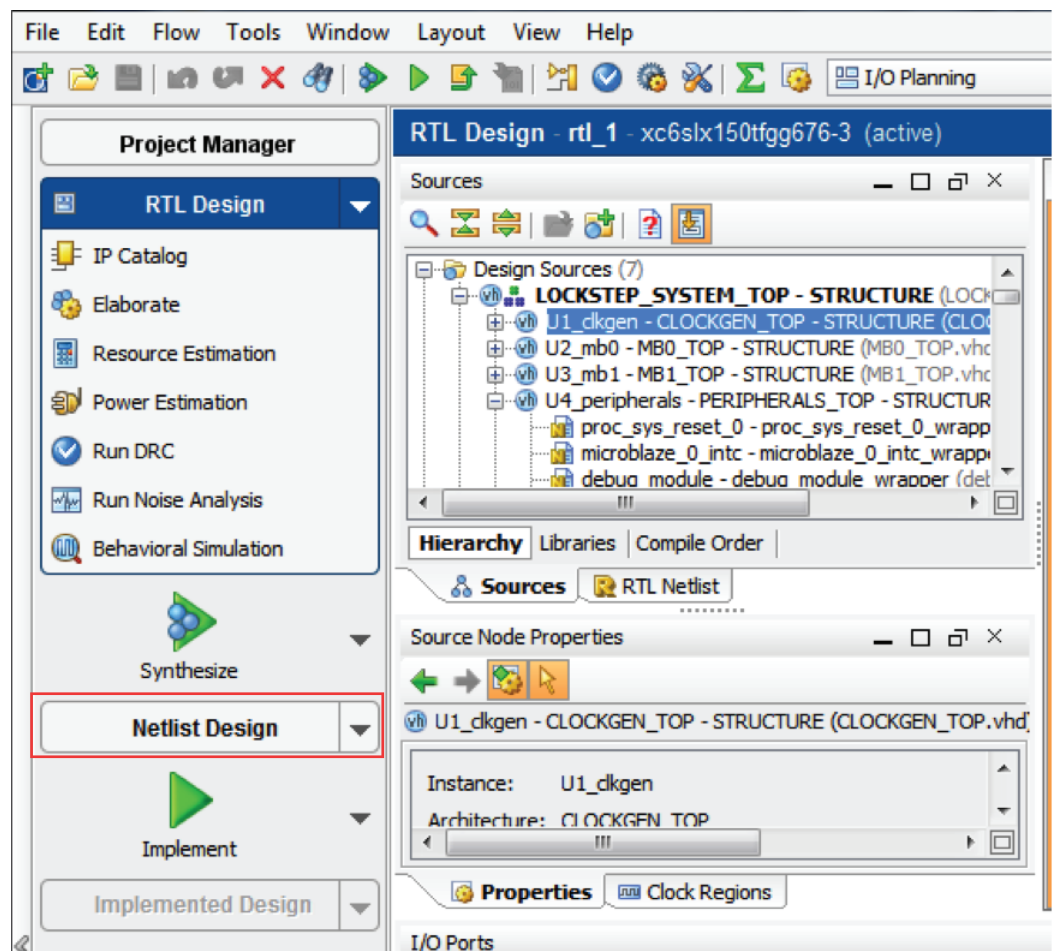
**Note:** Running synthesis creates a component netlist that can be floorplanned for an isolated design.



X584\_41\_040412

Figure 41: PlanAhead Tool Synthesis Settings - Run

9. When synthesis is complete, open the netlist design by either selecting **Open Netlist Design** at the `Synthesis Completed` prompt or click **Netlist Design** on the left side of the PlanAhead tool window (Figure 42).



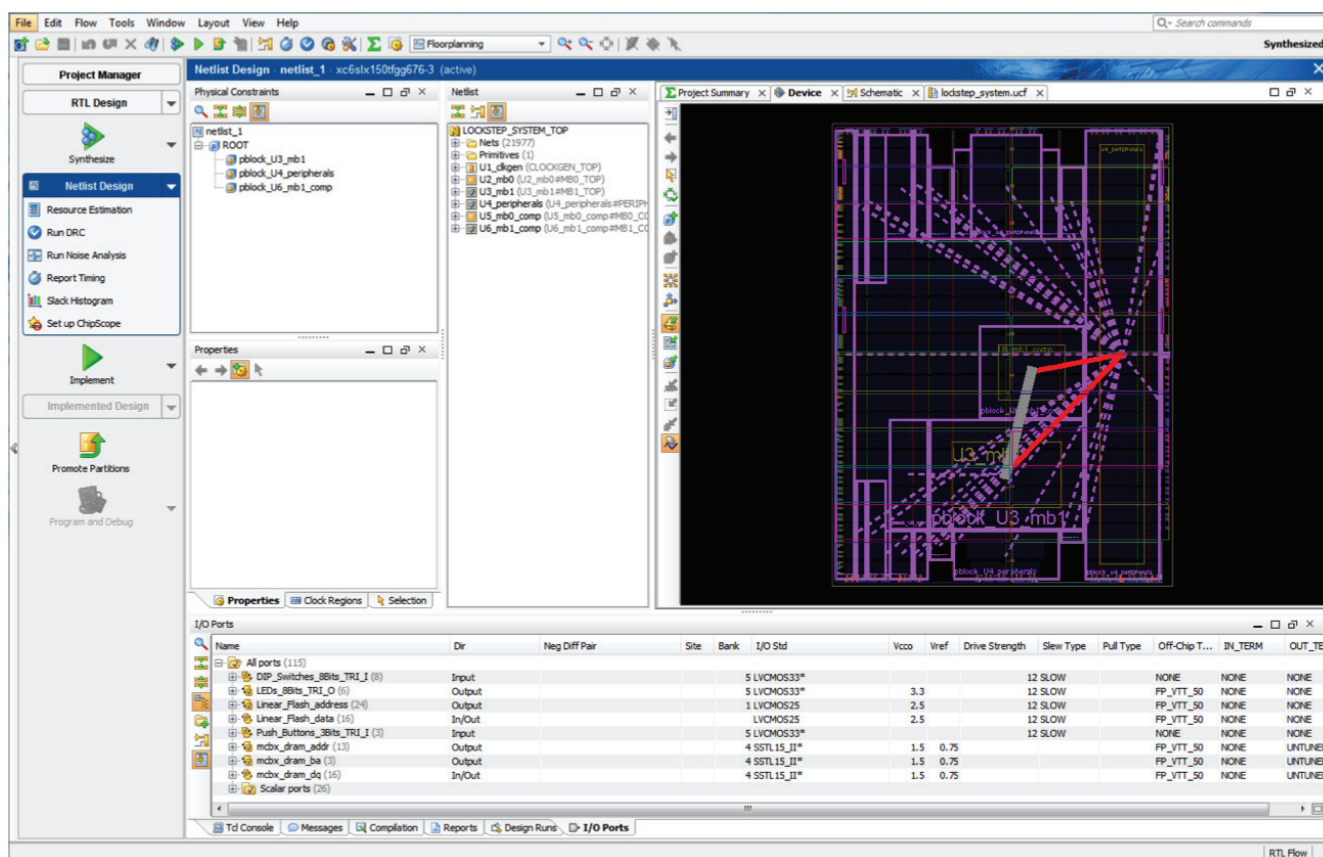
X584\_42\_040412

Figure 42: PlanAhead Tool Window - Netlist Design Button



10. Click **Yes** to close the **RTL Design** before opening **Netlist Design**.

After the netlist is loaded, the PlanAhead tool updates the Device tab with a graphical view of the device showing the area group constraints that already exist in the UCF (Figure 43). For this reference design, area group constraints have already been provided for the U3\_mb1, U4\_peripherals, and U6\_mb1\_comp isolated functions. Area group constraints define the components that are reserved for implementation of a partition. For isolated designs, partitions define isolated regions. The PlanAhead tool uses pblocks to define the area group definition for an isolated region. The Netlist tab in the PlanAhead tool indicates the synthesized netlist. The Physical Constraints tab defines the partitions by identifying the pblocks and can be found by clicking from the PlanAhead tool window menu bar **Window > Physical Constraints**.



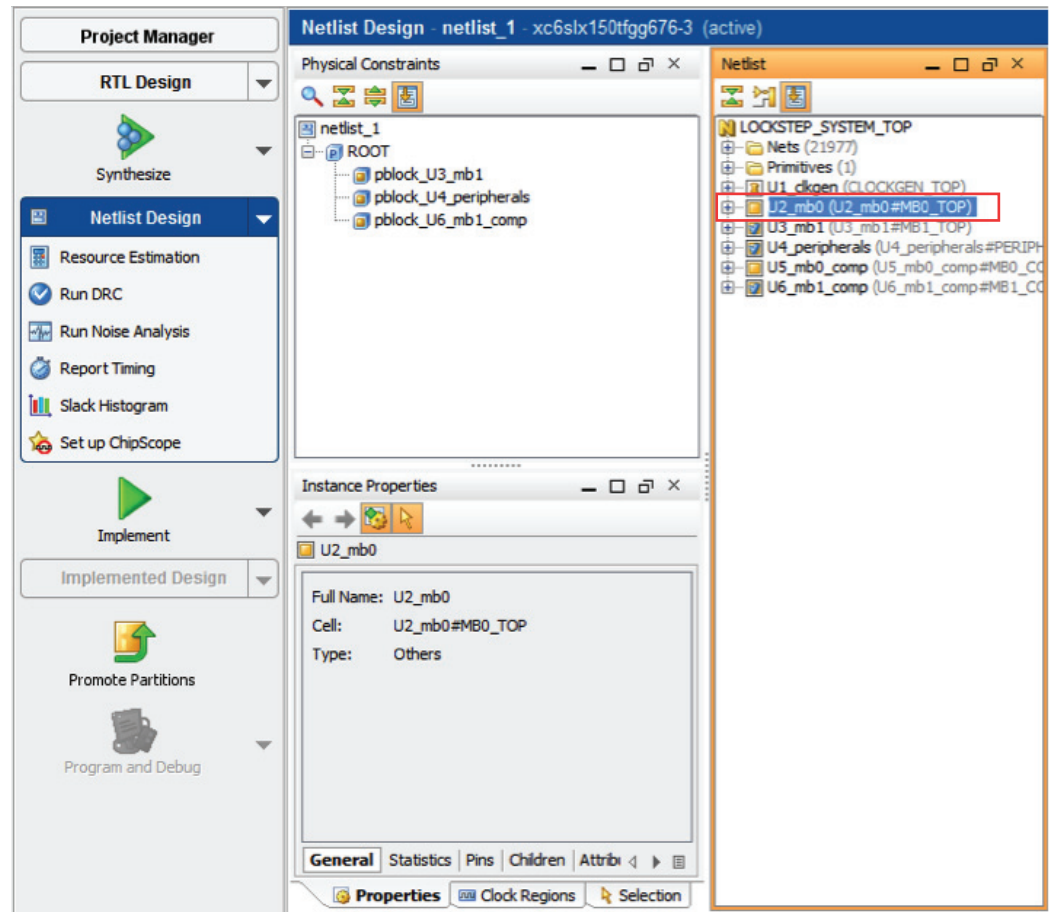
X584\_43\_040412

Figure 43: PlanAhead Tool with Netlist Design Open

### Creating the Remaining Pblocks

These steps describe assigning new pblocks to the design for the U2\_mb0 and U5\_mb0\_comp instances:

1. In the Netlist pane, left-click then right-click the **U2\_mb0** instance and select **New Pblock...** (Figure 44).



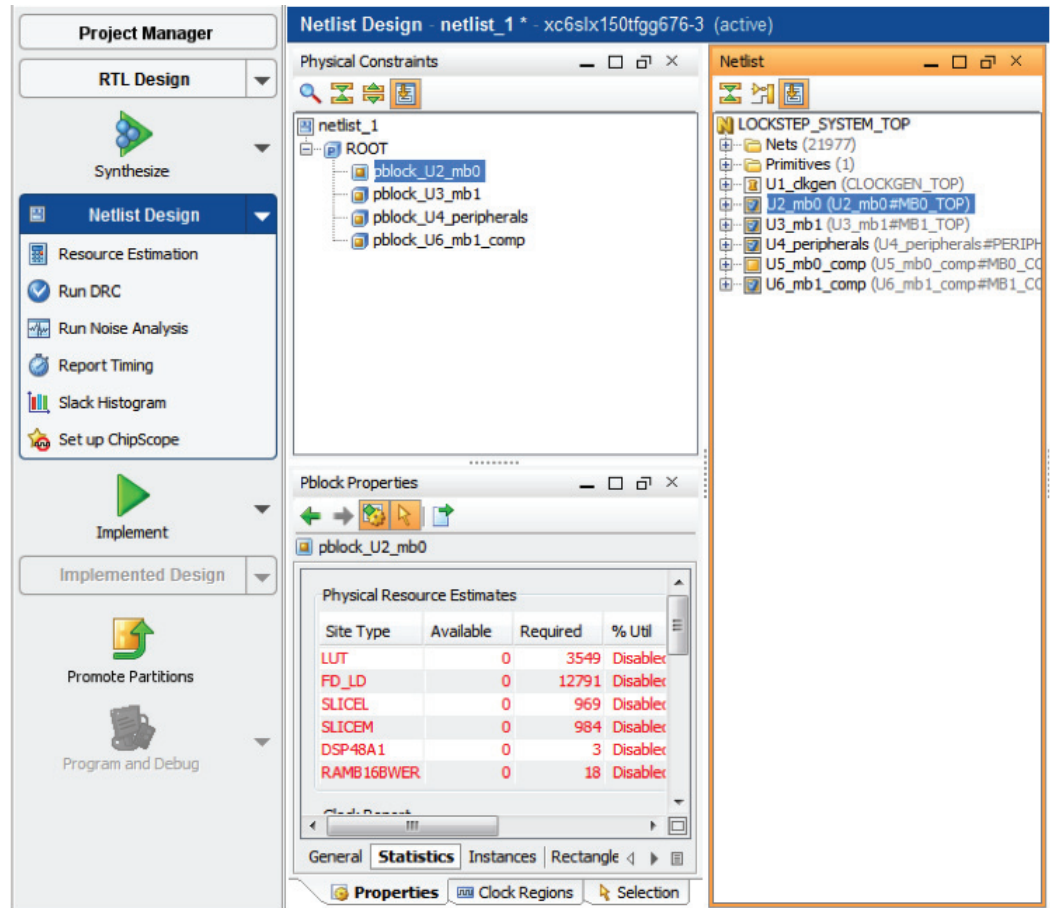
X584\_44\_040412

Figure 44: Netlist Instance of U2\_mb0

2. In the New Pblock window, verify that the **Assign selected instance** checkbox is checked and the name is set to **pblock\_U2\_mb0**.
3. Click **OK** to create the new pblock.

After the pblock is created, it appears in the Physical Constraints pane of the PlanAhead tool. The netlist instance also gets a blue check, indicating there is a pblock assigned to the instance (Figure 45).





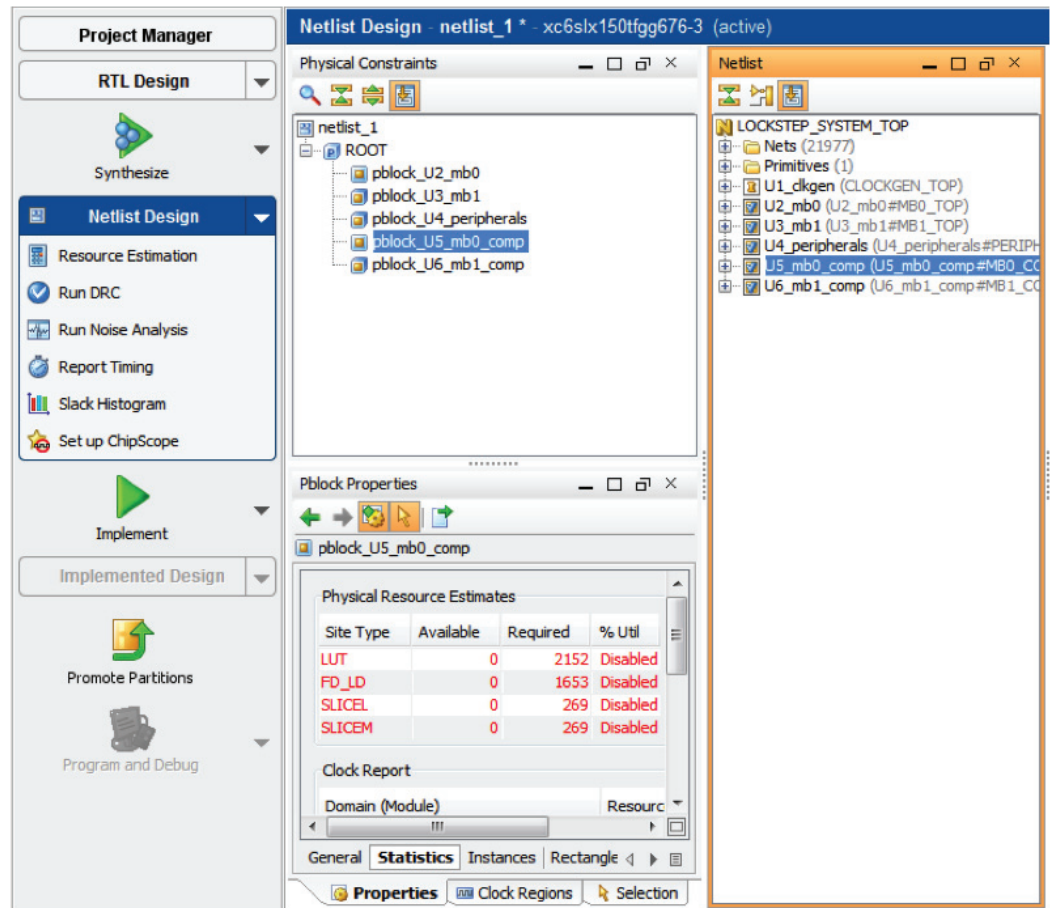
X584\_45\_040412

Figure 45: New Pblock Created for U2\_mb0

At this stage in the design, device resources are not assigned to the pblock but only create the pblock. Before assigning the device resources, IDF attributes need to be set so that the PlanAhead tool allows all device resources to be floorplanned.

- Repeat [step 1](#) through [step 3](#) to create a pblock for the U5\_mb0\_comp instance. The name of the instance should be **pblock\_U5\_mb0\_comp** (see [Figure 46](#)).

**Note:** The pblocks could have also been created within the RTL netlist, before synthesis, if desired. For this application note, it was chosen to create them after synthesis.



X584\_46\_040412

Figure 46: All Pblocks Created

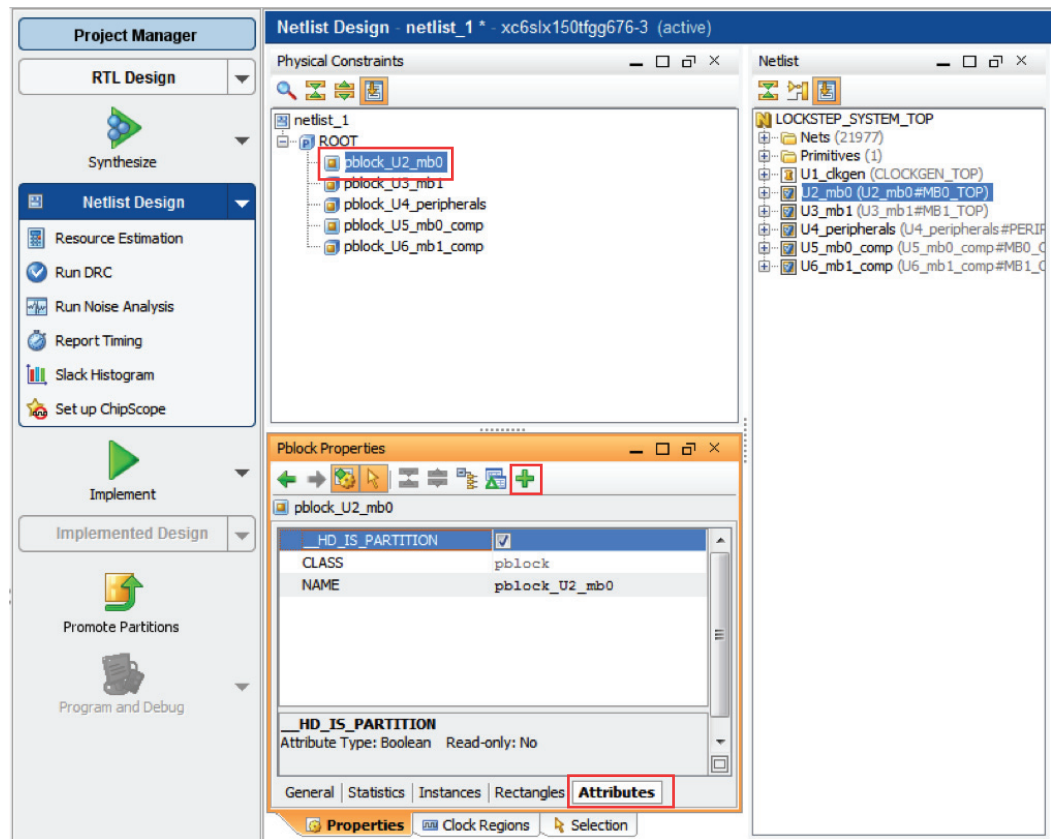
## Setting the IDF Attributes

Two attributes must be set through the PlanAhead tool on each partition to allow for complete floorplanning and building of an isolated design when using the IDF. The first is the PRIVATE attribute. The PRIVATE attribute, when set to **NONE**, allows global clock component placement and routing within and across an isolated region. The second attribute is the SCC\_ISOLATED attribute, which when set indicates to the Xilinx tools that the partition is to be floorplanned and implemented as a fully isolated function. If SCC\_ISOLATED is not set in the PlanAhead tool, a user can only floorplan SLICE, RAMB, and DSP48 components. If SCC\_ISOLATED is set, all components within the device can be floorplanned.

**Note:** Setting the SCC\_ISOLATED attribute in the PlanAhead tool sets the Isolated = True setting for each partition in the `xpartition.pxml` file.

These steps describe how to set the PRIVATE and SCC\_ISOLATED attributes within the PlanAhead tool (see Figure 47):

1. In the Physical Constraints pane, click the **pblock\_u2\_mb0** pblock.
2. In the Pblock Properties pane, click the **Attributes** tab.
3. In the Pblock Properties pane's Attributes tab, click the green plus sign to add an attribute.



X584\_47\_040412

Figure 47: Adding Attribute to Pblock

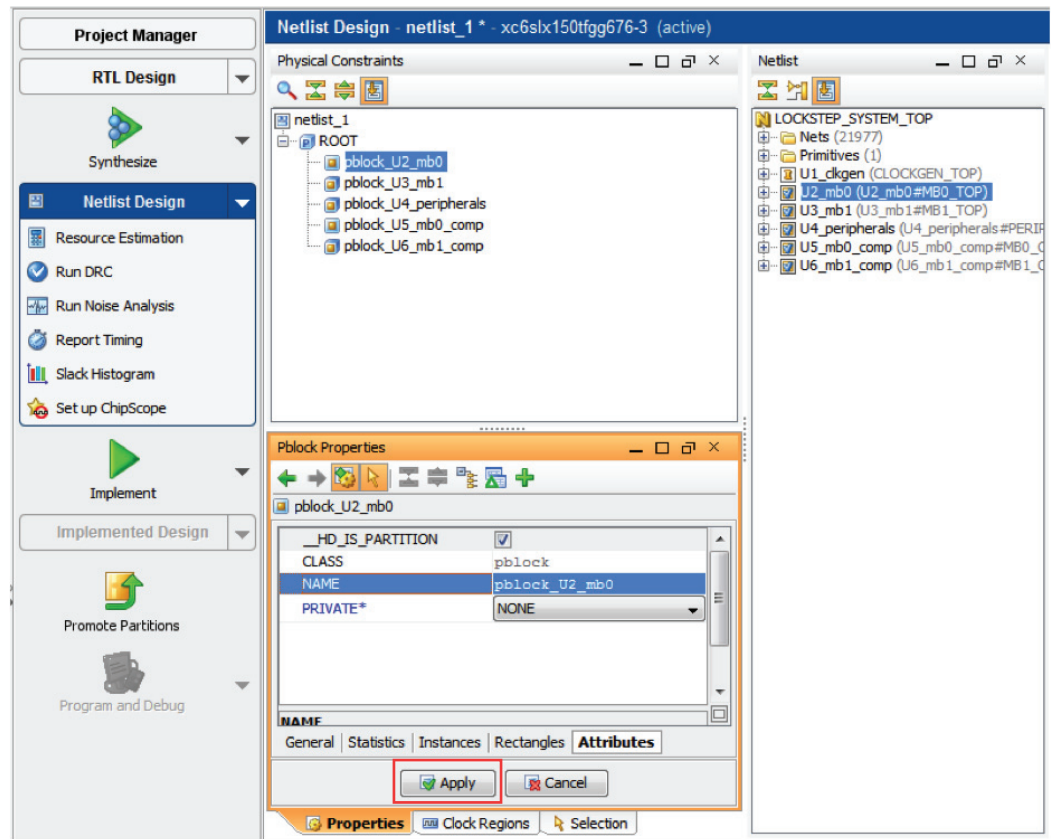
4. In the Add Pre-defined Attributes window (Figure 48), search and select the attribute **PRIVATE**. Click **OK**.



X584\_48\_040412

Figure 48: Selecting PRIVATE Attribute

5. In the Pblock Properties pane's Attributes tab, click **Apply** to set the PRIVATE attribute to **NONE** (Figure 49).



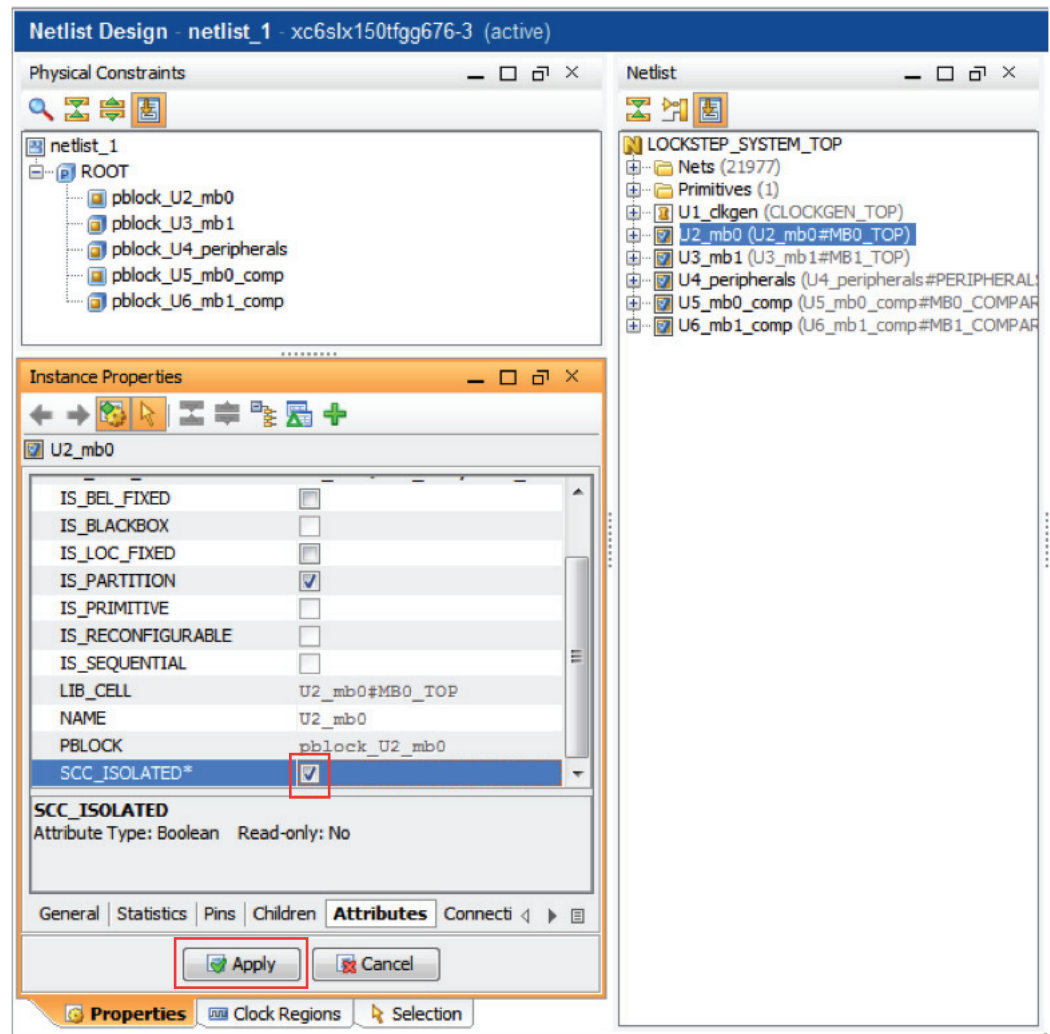
X584\_49\_040412

Figure 49: PlanAhead Tool Physical Constraints and Pblock Properties

6. Repeat [step 1](#) through [step 5](#) to add and set the PRIVATE attribute to **NONE** for **pblock\_U3\_mb1**, **pblock\_U4\_peripherals**, **pblock\_U5\_mb0\_comp**, and **pblock\_U6\_mb1\_comp**.
7. To begin setting the SCC\_ISOLATED attribute, in the Netlist pane, click the **U2\_mb0** instance ([Figure 50](#)).
8. In the Instance Properties pane, click the **Attributes** tab ([Figure 50](#)).
9. Add the SCC\_ISOLATED attribute by clicking the green plus sign within the Instance Properties menu ([Figure 50](#)).

**Note:** If the green plus sign is not selectable, save the PlanAhead tool project. Then close the netlist design by clicking the down arrow next to Netlist Design and selecting **Close**. After the netlist design has closed, re-open the netlist by clicking **Netlist Design**. After the netlist design is open again, restart at [step 7](#).

10. Within the Add Pre-defined Attributes window, search SCC\_ISOLATED and select the **SCC\_ISOLATED** attribute. Click **OK**.
11. Set the **SCC\_ISOLATED** attribute by checking the checkbox in the Instance Properties pane's **Attributes** tab. Click **Apply** to save the change ([Figure 50](#)).



X584\_50\_040412

Figure 50: PlanAhead Tool Instance Properties - SCC\_ISOLATED

Repeat [step 7](#) through [step 11](#) to set the **SCC\_ISOLATED** attribute for the **U3\_mb1**, **U4\_peripherals**, **U5\_mb0\_comp**, and **U6\_mb1\_comp** instances. **Save** the overall PlanAhead tool project.

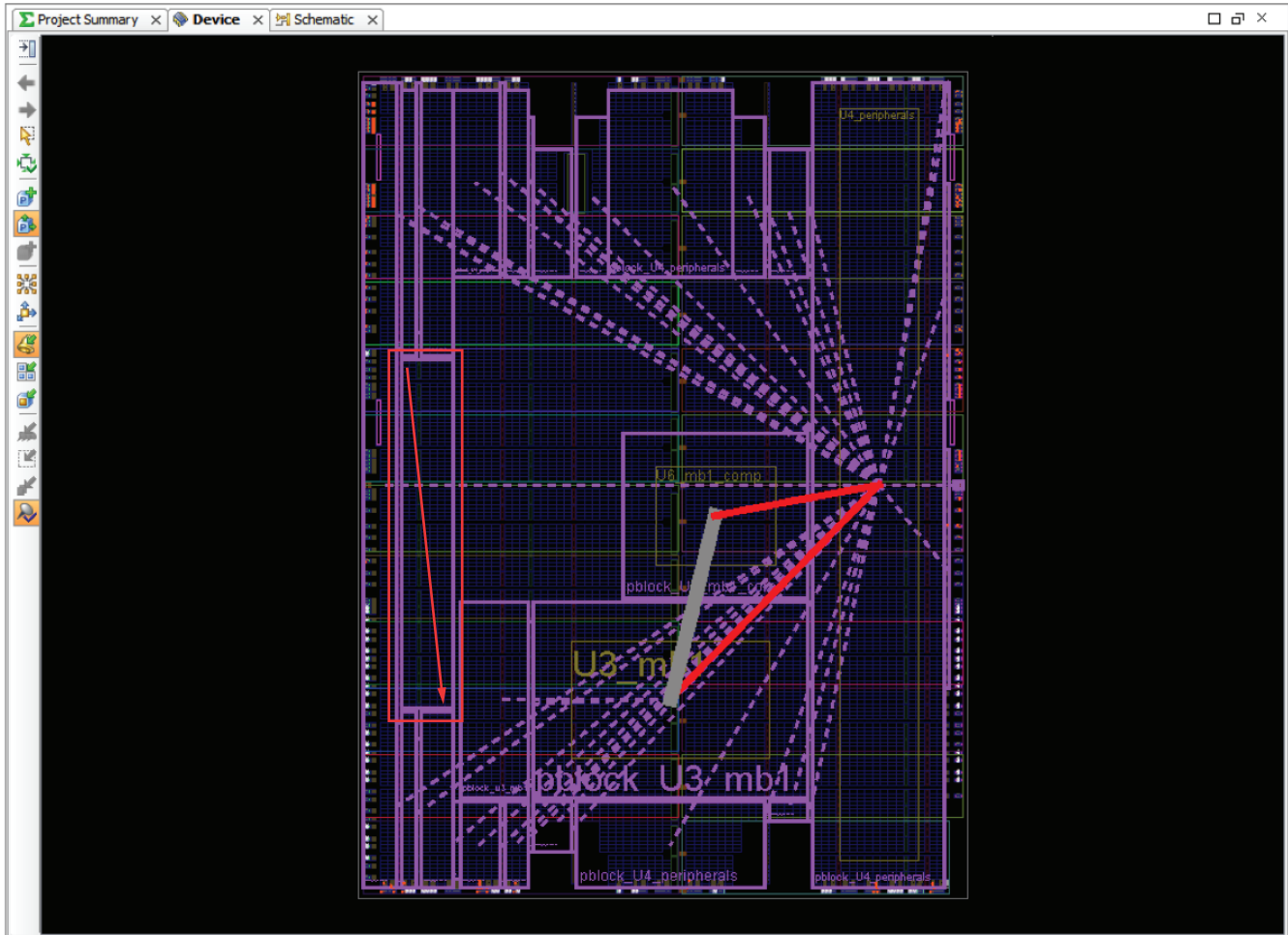
## Floorplanning the Design

The floorplan for the U3\_mb1, U4\_peripherals, and U5\_mb1\_comp isolated functions have already been provided through the `lockstep_system.ucf` file provided with the reference design. In this section, the floorplans for the remaining isolated functions are created. Creating a floorplan is done by creating pblock rectangles in the Device pane of the PlanAhead tool that visually define the area and resources of the device where that isolated function is to be implemented. For isolated designs, each isolated function is separated by a fence, creating isolated regions. A fence is defined as one or multiple user tiles that exist between two isolated regions, and does not contain routing or logic. The minimum required user tile number for a valid fence is dependent on the user tile type (i.e., DSP48, RAMB, SLICE, DCM, or PLL) that defines the fence.

The steps in this section define creating the floorplan for the remaining isolated functions in the design. A fully floorplanned UCF is provided at `<reference design>\reference_files\ucf_w_all_partitions`, which can be copied into the project.



1. In the Physical Constraints pane, left-click then right-click the **pblock\_U5\_mb0\_comp** pblock and select **Set Pblock Size**.
2. In the Device pane, draw a vertical rectangle on the left side of the device in the empty cutout space next to the pblock\_U4\_peripherals floorplanned area (Figure 51). Rectangles are drawn by holding down the left mouse button while tracing the rectangle's diagonal.

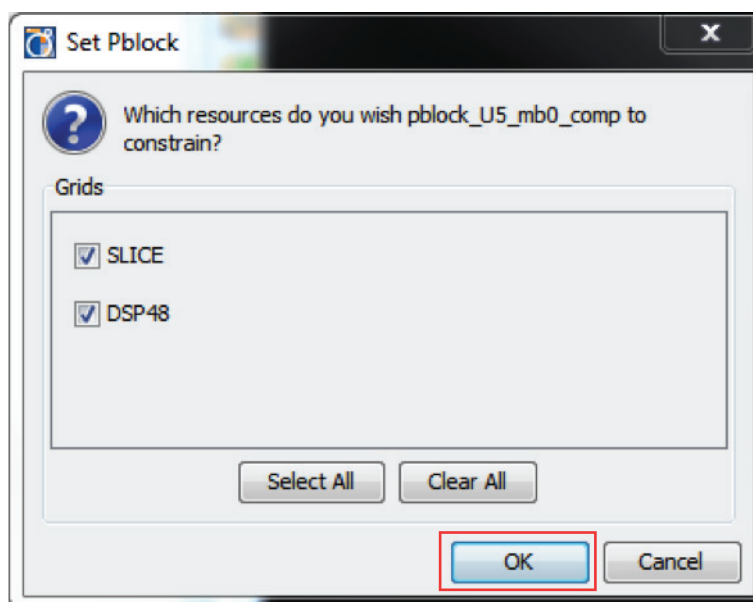


X584\_51\_040412

Figure 51: Drawing the Pblock Rectangle for pblock\_U5\_mb0\_comp

3. In the Set Pblock window (Figure 52), make sure that **SLICE** and **DSP48** resources are checked to be constrained in **pblock\_U5\_mb0\_comp**. Click **OK**.



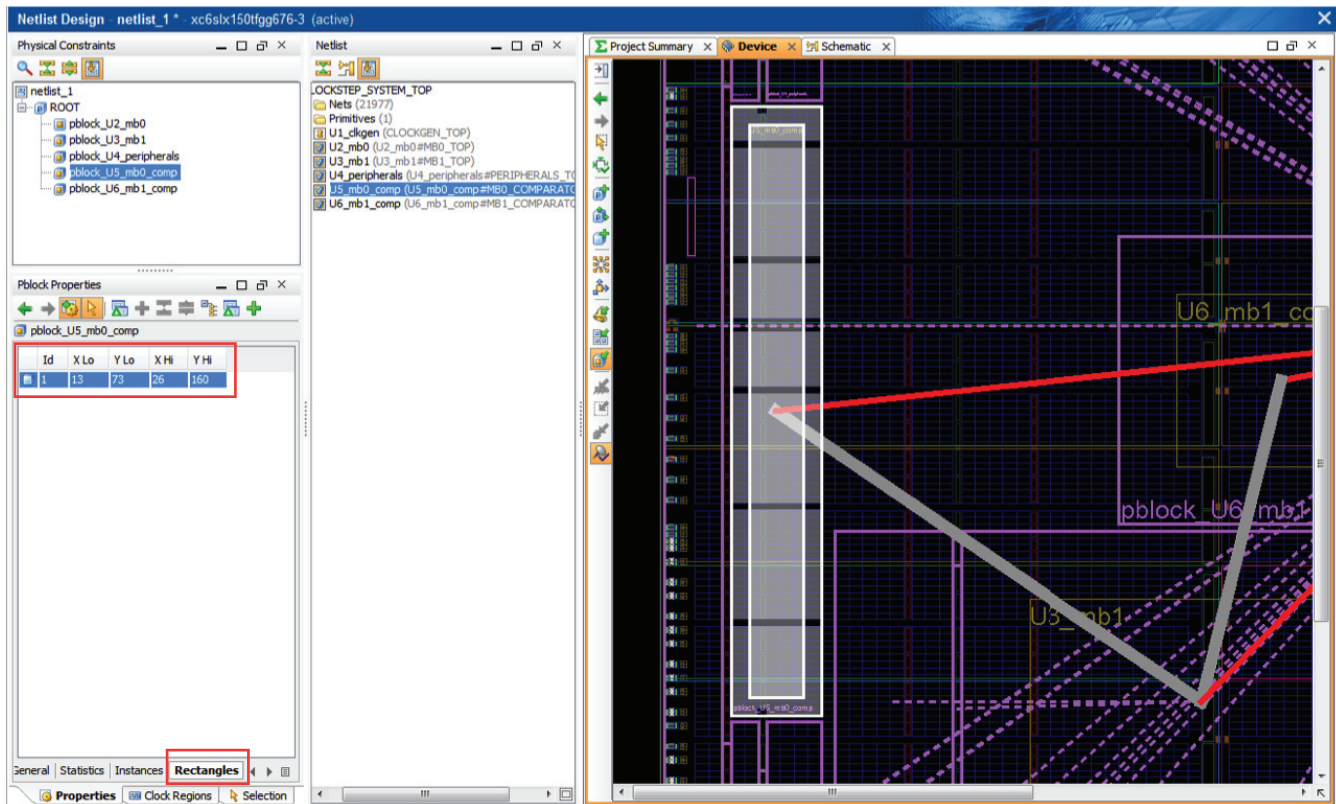


X584\_52\_040412

Figure 52: Resources for pblock\_U5\_mb0\_comp

A highlighted rectangle appears, indicating that the device resources have been assigned to the pblock associated with the U5\_mb0\_comp partition (see [Figure 53](#)).

4. Click the Rectangles tab in the Pblock Properties pane and adjust the rectangle through click and drag actions in the Device pane (see [Figure 53](#)) until the Rectangle settings are:
  - X Lo: **13**
  - Y Lo: **73**
  - X Hi: **26**
  - Y Hi: **160**

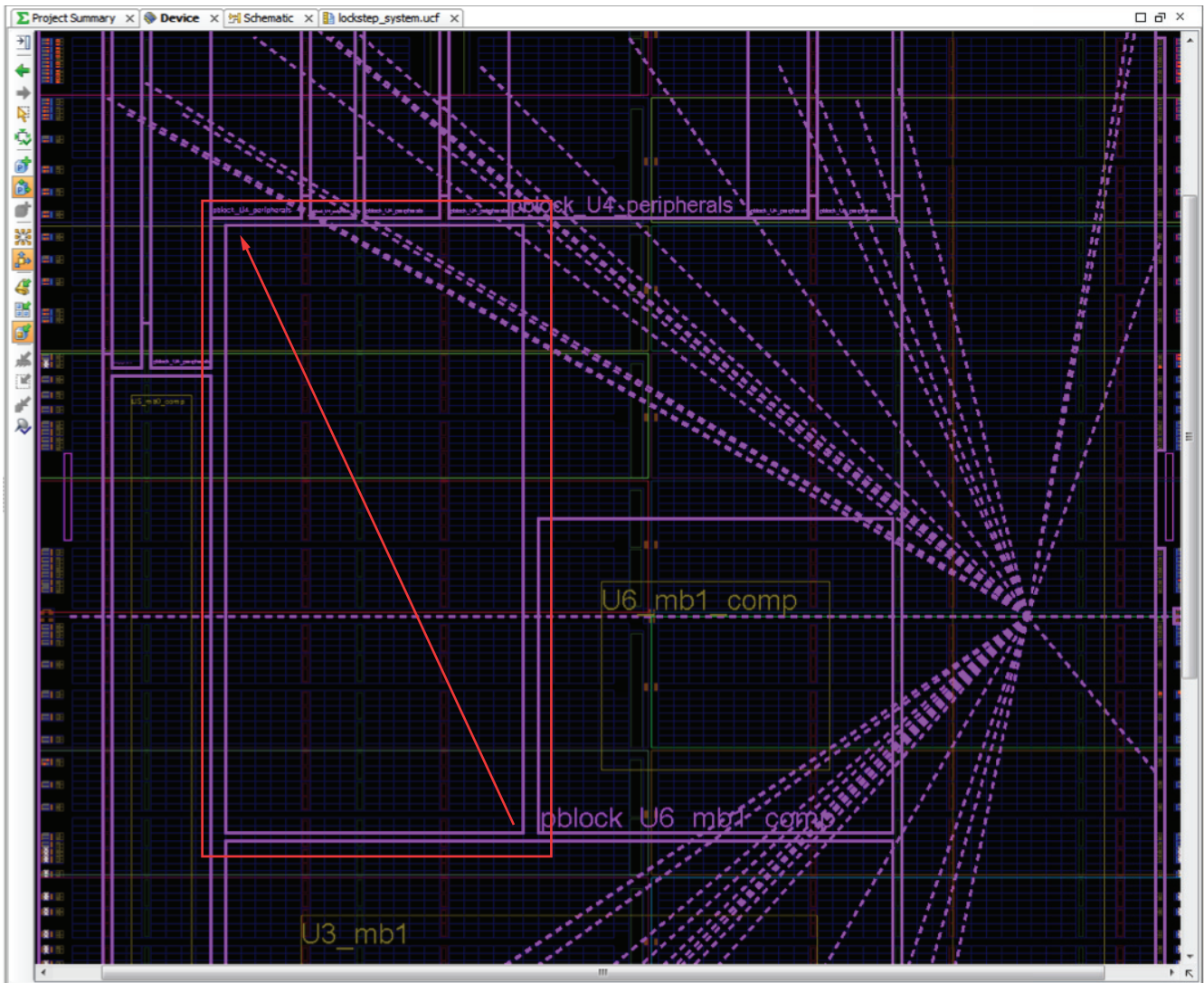


X584\_53\_040412

Figure 53: Final Pblock Rectangle for pblock\_U5\_mb0\_comp

**Note:** In the new pblock rectangle for pblock\_U5\_mb0\_comp, on the left side of the rectangle, there is one column of unselected RAMBs between the rectangle and the pblock\_U4\_peripherals area. On the top and bottom of the pblock\_U5\_mb0\_comp rectangle, there is a row of unselected slices and two unselected DSP48 tiles separating the pblock\_U4\_peripherals area. These unselected resources are the fence between the U5\_mb0\_comp isolated region and the U4\_peripherals isolated region. There are two unused DSP48 tiles on both the top and bottom horizontal fences because for the Spartan-6 device IDF, a minimum of two DSP48 tiles are required for a valid horizontal fence.

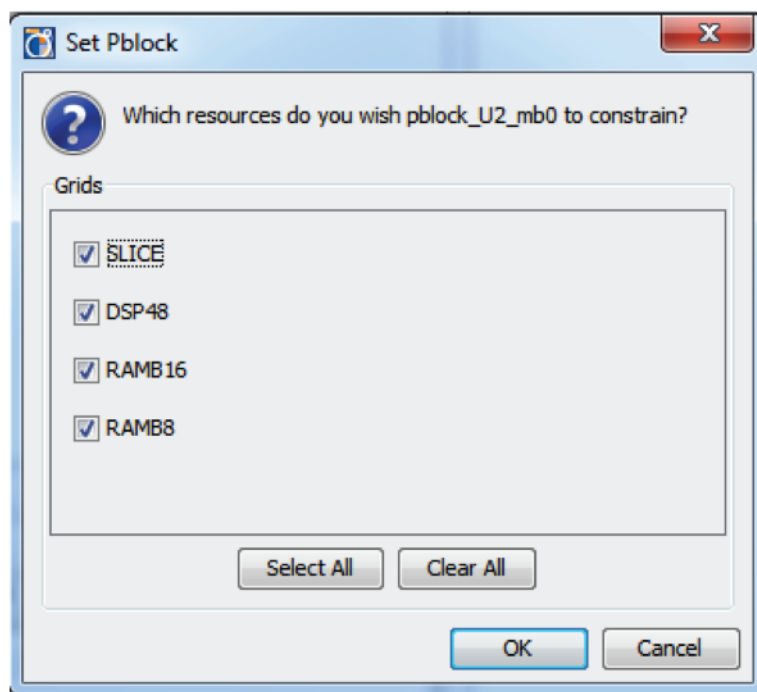
- In the Physical Constraints pane, left-click then right-click the **pblock\_U2\_mb0** pblock and select **Set Pblock Size**.
- In the Device pane, draw a vertical rectangle starting from one SLICE column to the left of the bottom-left of the pblock\_U6\_mb1\_comp rectangle. The vertical rectangle should be drawn up to within one SLICE column and one SLICE row of the pblock\_U4\_peripherals area in the top left quadrant of the device (see Figure 54).



X584\_54\_040412

Figure 54: Drawing the Pblock Rectangle for pblock\_U2\_mb0

7. In the Set Pblock window, make sure that all the resources are selected (Figure 55). Click **OK**.

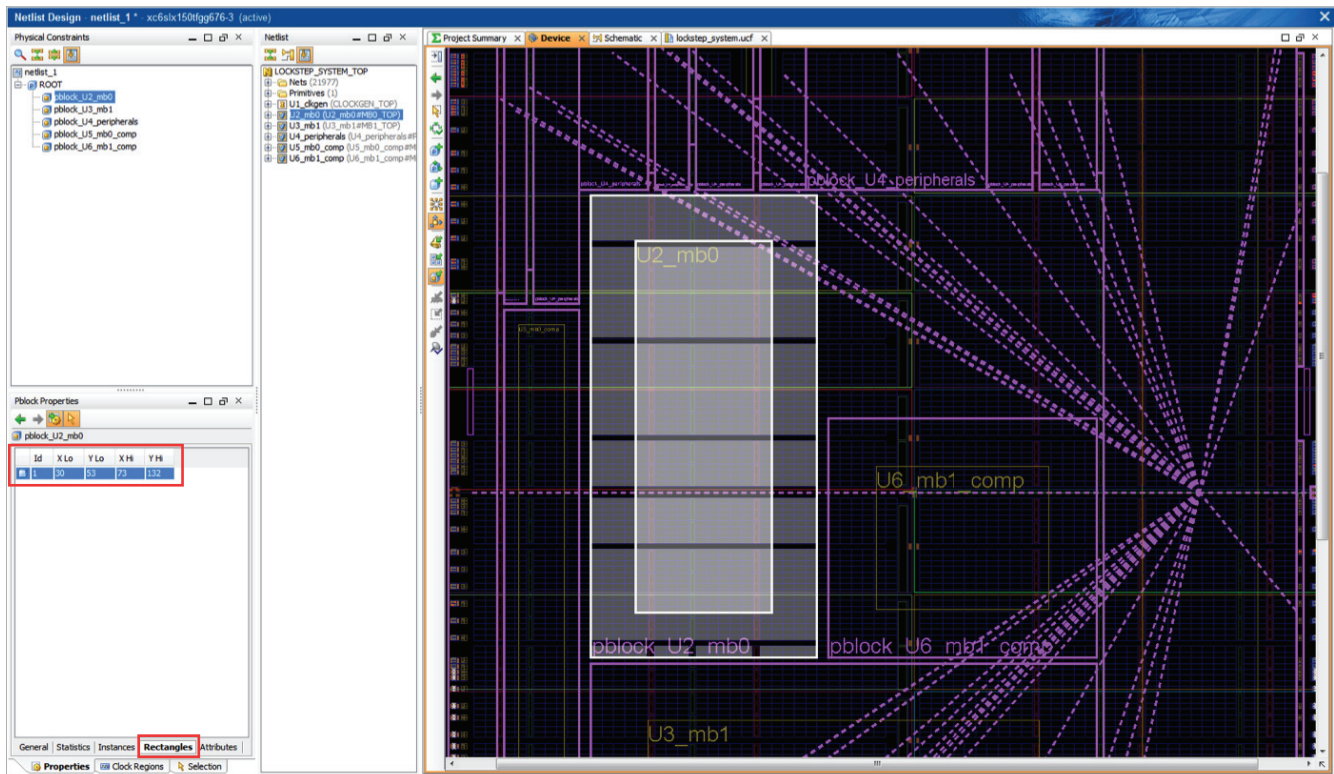


X584\_55\_040412

Figure 55: Resources for pblock\_U2\_mb0

A highlighted rectangle appears, indicating that the device resources have been assigned to the pblock associated with the U2\_mb0 partition (Figure 56).

8. Click the Rectangles tab in the Pblock Properties pane and adjust the rectangle through click and drag actions in the Device pane (see Figure 56) until the rectangle settings are:
  - X Lo: **30**
  - Y Lo: **53**
  - X Hi: **73**
  - Y Hi: **132**



X584\_56\_040512

Figure 56: Final First Pblock Rectangle for pblock\_U2\_mb0

9. The remaining empty space is also assigned to pblock\_U3. To add to the floorplan for pblock\_U2\_mb0, in the Physical Constraints pane, left-click then right-click the **pblock\_U2\_mb0 pblock** and select **Add Pblock Rectangle**.
10. In the Device pane, draw a horizontal rectangle starting from the top left of the rectangle just drawn (see Figure 57). The horizontal rectangle should be drawn up to within one SLICE row and one DSP48 column of the pblock\_U4\_peripherals area in the top right quadrant of the device and within one slice row of the pblock\_U6\_mb1\_comp rectangle.



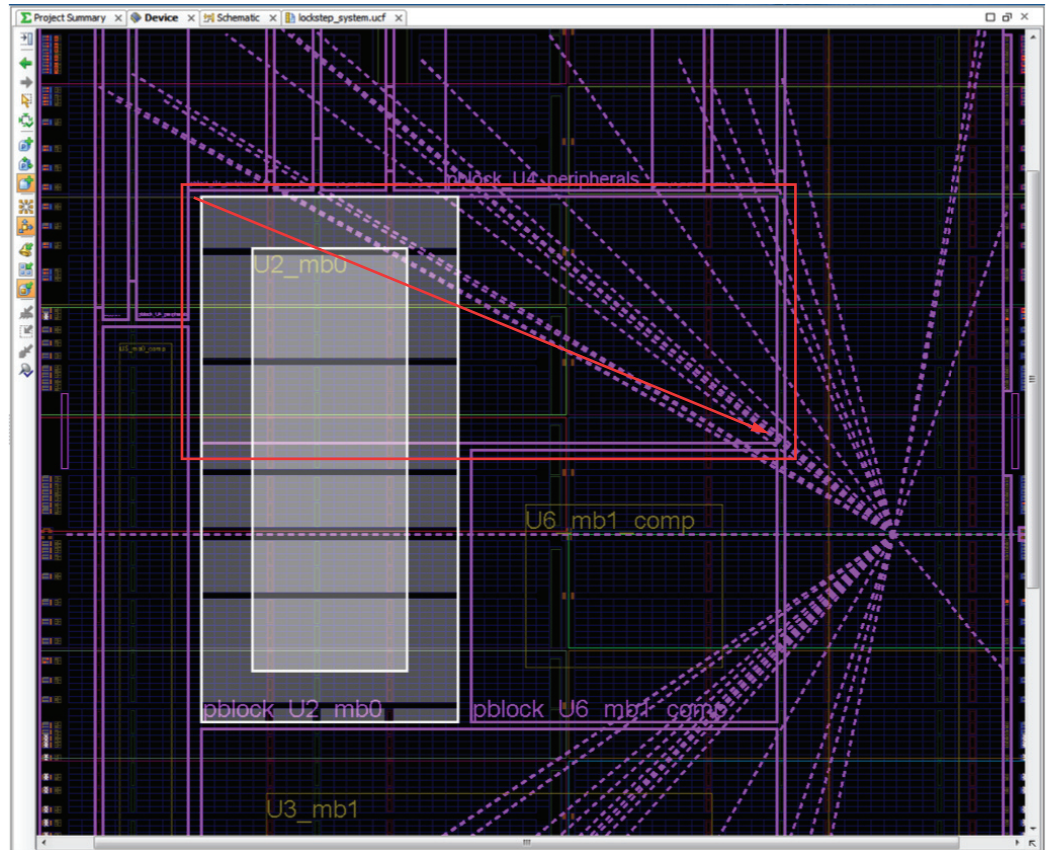
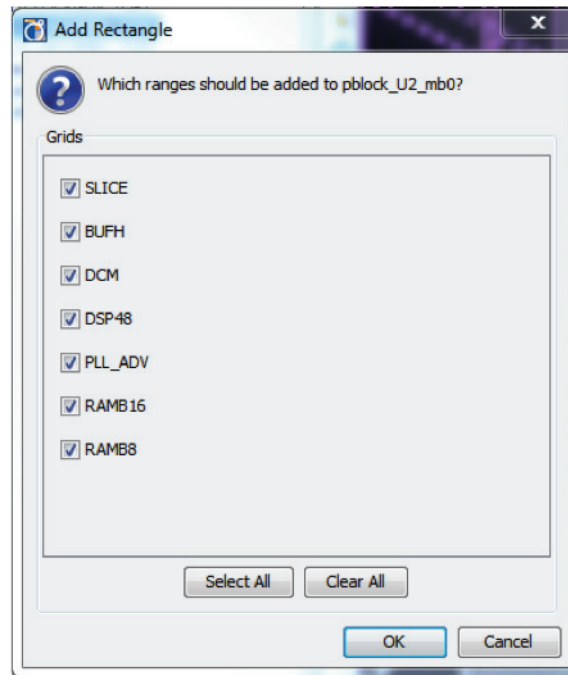


Figure 57: Drawing the Second Pblock Rectangle for pblock\_U2\_mb0

11. In the Set Pblock window (Figure 58), make sure that all the resources are selected. Click **OK**.



X584\_58\_040512

*Figure 58: Resources for the pblock\_U2\_mb0 Second Rectangle*

A second highlighted rectangle appears, indicating that additional device resources have been assigned to the Pblock associated with the U2\_mb0 partition (Figure 59).

12. Click the Rectangles tab in the Pblock Properties pane and adjust the rectangle through click-and-drag actions in the Device pane (Figure 59) until the rectangle settings for Rectangles 1, 2, and 3 match the listed settings:

Rectangle 1:

- X Lo: **91**
- Y Lo: **53**
- X Hi: **122**
- Y Hi: **90**

Rectangle 2:

- X Lo: **88**
- Y Lo: **53**
- X Hi: **89**
- Y Hi: **86**

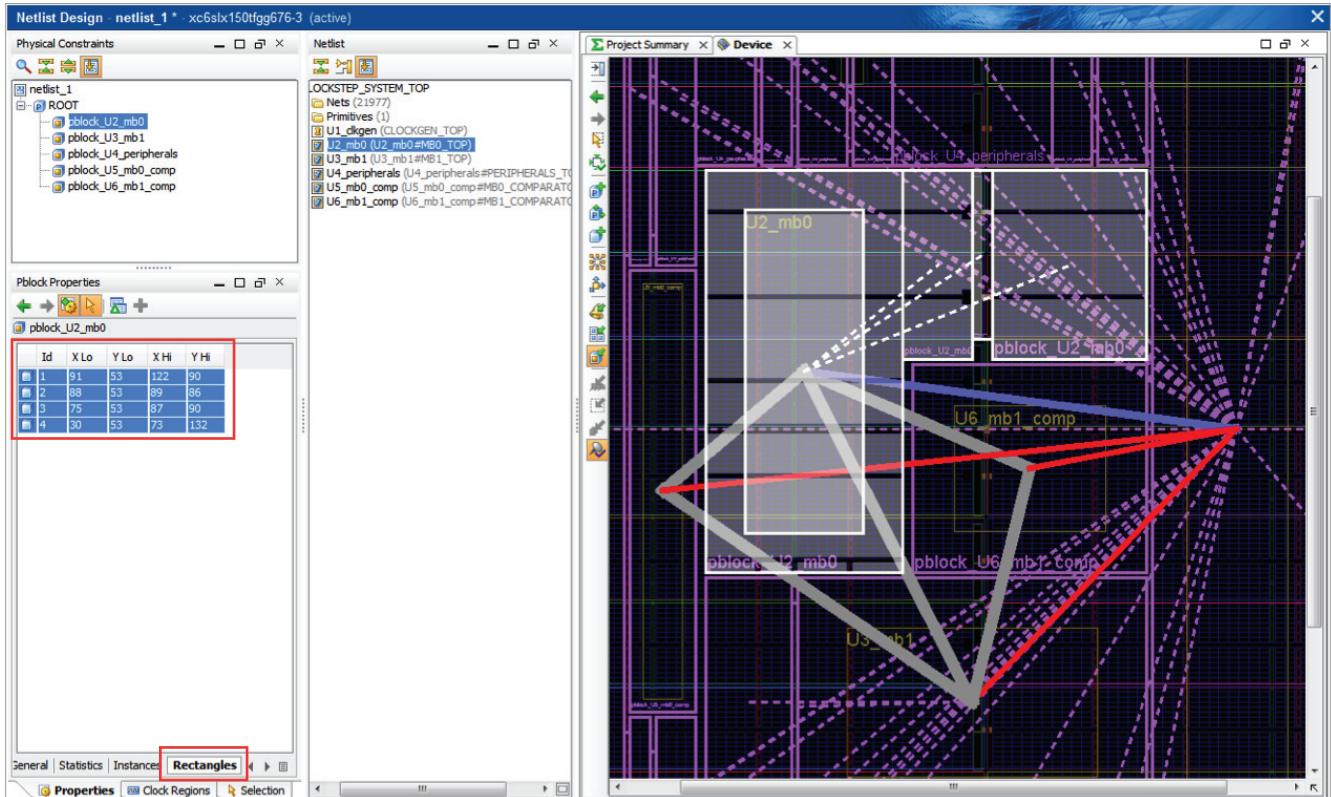
Rectangle 3:

- X Lo: **75**
- Y Lo: **53**
- X Hi: **87**
- Y Hi: **90**



Rectangle 4 settings should remain as was previously drawn.

- X Lo: 30
- Y Lo: 53
- X Hi: 73
- Y Hi: 132



X584\_59\_040512

Figure 59: Final Floorplan for pblock\_U2\_mb0

13. Save the updated floorplan and the PlanAhead tool project by selecting **File > Save Design** from the menu bar.

When the floorplan is saved, the target UCF is updated with AREA RANGE constraints based on the pblock rectangles drawn in the Device pane.

### Discussing the Provided Floorplans

The floorplan for the U3\_mb1, U4\_peripherals, and U6\_mb1\_comp isolated functions were provided through the `lockstep_system.ucf` file provided with the reference design. In this section, the methodology of generating the provided floorplans is discussed.

#### U3\_mb1 Floorplan

The U3\_mb1 floorplan (Figure 60) is initially generated as a simple horizontal rectangle. When drawing the simple horizontal rectangle, the DSP48 in the upper right of the rectangle (DSP48\_X1Y16) gets included. For isolation in Spartan-6 devices, a minimum of two DSP48 user tiles is required for a valid horizontal fence. This DSP48 needs to be excluded from the

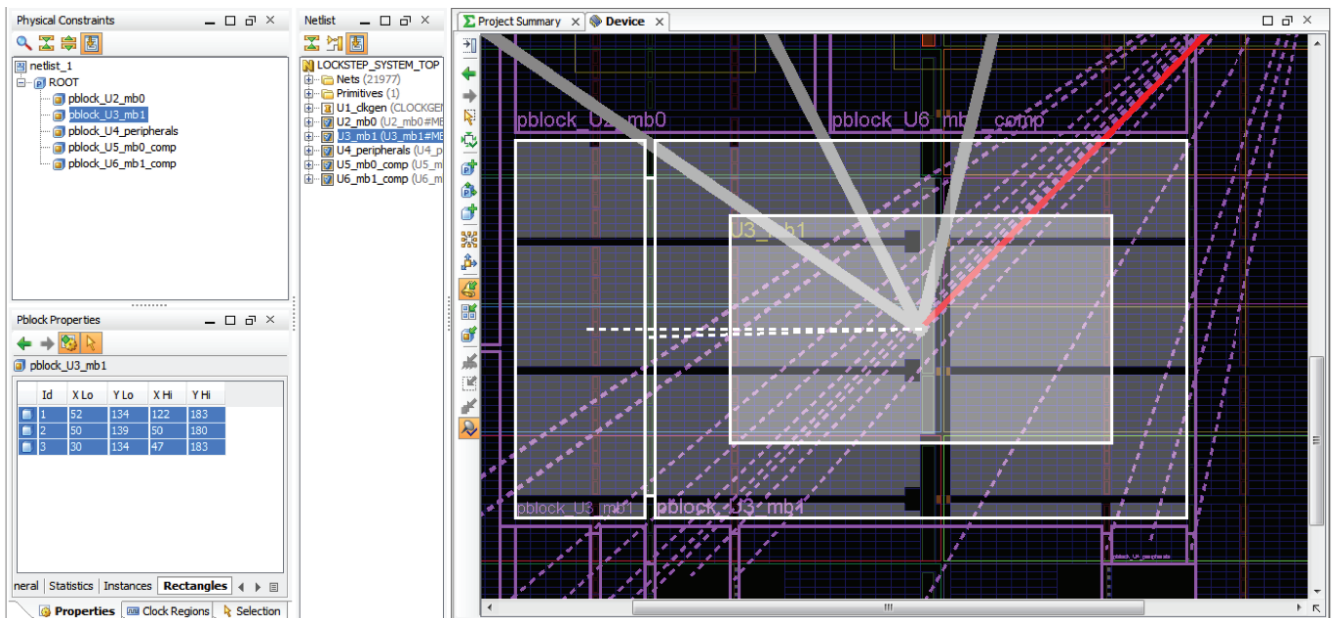
floor plan. To do this, the simple rectangle is drawn and the UCF is then edited to change this constraint:

```
AREA_GROUP "pblock_U3_mb1" RANGE=DSP48_X1Y6:DSP48_X1Y16;
```

to:

```
AREA_GROUP "pblock_U3_mb1" RANGE=DSP48_X1Y6:DSP48_X1Y15;
```

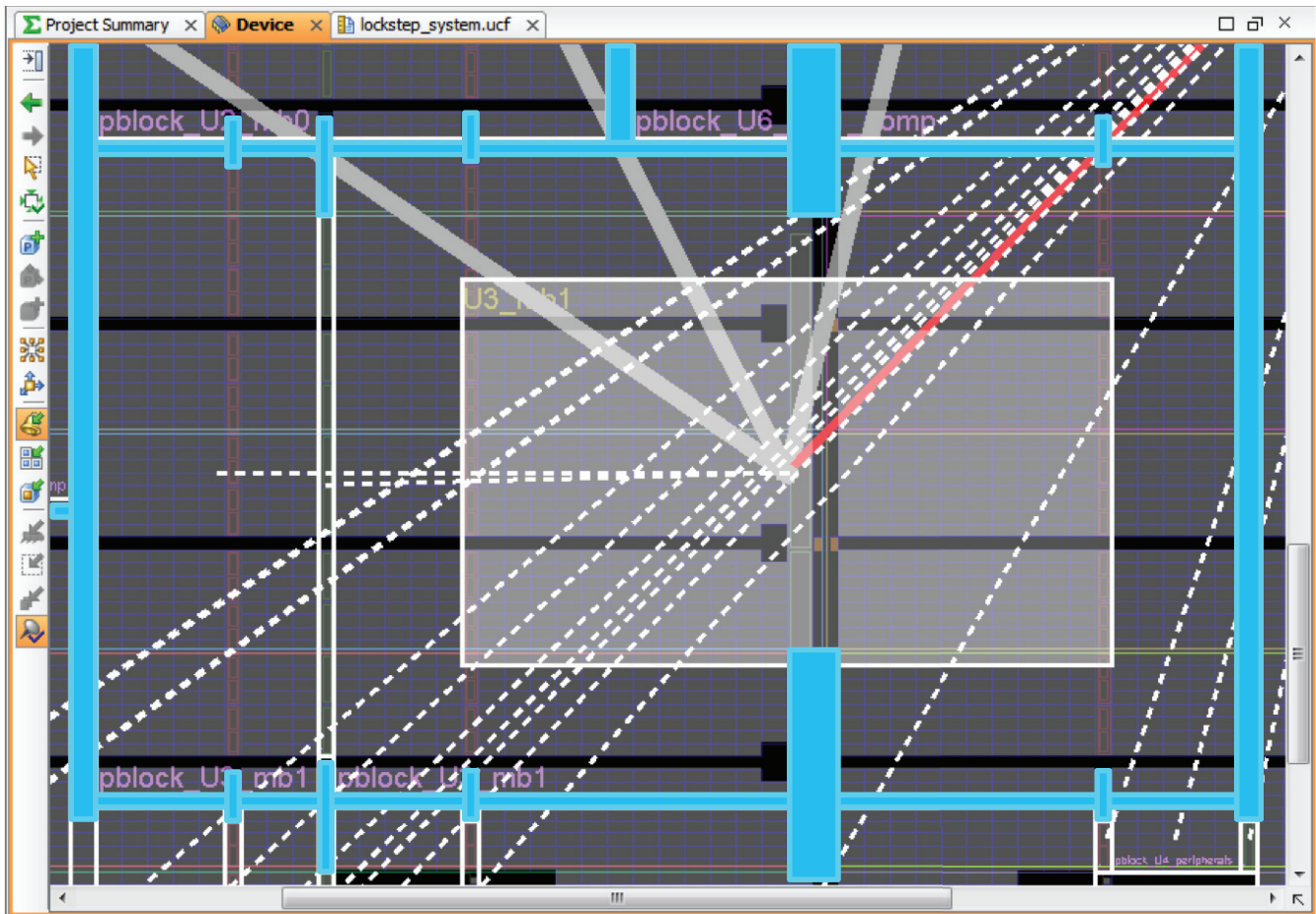
Changing the constraint removes the DSP48 user tile and creates a valid horizontal fence. DSP48\_X1Y17 gets excluded when the floorplan for U2\_mb0 is drawn due to the manner in which the rectangle crosses the tile.



X584\_60\_040512

Figure 60: U3\_mb1 Floorplan

On the left side of the U3\_mb1 floorplan, a single column of SLICE user tiles is used to implement the vertical fence between U3\_mb1 and U4\_peripherals. On the right side, a single column of DSP48 user tiles is used to implement the vertical fence between U3\_mb1 and U4\_peripherals. Separating U3\_mb1 from U2\_mb0 is a single row of SLICE user tiles, two DSP48 user tiles, and in two columnar locations, a single RAMB user tile. Separating U3\_mb1 from U6\_mb1\_comp is a single row of SLICE user tiles, a single DCM user tile, and a single RAMB user tile. On the bottom side, separating U3\_mb1 from U4\_peripherals is a single row of SLICE user tiles, three columnar instances of a single RAMB user tile, two DSP48 user tiles, and one PLL\_ADV user tile. All fences were generated using the required minimum user tiles for each user tile type for both vertical and horizontal fences. Figure 61 shows the fence surrounding U3\_mb1 highlighted in blue.



X584\_61\_041412

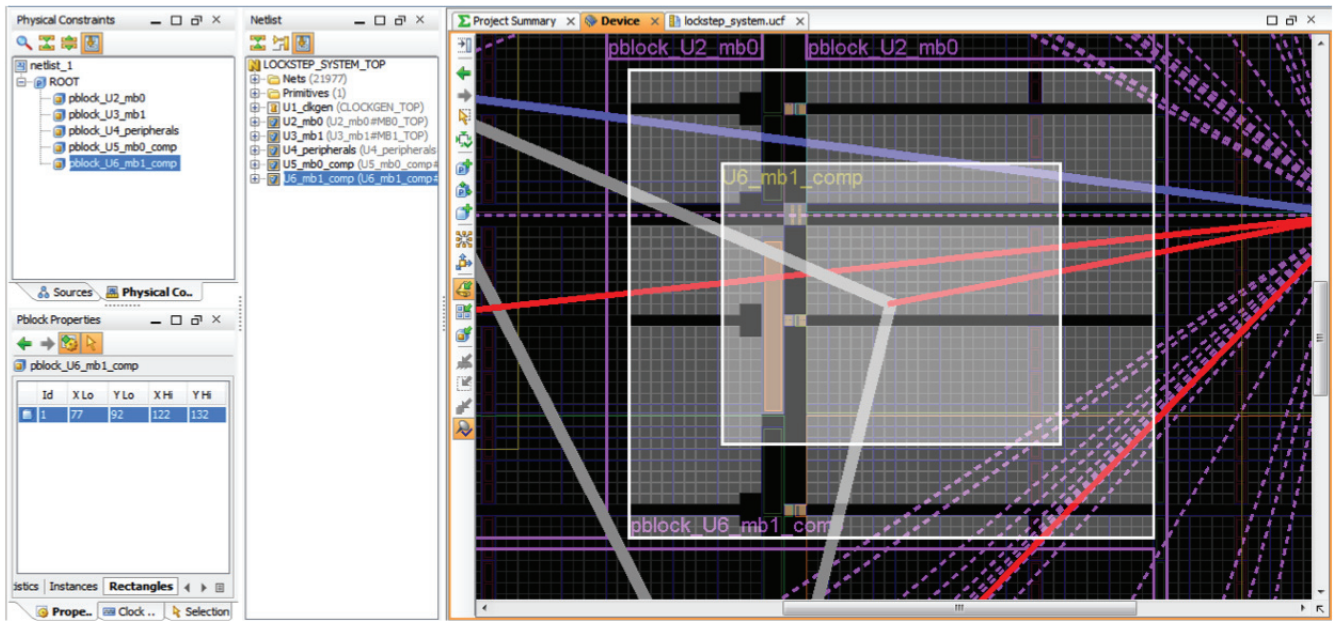
Figure 61: U3\_mb1 Fence Highlighted in Blue

The area group constraints for pblock\_U3\_mb1 show that BUFH constraints have not been included. They were excluded because they are not a resource that is required for implementing U3\_mb1.

#### **U6\_mb1\_comp Floorplan**

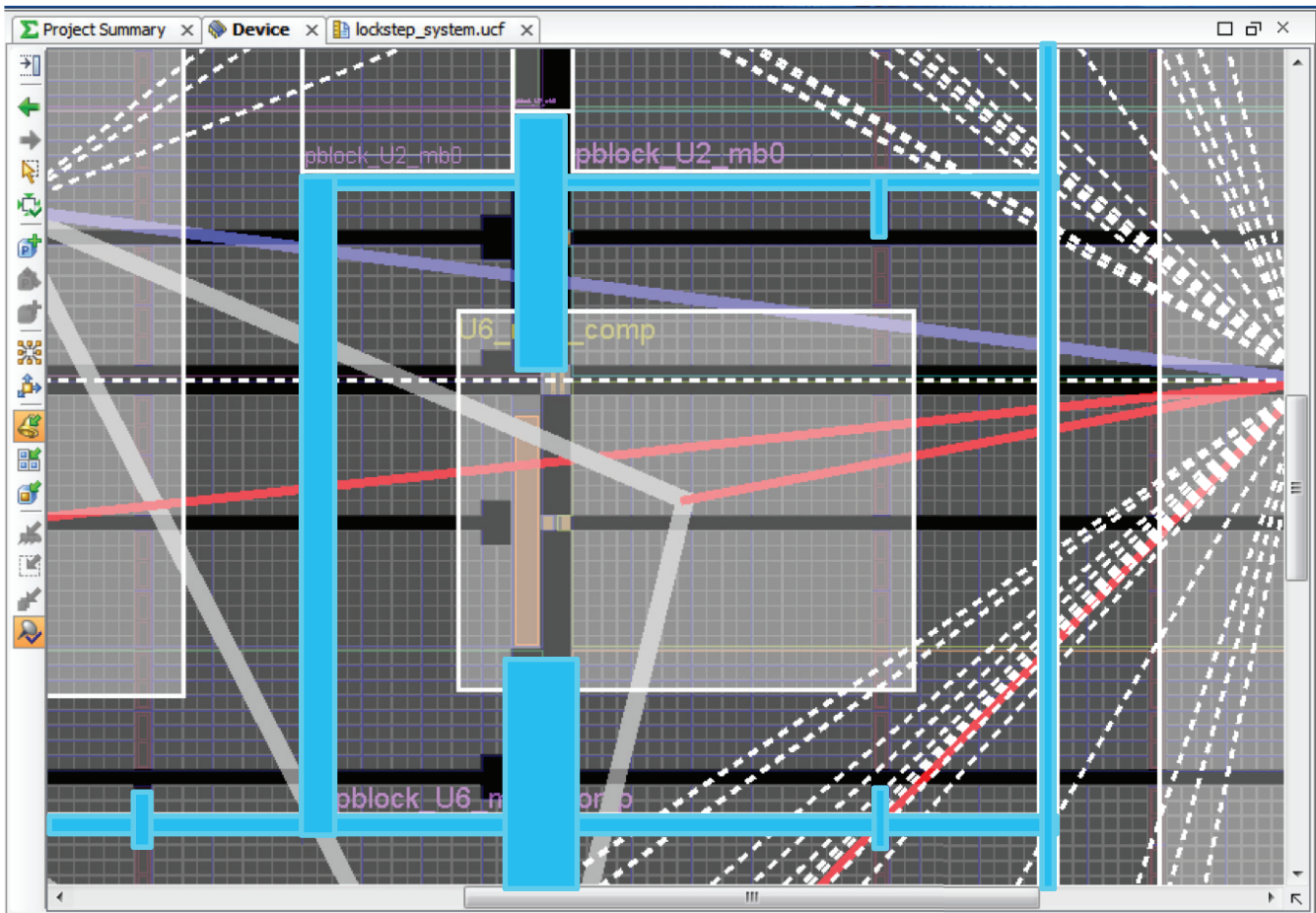
The U6\_mb1\_comp floorplan (Figure 62) is drawn as a simple rectangle. Given its location, it naturally selects resources that fulfill the minimum user tile separations for a fence. On the left side of the U6\_mb1\_comp floorplan, a single column of SLICE user tiles is used to implement the vertical fence between U6\_mb1\_comp and U2\_mb0. On the right side, a single column of DSP48 user tiles is used to implement the vertical fence between U6\_mb1\_comp and U4\_peripherals. On the top and bottom side, for the horizontal fences, a single row of SLICE user tiles, a single RAMB user tile, and a single DCM user tile are used. Figure 63 shows the fence surrounding U6\_mb1\_comp highlighted in blue.





X584\_62\_040512

Figure 62: U6\_mb1\_comp Floorplan



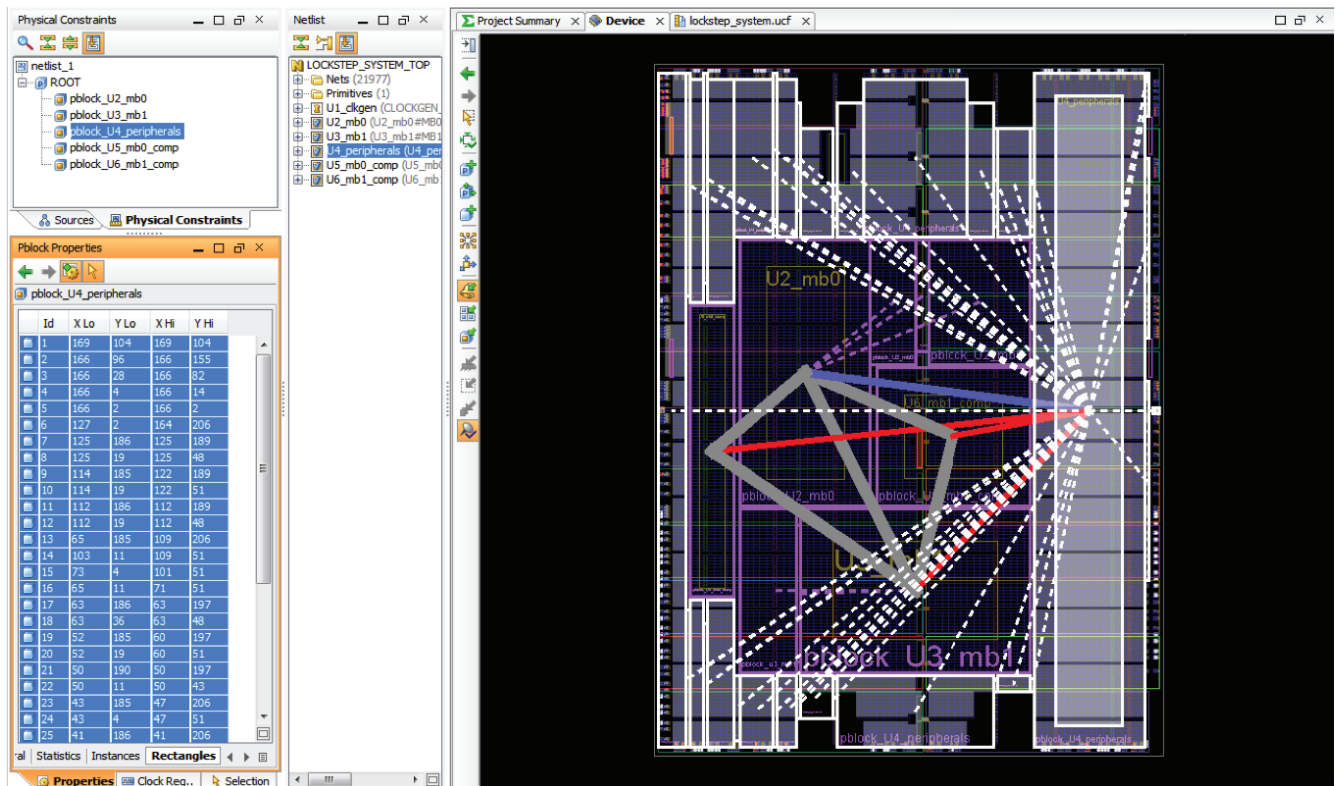
X584\_63\_041412

Figure 63: U6\_mb1\_comp Fence Highlighted in Blue

The area group constraints for pblock\_U6\_mb1\_comp show that the BUFH and BUFGMUX constraints have been included as resources available to implement U6\_mb1\_comp. While the BUFGMUX is not required to implement U6\_mb1\_comp, it is required to implement the global clocking for the overall design. For IDF, global logic must logically be owned at the top level, but must be physically owned by an area group so that the global logic can be implemented. The PLL\_ADV used by the CLOCKGEN module is locked through placement constraints to be physically owned by the U6\_mb1\_comp isolated region for this design. The BUFH was included in the area group because removing it brought no added benefit, even though it is not required to implement U6\_mb1\_comp.

### U4\_peripherals Floorplan

For this design, the U4\_peripherals floorplan (Figure 64) was the most difficult to develop. All the I/O for the design was originally owned by the U4\_peripherals isolated function, except for the two comparator error outputs. It became too difficult to route the design given the existing pinout of the Avnet Spartan-6 FPGA LX150T development board if these two I/O were not also included in the U4\_peripherals isolated function. With this, the HDL was updated so that all the I/O were within the U4\_peripherals isolated function.



X584\_64\_040512

Figure 64: U4\_peripherals Floorplan

To draw the floorplan, a pblock rectangle was added on the left side of the device parallel to what is now the pblock for U5\_mb0\_comp. This pulled in the ILOGIC, OLOGIC, IODELAYS, memory controller blocks (MCBs), BUFIO, BUFIO2, BUFPLL, and BUFPLL\_MCB instances on the left side of the device into the area group. The MCB that is mapped to the DDR3 pinout is the MCB on the upper left of the device, which also needed the BUFPLL\_MCB on the left side of the device for clocking. The I/O components on the upper left side of the device were also required for the DDR3 interface, along with I/O components toward the middle of the left side for the RS-232 interface. To ease floorplanning, all instance types on the left side of the device were included in the U4\_peripherals floorplan.

A second pblock rectangle was added across the top of the device, excluding the ILOGIC, OLOGIC, and IODELAYS across the top. The I/O components across the top were excluded for two reasons. First, there are no I/Os across the top used in the design. Second, there is a limitation in the way that the Xilinx software calculates overlapping area group ranges for the I/O components. Leaving a single or group of excluded I/O components prevents the Xilinx software from calculating a larger area group for the I/O than is actually used. This method prevents the tools from reporting erroneous area group overlaps. See *Developing Secure Designs with the Spartan-6 Family Using the Isolation Design Flow* [Ref 2] for further information regarding this limitation with the Xilinx tools and the workaround.

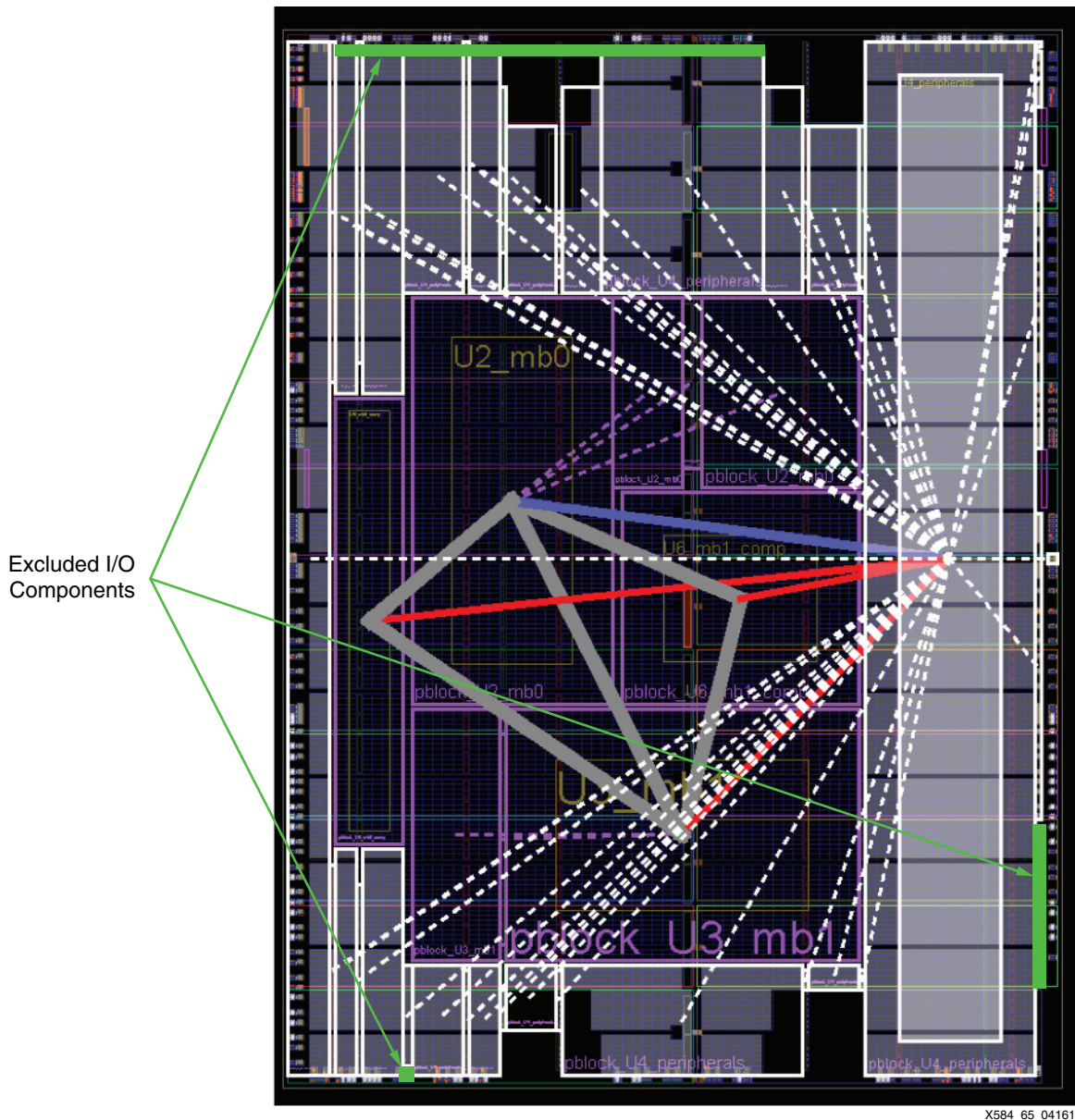
For this second pblock rectangle, the BUFIO2, BUFIO2FB, BUFPLL, BUFPLL\_MCB, BUFDS, IPAD, OPAD, GTPA1\_DUAL, and PCIE\_A1 components located on the top of the device were also excluded from the area group to prevent unintentional area group overlaps. They also were not required for the design. The GTPA1\_DUAL, IPAD, OPAD, and PCIE components are completely excluded for the U4\_peripherals area group because they are not required for the design.

A third pblock rectangle was added on the right side of the device parallel to what are now the pblocks for U2\_mb0, U6\_mb1\_comp, and U3\_mb1. The pblock was drawn to include the I/O components on the top and bottom right of the device but initially excluded all the I/O components on the right side of the device, along with the MCBs, BUFIO2s, BUFIO2FBs, BUFPLLs, BUFPLL\_MCBs, and BSCANs. The components were initially excluded so that specific instances could be included later.

A series of pblocks were added after this to include a series of components on the right side of the device. One pblock was added to include BSCAN\_X0Y1 and BSCAN\_X0Y0. Another pblock was added to include ILOGIC, OLOGIC, and IODELAY components from X35Y171 down to X35Y46. This leaves a gap of excluded I/O components down the remainder of the right side of the device, which prevents the Xilinx tools from reporting erroneous area group overlaps. A pblock was also added to include the BUFIO2, BUFIO2FB, BUFPLL, and BUFPLL\_MCB on the right side of the device. The MCBs on the right side of the device are completely excluded because they are not required for this design.

Finally, a series of pblocks were added to pull in the U4\_peripherals isolated region to within the minimum required user tiles for a fence. It was determined that because the U4\_peripherals floorplan was already an irregular shape, this floorplan would be adjusted to include the minimum of two unused DSP48 user tiles for a valid horizontal fence, wherever possible. This same thinking was also applied to floorplanning RAMB tiles for the fence. This meant manually dragging and resizing the small rectangles that the PlanAhead tool automatically created to define the irregular floorplan shape. Also, on the bottom side of the device, IODELAY, ILOGIC, and OLOGIC X8Y0 to X8Y3 were excluded to provide a gap, preventing the Xilinx tools from reporting unintentional area group overlaps. For completeness, [Figure 65](#) shows the locations of the I/O components excluded from the U4\_peripherals area group.





**Figure 65: U4\_peripherals Floorplan Excluded I/O Components**

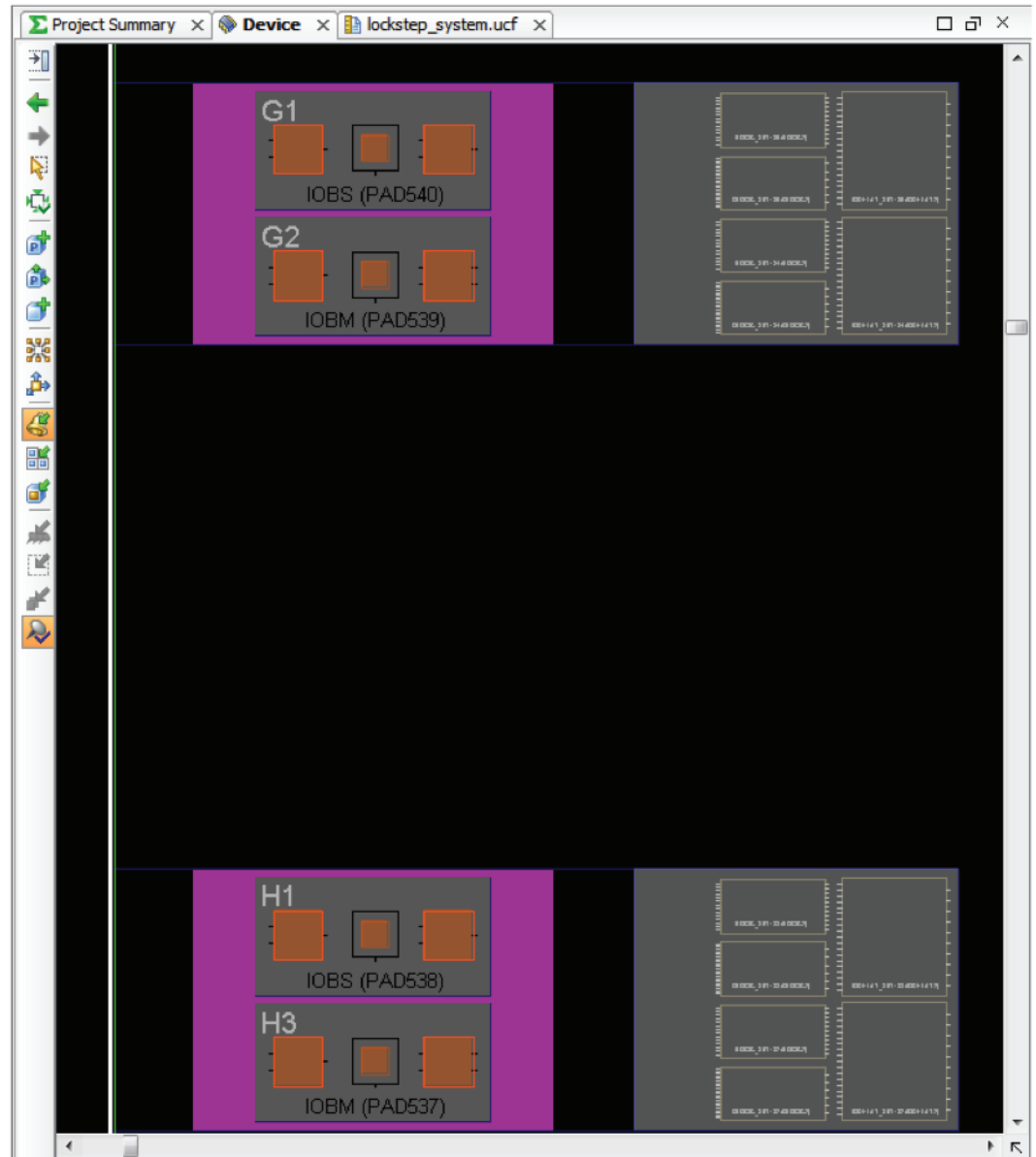
These components were completely excluded from the floorplan because they were not required for the design:

- ICAP
- SPI\_ACCESS
- SUSPEND\_SYNC
- POST\_CRC\_INTERNAL
- STARTUP
- SLAVE\_SPI
- DNA\_PORT
- GTPA1\_DUAL
- PCIE\_A1



- IPAD
- OPAD

For Spartan-6 device designs, the pad ranges need to be set manually for the I/O blocks or pads used in the design so that they are included in the isolated region. The PAD range is identified in the UCF with the AREA\_GROUP "pblock\_U4\_peripherals" RANGE=PAD211, PAD190, PAD191, ...; line. The PADXXX numbers correspond to the I/O pad and I/O block used for the pin. The PAD numbers can be found by zooming in on the I/O block in the PlanAhead tool's Device pane, as shown in [Figure 66](#). The PAD number is listed in parentheses. Each pad used in the design must be listed individually in the Range constraint, separated by a comma.



X584\_66\_040512

*Figure 66:* View of PAD in the PlanAhead Tool Device Pane

As previously discussed in this section, there is a limitation in the way that the ISE tools calculate overlapping area group ranges for the I/O components. This limitation also affects the BUFPLLs, BUFPLL\_MCBs, BUFIO2s, and BUFIO2FBs. For the design, only the components located on the left and right side of the device are included. The PlanAhead tool incorrectly

defines the constraints for these components when they are included in the area group, making the check for overlapping area groups think that these components overlap with the U2\_mb0, U3\_mb1, U5\_mb0\_comp, and U6\_mb1\_comp area groups.

On the right side of the device are:

- BUFIO2\_X3Y10, BUFIO2\_X3Y11, BUFIO2\_X3Y12, BUFIO2\_X3Y13, BUFIO2\_X4Y18, BUFIO2\_X4Y19, BUFIO2\_X4Y20, BUFIO2\_X4Y21
- BUFIO2FB\_X3Y10, BUFIO2FB\_X3Y11, BUFIO2FB\_X3Y12, BUFIO2FB\_X3Y13, BUFIO2FB\_X4Y18, BUFIO2FB\_X4Y19, BUFIO2FB\_X4Y20, BUFIO2FB\_X4Y21
- BUFPLL\_X2Y2, BUFPLL\_X2Y3
- BUFPLL\_MCB\_X2Y5

On the left side of the device are:

- BUFIO2\_X1Y8, BUFIO2\_X1Y9, BUFIO2\_X1Y14, BUFIO2\_X1Y15, BUFIO2\_X0Y16, BUFIO2\_X0Y17, BUFIO2\_X0Y22, BUFIO2\_X0Y23
- BUFIO2FB\_X1Y8, BUFIO2FB\_X1Y9, BUFIO2FB\_X1Y14, BUFIO2FB\_X1Y15, BUFIO2FB\_X0Y16, BUFIO2FB\_X0Y17, BUFIO2FB\_X0Y22, BUFIO2FB\_X0Y23
- BUFPLL\_X0Y2, BUFPLL\_X0Y3
- BUFPLL\_MCB\_X0Y5

The user works around the overlap limitation by listing each area group constraint as separate groups, where the first group defines the components on the right side, and the second group defines the components on the left side separated by a comma.

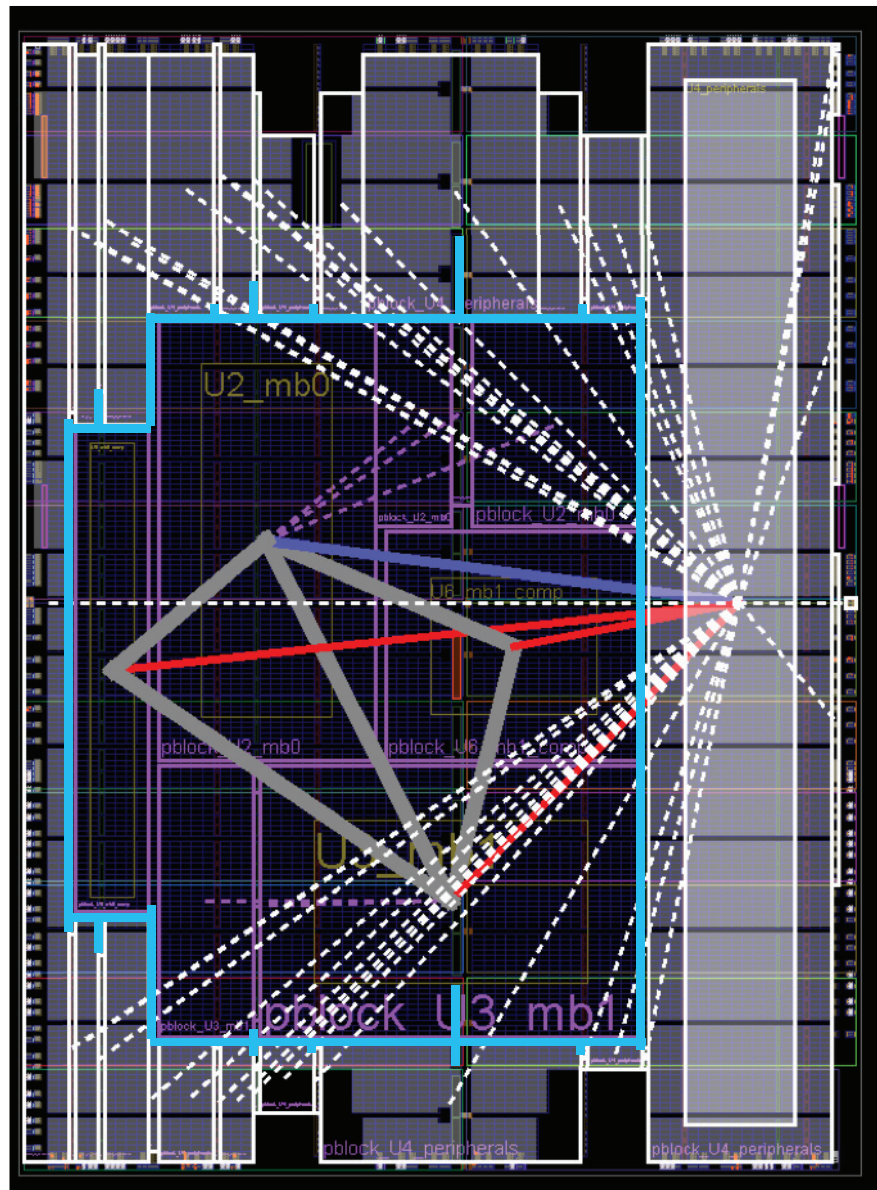
**Note:** The constraints could have been written left side first, then right side.

The constraints in the UCF become:

- AREA\_GROUP "pblock\_U4\_peripherals"  
RANGE=BUFIO2\_X3Y10:BUFIO2\_X4Y21, BUFIO2\_X0Y16:BUFIO2\_X0Y23;
- AREA\_GROUP "pblock\_U4\_peripherals"  
RANGE=BUFIO2FB\_X3Y10:BUFIO2FB\_X4Y21, BUFIO2FB\_X0Y16:BUFIO2FB\_X0Y23;
- AREA\_GROUP "pblock\_U4\_peripherals"  
RANGE=BUFPLL\_X2Y3:BUFPLL\_X2Y3, BUFPLL\_X0Y2:BUFPLL\_X0Y3;
- AREA\_GROUP "pblock\_U4\_peripherals"  
RANGE=BUFPLL\_MCB\_X2Y5:BUFPLL\_MCB\_X2Y5, BUFPLL\_MCB\_X0Y5:BUFPLL\_MCB\_X0Y5;

This workaround to the overlap limitation forces the tools to know that the components are defined in two smaller pblock rectangles instead of one large pblock rectangle that spans the whole device. The UCF constraint for the BUFPLL shows that BUFPLL\_X2Y2 is missing. That is because with it included, the PlanAhead tool removes the constraint segments and changes the constraint to RANGE = BUFPLL\_X0Y2:BUFPLL\_X2Y3, which causes overlap errors. Basically, removing BUFPLL\_X2Y2 puts a gap in the constraint.

Figure 67 shows the fence separating U4\_peripherals from the rest of the isolated functions highlighted in blue.



X584\_67\_041412

Figure 67: U4\_peripherals Fence Highlighted in Blue

**Note:** The user should start floorplanning any isolated design as early as possible after the pinout and inter-isolated function communication is known. For a design that uses IDF, the user should also plan the design pinout to make floorplanning easier. Also, floorplanning can be an iterative process, requiring small to medium tweaks when it comes to executing final routing and meeting timing.

### Verifying the Floorplans are Large Enough

The user might question if the assigned pblocks are large enough to implement the isolated function. The PlanAhead tool provides a quick pre-implementation check to indicate, based on the required component count, if the pblock is large enough.

To verify that the pblock associated with each isolated function is large enough, these steps should be performed for each isolated function:

1. In the Physical Constraints pane, click the pblock that you want to check.
2. In the Pblock Properties pane, click the **Statistics** tab.

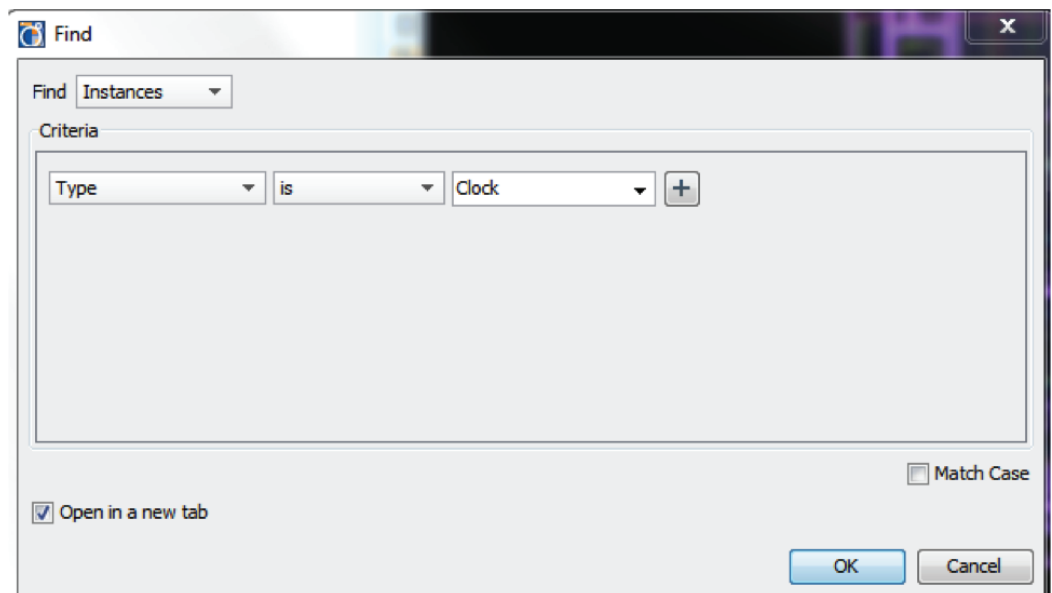
- In the Statistics tab, there is a section entitled Physical Resource Estimates, indicating the percentage of the resources within the pblock that the isolated function consumes, excluding routing resources. If the percentage exceeds 100%, the Site Type row turns red.

### Placing Design Components

For the reference design, three components are locked down to specific sites within the device. The three components are the PLL used for the overall design clock generation, a BUFPLL\_MCB that provides the clocking for the DDR3 memory controller, and the memory controller itself. By locking down these components, the implementation time is shortened and the PLL is physically owned by an isolated region.

These steps describe assigning the three component instances to specific device sites:

- Click in the Device pane. Press **Ctrl+F** to begin a search.
- Change the Find to **Instances**, and the Criteria to **Type is Clock** (Figure 68).



X584\_68\_040512

Figure 68: Find Instance Clock

- Click **OK**.
- A Find Results pane appears at the bottom of the PlanAhead tool window listing the BUFGs, BUFPLL\_MCB, and PLL\_ADV found in the netlist. In the Find Results pane, click the **BUFPLL\_MCB**.
- With **BUFPLL\_MCB** selected, go to the Instance Properties pane, and click the **Attributes** tab (Figure 69).



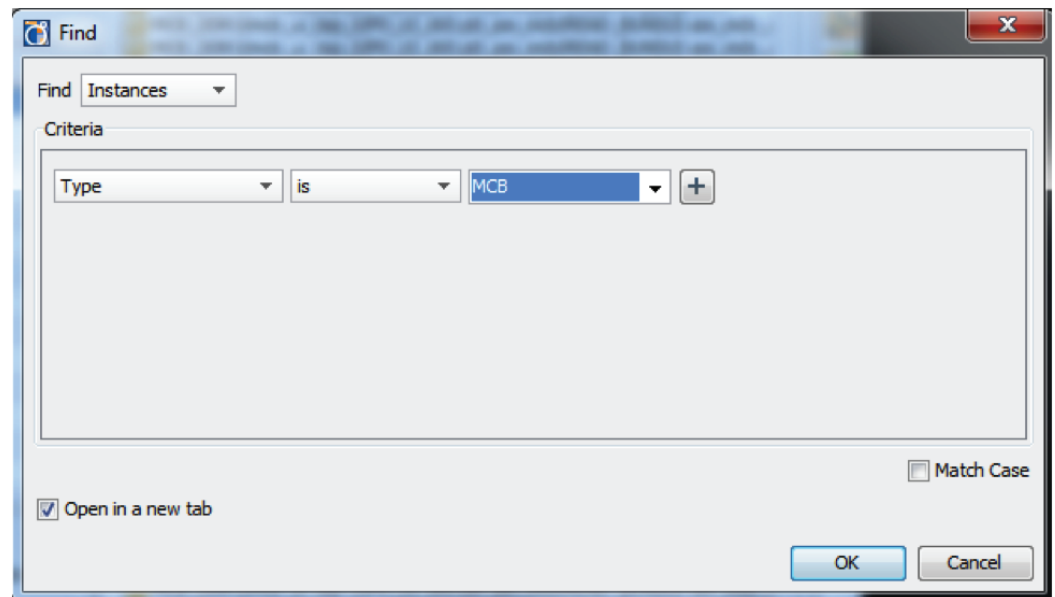
The screenshot shows the Xilinx ISE Netlist Design tool interface. The main window displays the netlist design for 'netlist\_1'. The Instance Properties window is open for the instance 'MCB\_DDR3/mcb\_ui\_top\_0/gen\_spartan6\_bufpll\_mcb.bufpll\_0'. The 'Attributes' tab is selected, and the 'LOC\*' attribute is highlighted in red, with the value 'BUFPLL\_MCB\_X0Y5' entered. The 'Apply' button is also highlighted. Below the Instance Properties window, the 'Find Results - Instances - Type is 'Clock' (5)' window is open, displaying a table of instances:

Id	Name	Cell	Pins	Partition
1	BUFGEN_DRCK	BUFGEN	2	Top
2	U4_peripherals/MCB_DDR3/MCB_DDR3/mcb_ui_top_0/gen_spartan6_bufpll_mcb.bufpll_0	BUFPLL_MCB	9	U4_peripherals
3	U1_clkgen/clock_generator_0/clock_generator_0/PLL0_CLKFBOUT_BUFGEN_INST	BUFGEN	2	Top
4	U1_clkgen/clock_generator_0/clock_generator_0/PLL0_CLKOUT2_BUFGEN_INST	BUFGEN	2	Top
5	U1_clkgen/clock_generator_0/clock_generator_0/PLL0_INST/Using_PLL_ADV.PLL_ADV_inst	PLL_ADV	62	Top

X584\_69\_040512

Figure 69: Setting LOC Attribute for BUFPLL\_MCB

- Click the green cross within the Attributes tab, and add the **LOC** Attribute to the BUFPLL\_MCB (Figure 69).
- Set the **LOC** attribute to **BUFPLL\_MCB\_X0Y5** to assign the instance to the site. Click **Apply** to set the attribute assignment (Figure 69).
- Repeat step 4 through step 7 to lock the **PLL\_ADV** instance to **PLL\_ADV\_X0Y2**.
- Click in the **Device** pane. Press **Ctrl+F** to begin a search.
- Change the Find to **Instances**, and the Criteria to **Type is MCB** (Figure 70).



X584\_70\_040512

Figure 70: Find Instance MCB

11. In the Find Results pane, click the **MCB**.
12. With the **MCB** selected, through the Attributes tab in the Instance Properties pane add the **LOC** attribute and set it to **MCB\_X0Y3**, which is the MCB tied to the DDR3 pins.
13. Click **Apply**.
14. Click **Save** in the PlanAhead window to add the Instance **LOCs** to the `lockstep_system.ucf` file.

## Run IVT on the Design in UCF Mode

Figure 71 shows the reference design progress to this point.



<input checked="" type="checkbox"/>	1. Generate Dual MicroBlaze System in EDK Platform Studio.
<input checked="" type="checkbox"/>	2. Modify Dual MicroBlaze System into Lockstep through EDK Platform Studio.
<input checked="" type="checkbox"/>	3. Build the Dual-Lockstep MicroBlaze System in EDK Platform Studio.
<input checked="" type="checkbox"/>	4. Perform a Quick Sanity Check of the Design.
<input checked="" type="checkbox"/>	5. Prepare Dual-Lockstep MicroBlaze Design Source for Isolated Design.
<input checked="" type="checkbox"/>	6. Synthesize and Floorplan Hierarchical Design.
	7. Run IVT on the Design in UCF Mode.
	8. Implement the Design.
	9. Run IVT on the Design in NCD Mode.
	10. Build Final Software in SDK.
	11. Disconnect the MicroBlaze Processors and Debug Logic.
	12. Re-verify the Re-implemented Design in IVT.

X584\_71\_041112

**Figure 71: Reference Design Progress**

The IVT software is a command line tool that verifies that an FPGA design partitioned into isolated regions and functions meets stringent standards for fail-safe design. The IVT is used at two stages in the IDF. Early in the flow, the IVT is used to perform a series of design rule checks on floorplans and pin assignments. Use of the IVT at this stage in the flow is optional but highly recommended. The goal of UCF checking is to identify potential isolation problems before commitment to board layout. After the design is complete, the IVT is used again on the NCD to validate that the required isolation is built into the design. The use of the IVT on the NCD is mandatory in the flow to verify isolation.

At this stage in the application note, the IVT is run on the `lockstep_system.ucf` file in what is known as UCF mode. The IVT in UCF mode checks these conditions:

- Pins from different isolation regions are not physically adjacent, vertically or horizontally, at the die.
- Pins from different isolation regions are not physically adjacent at the package. Adjacency is defined in eight compass directions: north, northwest, west, southwest, south, southeast, east, and northeast.
- Pins from different isolation regions are not co-located in an I/O block.  
While the IVT does not fault such conditions, the real rule is application-specific and whether or not it is a fault depends on the program.
- The `AREA_RANGE` constraints are defined such that the minimum user tile for a fence is defined between isolated regions.

The files to run the IVT on the design are provided as part of the application note. Subsequent sections contain steps to install and run the IVT on the design's UCF and describe the structure of the input and output files.

### Installing the IVT

The IVT executable, version 7.08, is included within the `<reference design>\ivt` directory.

1. In the `<reference design>\ivt` directory, extract the file `ivt_7_08_nt.zip` to the `<reference design>\ivt` directory.

2. Navigate down to <reference design>\ivt\ivt\_7\_08\_nt\bin\_nt\13.4\nt and copy ivt.exe to the 32-bit binary executables directory of the ISE tools 13.4 installation directory (usually located at C:\Xilinx\13.4\ISE\_DS\ISE\bin\nt) to install the 32-bit Windows version of the IVT.
3. Navigate down to <reference design>\ivt\ivt\_7\_08\_nt\bin\_nt\13.4\nt64 and copy ivt.exe to the 64-bit binary executables directory of the ISE tools 13.4 installation directory (usually located at C:\Xilinx\13.4\ISE\_DS\ISE\bin\nt64) to install the 64-bit Windows version of the IVT.

### Executing the IVT in UCF Mode

When running the IVT in UCF mode, two files are required to run the tool. The first is the pin isolation group (PIG) file. The PIG file uses the UCF syntax, so it can be copied directly from the UCF to define which pins go with which isolated function. The pins for each isolated function are listed in the format NET "Net Name" LOC = Pin Number; and bracketed with this formatted statement:

```
ISOLATION_GROUP Isolated Function Instance Name BEGIN
  NET "Net Name" LOC = Pin Number;
  NET "Net Name" LOC = Pin Number;
END ISOLATION_GROUP
```

All isolation functions must be listed, even if they do not have any pin I/O within the isolated function.

**Note:** The global clock input pin is not part of an isolated function so it does not have to be included in the PIG file.

The lockstep\_system.pig file for the dual-lockstep MicroBlaze processor system is located at <reference design>\ivt\ucf and has the following contents. All the pins are part of the U4\_peripherals isolated function.

```
# Place all Global (top level) signals here (each commented out)
#NET "U1_clkgen/EXTERN_CLK_IN" LOC = U23;

#U2_mb0 pin definitions
ISOLATION_GROUP U2_mb0 BEGIN
END ISOLATION_GROUP

#U3_mb1 pin definitions
ISOLATION_GROUP U3_mb1 BEGIN
END ISOLATION_GROUP

#U4_peripherals pin definitions.
ISOLATION_GROUP U4_peripherals BEGIN
  NET "U4_peripherals/EXTERN_DIP_SWITCHES_8BITS_TRI_I[0]" LOC = K21;
  NET "U4_peripherals/EXTERN_DIP_SWITCHES_8BITS_TRI_I[1]" LOC = G23;
  NET "U4_peripherals/EXTERN_DIP_SWITCHES_8BITS_TRI_I[2]" LOC = G24;
  NET "U4_peripherals/EXTERN_DIP_SWITCHES_8BITS_TRI_I[3]" LOC = J20;
  NET "U4_peripherals/EXTERN_DIP_SWITCHES_8BITS_TRI_I[4]" LOC = J22;
  NET "U4_peripherals/EXTERN_DIP_SWITCHES_8BITS_TRI_I[5]" LOC = E24;
  NET "U4_peripherals/EXTERN_DIP_SWITCHES_8BITS_TRI_I[6]" LOC = E23;
  NET "U4_peripherals/EXTERN_DIP_SWITCHES_8BITS_TRI_I[7]" LOC = K22;
  NET "U4_peripherals/EXTERN_LEDS_8BITS_TRI_O[0]" LOC = M18;
  NET "U4_peripherals/EXTERN_LEDS_8BITS_TRI_O[1]" LOC = L19;
  NET "U4_peripherals/EXTERN_LEDS_8BITS_TRI_O[2]" LOC = M21;
  NET "U4_peripherals/EXTERN_LEDS_8BITS_TRI_O[3]" LOC = F22;
  NET "U4_peripherals/EXTERN_LEDS_8BITS_TRI_O[4]" LOC = H22;
  NET "U4_peripherals/EXTERN_LEDS_8BITS_TRI_O[5]" LOC = C25;
  NET "U4_peripherals/EXTERN_MB0_COMPARATOR_ERROR_OUT" LOC = C26;
  NET "U4_peripherals/EXTERN_MB1_COMPARATOR_ERROR_OUT" LOC = F23;
  NET "U4_peripherals/EXTERN_LINEAR_FLASH_ADDRESS[0]" LOC = L24;
  NET "U4_peripherals/EXTERN_LINEAR_FLASH_ADDRESS[10]" LOC = R19;
```

```
NET "U4_peripherals/EXTERN_LINEAR_FLASH_ADDRESS[11]" LOC = P21;
NET "U4_peripherals/EXTERN_LINEAR_FLASH_ADDRESS[12]" LOC = P22;
NET "U4_peripherals/EXTERN_LINEAR_FLASH_ADDRESS[13]" LOC = R20;
NET "U4_peripherals/EXTERN_LINEAR_FLASH_ADDRESS[14]" LOC = R21;
NET "U4_peripherals/EXTERN_LINEAR_FLASH_ADDRESS[15]" LOC = P24;
NET "U4_peripherals/EXTERN_LINEAR_FLASH_ADDRESS[16]" LOC = P26;
NET "U4_peripherals/EXTERN_LINEAR_FLASH_ADDRESS[17]" LOC = R23;
NET "U4_peripherals/EXTERN_LINEAR_FLASH_ADDRESS[18]" LOC = R24;
NET "U4_peripherals/EXTERN_LINEAR_FLASH_ADDRESS[19]" LOC = T24;
NET "U4_peripherals/EXTERN_LINEAR_FLASH_ADDRESS[1]" LOC = N19;
NET "U4_peripherals/EXTERN_LINEAR_FLASH_ADDRESS[20]" LOC = T26;
NET "U4_peripherals/EXTERN_LINEAR_FLASH_ADDRESS[21]" LOC = V24;
NET "U4_peripherals/EXTERN_LINEAR_FLASH_ADDRESS[22]" LOC = V26;
NET "U4_peripherals/EXTERN_LINEAR_FLASH_ADDRESS[23]" LOC = N17;
NET "U4_peripherals/EXTERN_LINEAR_FLASH_ADDRESS[2]" LOC = N20;
NET "U4_peripherals/EXTERN_LINEAR_FLASH_ADDRESS[3]" LOC = N21;
NET "U4_peripherals/EXTERN_LINEAR_FLASH_ADDRESS[4]" LOC = N22;
NET "U4_peripherals/EXTERN_LINEAR_FLASH_ADDRESS[5]" LOC = P17;
NET "U4_peripherals/EXTERN_LINEAR_FLASH_ADDRESS[6]" LOC = P19;
NET "U4_peripherals/EXTERN_LINEAR_FLASH_ADDRESS[7]" LOC = N23;
NET "U4_peripherals/EXTERN_LINEAR_FLASH_ADDRESS[8]" LOC = N24;
NET "U4_peripherals/EXTERN_LINEAR_FLASH_ADDRESS[9]" LOC = R18;
NET "U4_peripherals/EXTERN_LINEAR_FLASH_CE_N" LOC = AB9;
NET "U4_peripherals/EXTERN_LINEAR_FLASH_DATA[0]" LOC = W25;
NET "U4_peripherals/EXTERN_LINEAR_FLASH_DATA[10]" LOC = W8;
NET "U4_peripherals/EXTERN_LINEAR_FLASH_DATA[11]" LOC = AF6;
NET "U4_peripherals/EXTERN_LINEAR_FLASH_DATA[12]" LOC = AD6;
NET "U4_peripherals/EXTERN_LINEAR_FLASH_DATA[13]" LOC = W19;
NET "U4_peripherals/EXTERN_LINEAR_FLASH_DATA[14]" LOC = V18;
NET "U4_peripherals/EXTERN_LINEAR_FLASH_DATA[15]" LOC = AD23;
NET "U4_peripherals/EXTERN_LINEAR_FLASH_DATA[1]" LOC = AB14;
NET "U4_peripherals/EXTERN_LINEAR_FLASH_DATA[2]" LOC = AF22;
NET "U4_peripherals/EXTERN_LINEAR_FLASH_DATA[3]" LOC = Y20;
NET "U4_peripherals/EXTERN_LINEAR_FLASH_DATA[4]" LOC = AD5;
NET "U4_peripherals/EXTERN_LINEAR_FLASH_DATA[5]" LOC = N18;
NET "U4_peripherals/EXTERN_LINEAR_FLASH_DATA[6]" LOC = AA11;
NET "U4_peripherals/EXTERN_LINEAR_FLASH_DATA[7]" LOC = AF3;
NET "U4_peripherals/EXTERN_LINEAR_FLASH_DATA[8]" LOC = AA10;
NET "U4_peripherals/EXTERN_LINEAR_FLASH_DATA[9]" LOC = W7;
NET "U4_peripherals/EXTERN_LINEAR_FLASH_OE_N" LOC = W26;
NET "U4_peripherals/EXTERN_LINEAR_FLASH_RESET" LOC = AA9;
NET "U4_peripherals/EXTERN_LINEAR_FLASH_WE_N" LOC = AA25;
NET "U4_peripherals/EXTERN_PUSH_BUTTONS_3BITS_TRI_I[0]" LOC = L20;
NET "U4_peripherals/EXTERN_PUSH_BUTTONS_3BITS_TRI_I[1]" LOC = L21;
NET "U4_peripherals/EXTERN_PUSH_BUTTONS_3BITS_TRI_I[2]" LOC = H20;
NET "U4_peripherals/EXTERN_RESET_IN" LOC = M19;
NET "U4_peripherals/EXTERN_RS232_USB_SIN" LOC = AE2;
NET "U4_peripherals/EXTERN_RS232_USB_SOUT" LOC = AE1;
NET "U4_peripherals/MCB_DDR3/mcbx_dram_addr[0]" LOC = L7;
NET "U4_peripherals/MCB_DDR3/mcbx_dram_addr[10]" LOC = J9;
NET "U4_peripherals/MCB_DDR3/mcbx_dram_addr[11]" LOC = E3;
NET "U4_peripherals/MCB_DDR3/mcbx_dram_addr[12]" LOC = K8;
NET "U4_peripherals/MCB_DDR3/mcbx_dram_addr[1]" LOC = L6;
NET "U4_peripherals/MCB_DDR3/mcbx_dram_addr[2]" LOC = K10;
NET "U4_peripherals/MCB_DDR3/mcbx_dram_addr[3]" LOC = M8;
NET "U4_peripherals/MCB_DDR3/mcbx_dram_addr[4]" LOC = J7;
NET "U4_peripherals/MCB_DDR3/mcbx_dram_addr[5]" LOC = L4;
NET "U4_peripherals/MCB_DDR3/mcbx_dram_addr[6]" LOC = L3;
NET "U4_peripherals/MCB_DDR3/mcbx_dram_addr[7]" LOC = L10;
NET "U4_peripherals/MCB_DDR3/mcbx_dram_addr[8]" LOC = C2;
NET "U4_peripherals/MCB_DDR3/mcbx_dram_addr[9]" LOC = C1;
NET "U4_peripherals/MCB_DDR3/mcbx_dram_ba[0]" LOC = B2;
```

```

NET "U4_peripherals/MCB_DDR3/mcbx_dram_ba[1]" LOC = B1;
NET "U4_peripherals/MCB_DDR3/mcbx_dram_ba[2]" LOC = G3;
NET "U4_peripherals/MCB_DDR3/mcbx_dram_cas_n" LOC = L8;
NET "U4_peripherals/MCB_DDR3/mcbx_dram_clk" LOC = K5;
NET "U4_peripherals/MCB_DDR3/mcbx_dram_clk_n" LOC = J5;
NET "U4_peripherals/MCB_DDR3/mcbx_dram_cke" LOC = K9;
NET "U4_peripherals/MCB_DDR3/mcbx_dram_ldm" LOC = J3;
NET "U4_peripherals/MCB_DDR3/mcbx_dram_dq[0]" LOC = H3;
NET "U4_peripherals/MCB_DDR3/mcbx_dram_dq[10]" LOC = K3;
NET "U4_peripherals/MCB_DDR3/mcbx_dram_dq[11]" LOC = K1;
NET "U4_peripherals/MCB_DDR3/mcbx_dram_dq[12]" LOC = M3;
NET "U4_peripherals/MCB_DDR3/mcbx_dram_dq[13]" LOC = M1;
NET "U4_peripherals/MCB_DDR3/mcbx_dram_dq[14]" LOC = N2;
NET "U4_peripherals/MCB_DDR3/mcbx_dram_dq[15]" LOC = N1;
NET "U4_peripherals/MCB_DDR3/mcbx_dram_dq[1]" LOC = H1;
NET "U4_peripherals/MCB_DDR3/mcbx_dram_dq[2]" LOC = G2;
NET "U4_peripherals/MCB_DDR3/mcbx_dram_dq[3]" LOC = G1;
NET "U4_peripherals/MCB_DDR3/mcbx_dram_dq[4]" LOC = D3;
NET "U4_peripherals/MCB_DDR3/mcbx_dram_dq[5]" LOC = D1;
NET "U4_peripherals/MCB_DDR3/mcbx_dram_dq[6]" LOC = E2;
NET "U4_peripherals/MCB_DDR3/mcbx_dram_dq[7]" LOC = E1;
NET "U4_peripherals/MCB_DDR3/mcbx_dram_dq[8]" LOC = J2;
NET "U4_peripherals/MCB_DDR3/mcbx_dram_dq[9]" LOC = J1;
NET "U4_peripherals/MCB_DDR3/mcbx_dram_dqs" LOC = F3;
NET "U4_peripherals/MCB_DDR3/mcbx_dram_dqs_n" LOC = F1;
NET "U4_peripherals/MCB_DDR3/mcbx_dram_odt" LOC = M6;
NET "U4_peripherals/MCB_DDR3/mcbx_dram_ras_n" LOC = L9;
NET "U4_peripherals/MCB_DDR3/mcbx_dram_ddr3_rst" LOC = E4;
NET "U4_peripherals/MCB_DDR3/mcbx_dram_udm" LOC = J4;
NET "U4_peripherals/MCB_DDR3/mcbx_dram_udqs" LOC = L2;
NET "U4_peripherals/MCB_DDR3/mcbx_dram_udqs_n" LOC = L1;
NET "U4_peripherals/MCB_DDR3/mcbx_dram_we_n" LOC = G4;
NET "U4_peripherals/MCB_DDR3/rzq" LOC = M4;
NET "U4_peripherals/MCB_DDR3/zio" LOC = H6;
END ISOLATION_GROUP

#U5_mb0_comp pin definitions
ISOLATION_GROUP U5_mb0_comp BEGIN
END ISOLATION_GROUP

#U6_mb1_comp pin definitions
ISOLATION_GROUP U6_mb1_comp BEGIN
END ISOLATION_GROUP

```

The second file that is needed is the parameter file. The parameter file usually has the extension `.ivt`. In UCF mode, the IVT accepts the parameters listed in [Table 5](#).

**Table 5: IVT Parameters in UCF Mode**

IVT Command Line Argument (1)	Description
<code>-device <i>device_name</i></code>	Specifies the device.
<code>-package <i>package_name</i></code>	Specifies the package.
<code>-group <i>isolation_group area_group</i></code>	Associates the area group name to an arbitrary isolation group name. At least two distinct isolation groups are required.
<code>[-pig <i>pin_isolation_groups</i>.[pig]]</code>	Specifies the pin isolation group file. If omitted, no pin-related analysis is performed.
<code>[-output <i>output</i>.[rpt]]</code>	Specifies the name of the output report.

Table 5: IVT Parameters in UCF Mode (Cont'd)

IVT Command Line Argument (1)	Description
[-verbose]	Writes out all IVT reporting information.
[-f <i>parameter_file</i> ]	Specifies an external file that lists all the IVT command line arguments.
constraint_file[.ucf]	Constraint file for the design (UCF).
[-h]	Displays a brief argument summary.
[-license]	Displays the license agreements.

**Notes:**

- Optional arguments are in brackets [ ].

The `lockstep_system_ucf.ivt` file for the dual-lockstep MicroBlaze processor system is located at `<reference design>\ivt\ucf` and has these contents:

```
#Verbose callout to print the verbose report.
-verbose

# define the device targeted for the design with the -device flag.
-device xc6slx150t -package fgg676

# define the isolation groups and their corresponding area groups as
identified in PlanAhead and the ucf.
# Groups Isolation Group Area Group
# -----
-group U2_mb0 pblock_U2_mb0
-group U3_mb1 pblock_U3_mb1
-group U4_peripherals pblock_U4_peripherals
-group U5_mb0_comp pblock_U5_mb0_comp
-group U6_mb1_comp pblock_U6_mb1_comp

#-pig flag identifies the Pin Isolation Group file, which defines the
association of the Isolation group
#to the design I/O.
# Pin Isolation Groups
-pig lockstep_system.pig

#Identify the location of the UCF file to evaluate.
# User Constraint File
..\..\src\ucf\lockstep_system.ucf

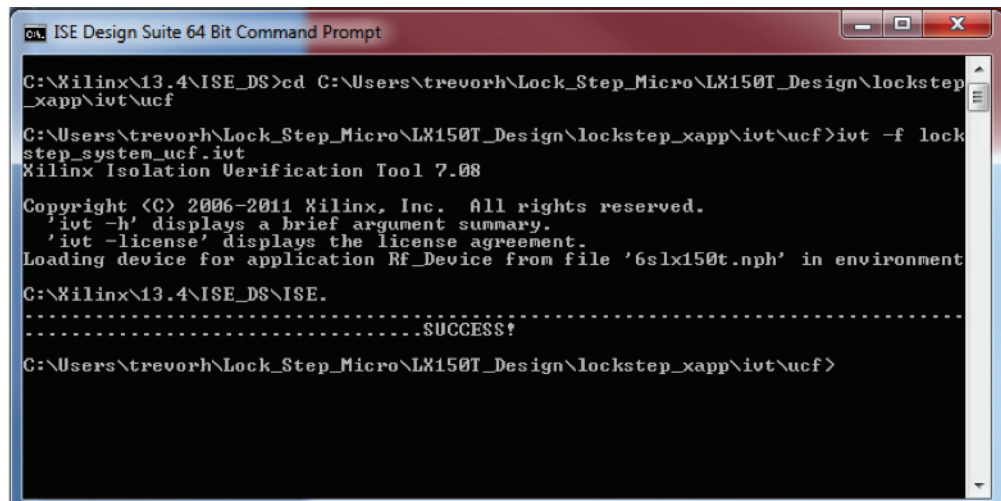
#Identify the output report file to generate.
# Output file
-output lockstep_system_ucf.rpt
```

- Open up an ISE Design Suite command prompt.

**Note:** The next action launches the 64-bit Windows version of the tool.

**Start > All Programs > Xilinx ISE Design Suite 13.4 > Accessories > ISE Design Suite 64 Bit Command Prompt.**

- Within the command prompt, change directory (**cd**) to `<reference design>\ivt\ucf`. See [Figure 72](#) for an example.
- Within the command prompt, type **ivt -f lockstep\_system\_ucf.ivt**. Press **Enter**.
- After the IVT runs, verify that the status displays **SUCCESS!**, indicating that no isolation violations were found.



```

C:\Xilinx\13.4\ISE_DS>cd C:\Users\trevorh\Lock_Step_Micro\LX150T_Design\lockstep_xapp\ivt\ucf

C:\Users\trevorh\Lock_Step_Micro\LX150T_Design\lockstep_xapp\ivt\ucf>ivt -f lockstep_system_ucf.ivt
Xilinx Isolation Verification Tool 7.08

Copyright (C) 2006-2011 Xilinx, Inc. All rights reserved.
'ivt -h' displays a brief argument summary.
'ivt -license' displays the license agreement.
Loading device for application RF_Device from file '6slx150t.nph' in environment

C:\Xilinx\13.4\ISE_DS\ISE.
.....SUCCESS!
C:\Users\trevorh\Lock_Step_Micro\LX150T_Design\lockstep_xapp\ivt\ucf>

```

X584\_72\_040512

Figure 72: IVT in UCF Mode Command Line

### Examining the Outputs of the IVT in UCF Mode

The IVT outputs two files when complete, a text report file and a graphical report file. The text report file for the run is located at <reference design>

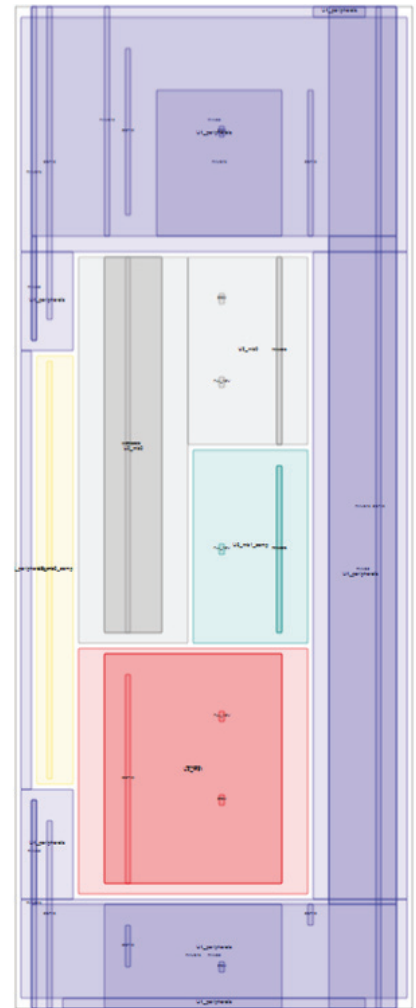
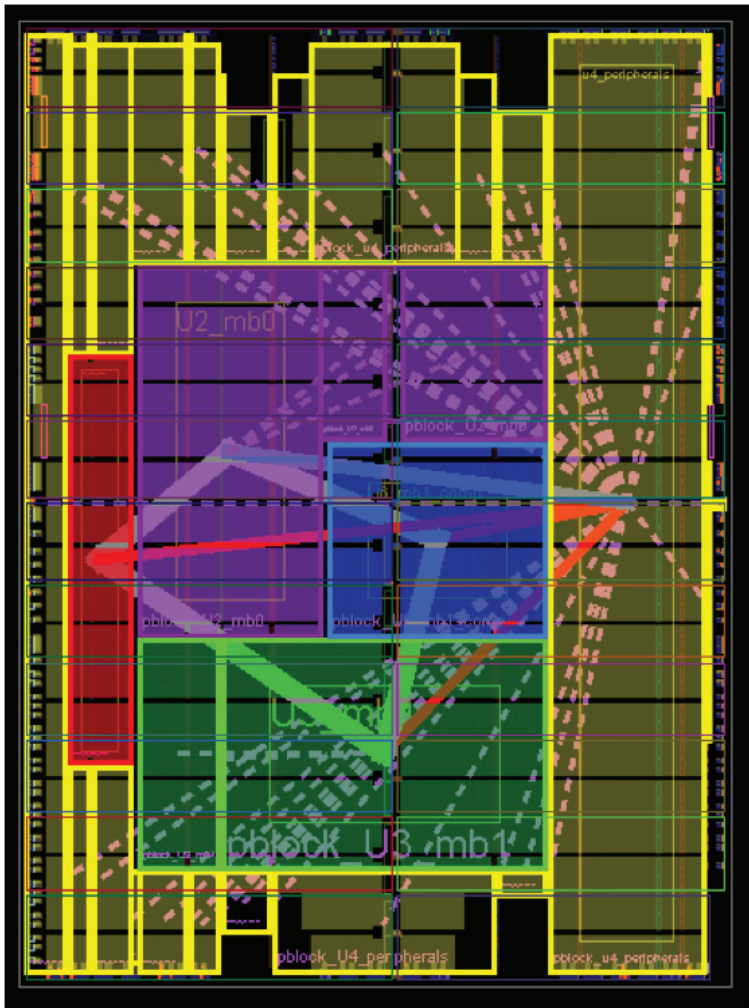
\ivt\ucf\lockstep\_system\_ucf.rpt. The text report file contains these sections:

- **Provenance:** This section includes the date, IVT version, ISE tools version used for the run, ISE tools version against which the IVT was compiled, the location of the ISE tools installation, the command line, the current working directory, the output report location, and the part and package.
- **Area range constraints:** This section contains the corresponding area groups and associated site ranges used to define the floorplan of the design are listed for each isolation group.
- **Package pins, I/O buffers, and I/O banks:** In this section, pin assignments of the design are presented with coordinates, I/O banks, isolation groups, and net names.
- **Pin isolation summary:** This section lists the number of isolation violations for die pins (I/O buffers), package pins, and I/O banks.
- **Area group separation:** This section lists the distance in tiles between area ranges from distinct isolation groups.
- **Area fault summary:** This section lists the number of area ranges from distinct isolation groups that are not separated by an adequate fence.
- **Isolation verification summary:** This section lists the total number of constraints violated, reports completion, and reports the elapsed time to perform the analysis and generate the report.

The graphical report file for the run is located at <reference design>

\ivt\ucf\lockstep\_system\_ucf.svg. This file is generated as a silicon vector graphics (SVG) file so that it can be displayed with a web browser. While the text report is all-encompassing, the SVG file gives a high-level view of any faults (these are indicated with X's) and is useful in debugging the floorplan. In the SVG file, the colored tiles denote the ownership of the tiles by each isolated region. The uncolored tiles denote the fence. [Figure 73](#) shows the SVG file next to the floorplan for the dual-lockstep MicroBlaze processor system.





X584\_73\_040512

Figure 73: Floorplanned Design and IVT SVG Output in UCF Mode

## Implement the Design

Figure 74 shows the reference design progress to this point.

<input checked="" type="checkbox"/>	1. Generate Dual MicroBlaze System in EDK Platform Studio.
<input checked="" type="checkbox"/>	2. Modify Dual MicroBlaze System into Lockstep through EDK Platform Studio.
<input checked="" type="checkbox"/>	3. Build the Dual-Lockstep MicroBlaze System in EDK Platform Studio.
<input checked="" type="checkbox"/>	4. Perform a Quick Sanity Check of the Design.
<input checked="" type="checkbox"/>	5. Prepare Dual-Lockstep MicroBlaze Design Source for Isolated Design.
<input checked="" type="checkbox"/>	6. Synthesize and Floorplan Hierarchical Design.
<input checked="" type="checkbox"/>	7. Run IVT on the Design in UCF Mode.
	8. Implement the Design.
	9. Run IVT on the Design in NCD Mode.
	10. Build Final Software in SDK.
	11. Disconnect the MicroBlaze Processors and Debug Logic.
	12. Re-verify the Re-implemented Design in IVT.

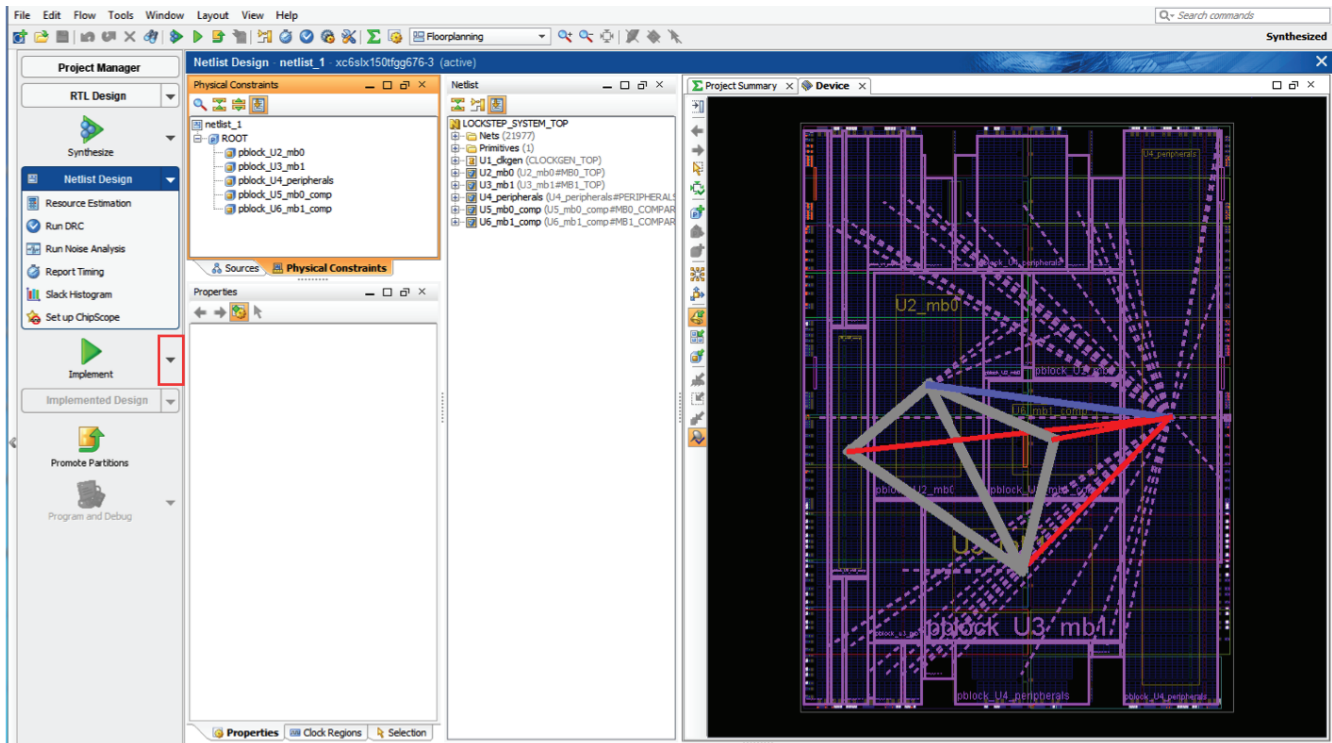
X584\_74\_041112

Figure 74: Reference Design Progress

With the reference design floorplan done and the floorplan verified through the IVT, the next step is to implement the design, and run `ngdbuild`, `par`, `map`, and `BitGen`. All of the implementation tools are configured and run through the PlanAhead tool for this reference design.

### Setting the Implementation Settings and Running Implementation

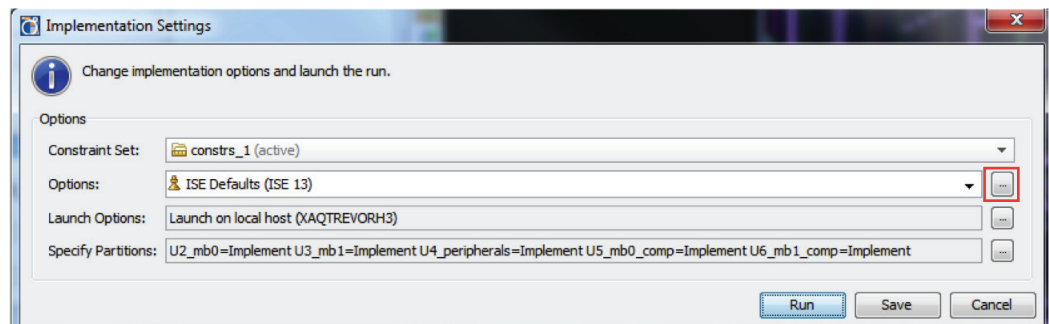
1. On the left side of the PlanAhead tool window, click the down arrow next to the **Implement** button, and select **Implementation Settings...** (see Figure 75).



X584\_75\_040512

Figure 75: Accessing the Implementation Settings

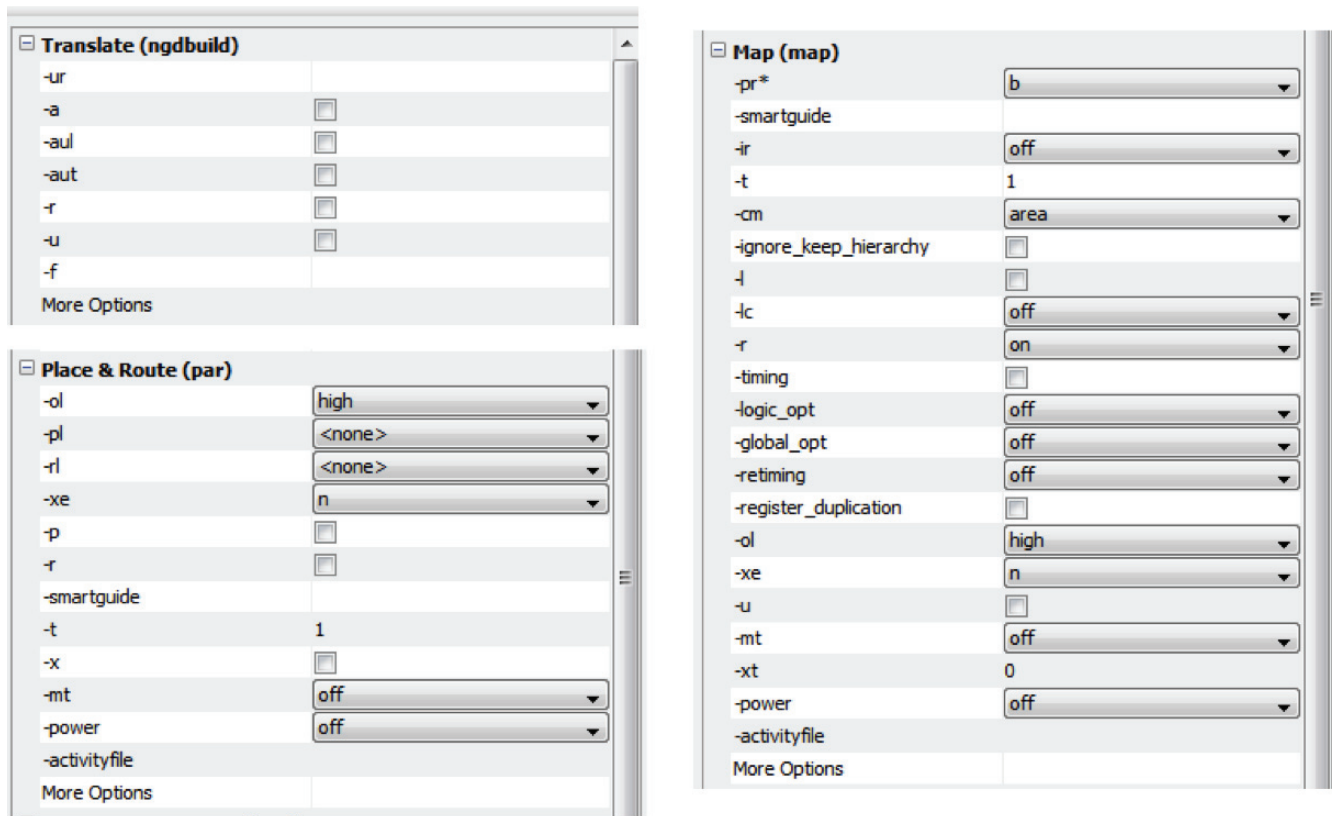
- In the Implementation Settings window, expand the implementation settings by clicking the ... button next to the **Options:** row (see Figure 76).



X584\_76\_040512

Figure 76: Expanding the Implementation Options

- In the Design Run Settings window, apply the settings for **Translate**, **Map**, and **Place & Route** as indicated in Figure 77. Click **OK** to apply the settings.

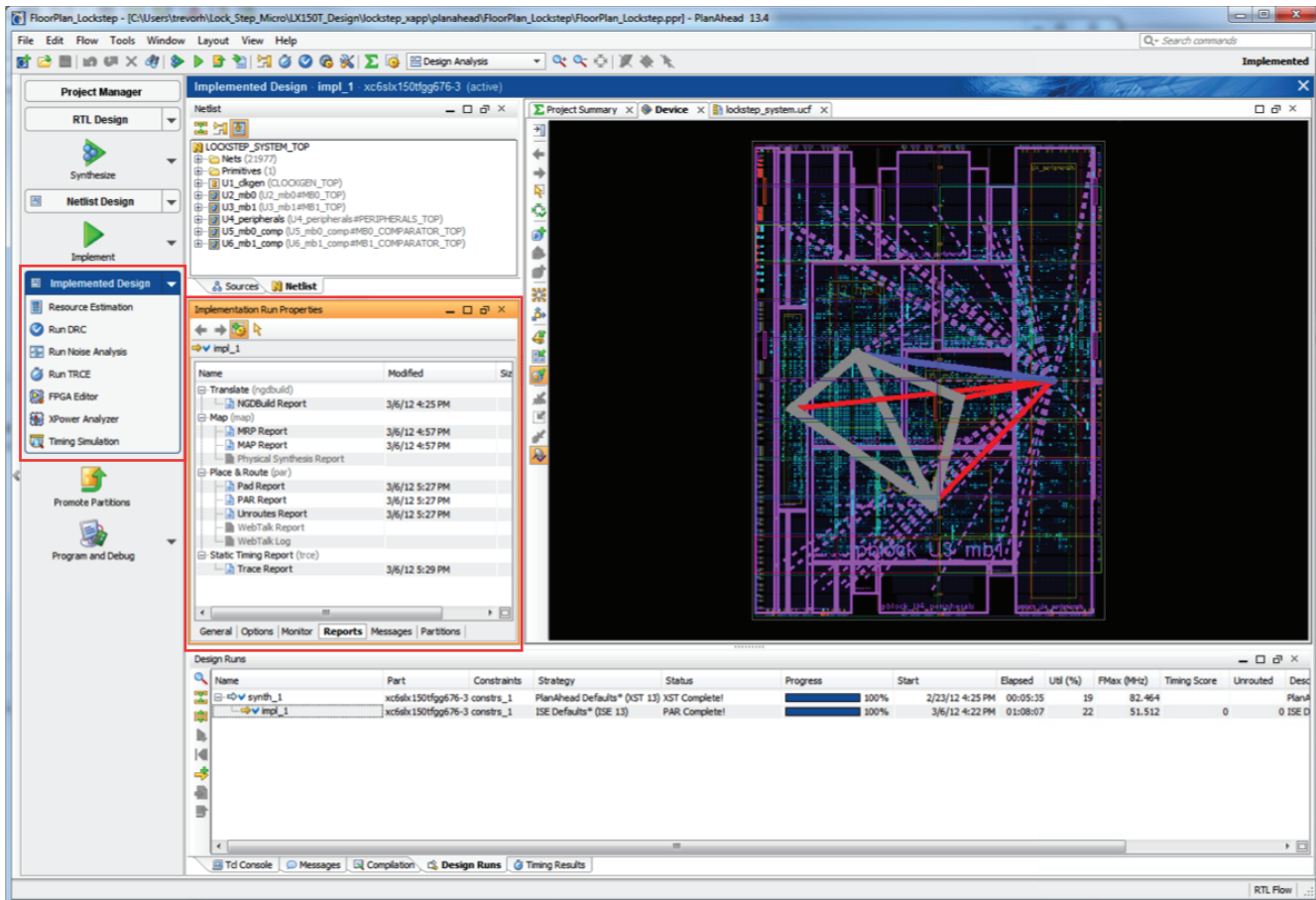


X584\_77\_040512

Figure 77: Implementation Settings

4. In the Implementation Settings window, click **Run** to start running the implementation.
 

**Note:** The implementation takes some time.
5. When implementation has completed, open the implemented design by either selecting **Open Implemented Design** at the Implementation Completed prompt or click **Implemented Design** on the left side of the PlanAhead window (Figure 78). When the implemented design is open, the placed components are shown in the PlanAhead tool Device pane. With the Implemented Design open, the user can run Trace for Timing, bring-up FPGA Editor, or execute other analysis tools all from the left side of the PlanAhead tool window. The implementation reports and logs can also be viewed from the Reports tab within the Implementation Run Properties pane. The overall design run should show a Timing Score of 0 and an Unrouted Score of 0 in the Design Runs tab, showing that the design routed completely and timing was met.



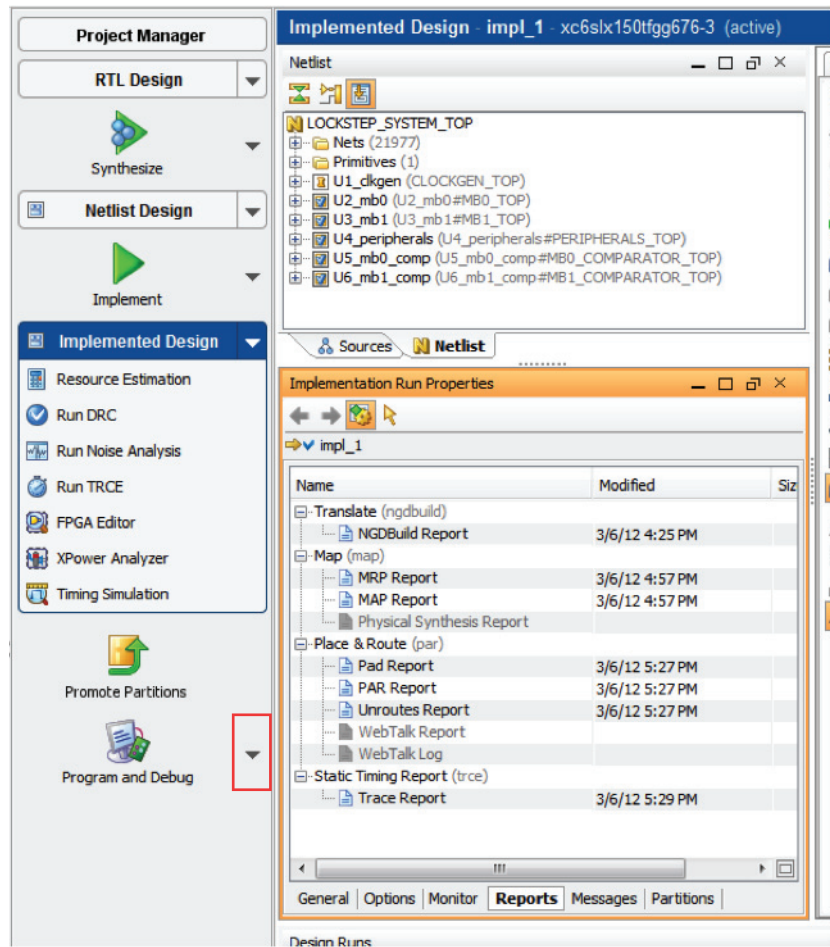
X584\_78\_040512

Figure 78: Implemented Design View in the PlanAhead Tool

## Running BitGen

While in the PlanAhead tool, generate the programming bitstream by executing BitGen.

1. On the left side of the PlanAhead tool window (Figure 79), click the down arrow next to Program and Debug, and select **Generate Bitstream....**



X584\_79\_040512

Figure 79: Select Program and Debug - BitGen

- In the Generate Bitstream window (Figure 80), click **OK** to start BitGen and generate the bitstream.



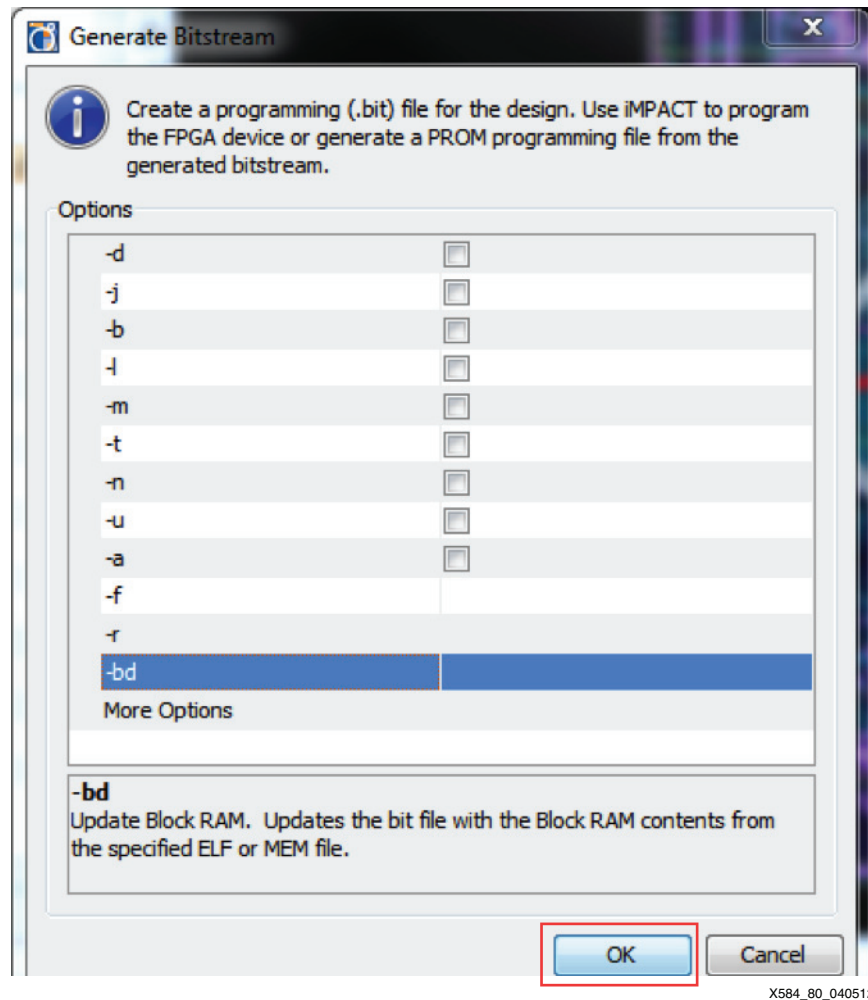


Figure 80: Generate Bitstream Window

3. After the bitstream is generated successfully, the Bitstream Generation Completed window appears. Click **OK** to close the window.

## Run the IVT on the Design in NCD Mode

Figure 81 shows the reference design progress to this point.

<input checked="" type="checkbox"/>	1. Generate Dual MicroBlaze System in EDK Platform Studio.
<input checked="" type="checkbox"/>	2. Modify Dual MicroBlaze System into Lockstep through EDK Platform Studio.
<input checked="" type="checkbox"/>	3. Build the Dual-Lockstep MicroBlaze System in EDK Platform Studio.
<input checked="" type="checkbox"/>	4. Perform a Quick Sanity Check of the Design.
<input checked="" type="checkbox"/>	5. Prepare Dual-Lockstep MicroBlaze Design Source for Isolated Design.
<input checked="" type="checkbox"/>	6. Synthesize and Floorplan Hierarchical Design.
<input checked="" type="checkbox"/>	7. Run IVT on the Design in UCF Mode.
<input checked="" type="checkbox"/>	8. Implement the Design.
	9. Run IVT on the Design in NCD Mode.
	10. Build Final Software in SDK.
	11. Disconnect the MicroBlaze Processors and Debug Logic.
	12. Re-verify the Re-implemented Design in IVT.

X584\_81\_041112

Figure 81: Reference Design Progress

The IVT, in NCD mode, does all the same checks it did in UCF mode, but now IVT works on the final routed design. Instead of looking at area group isolation, the IVT now ensures all the components and nets of each isolated module have a valid fence between them.

The files to run the IVT on the design are provided as part of this application note.

### Executing the IVT in NCD Mode

When running the IVT in NCD mode, only the parameter file is required. The parameter file usually has the extension `.ivt`. In NCD mode, the IVT accepts the parameters listed in Table 6.

Table 6: IVT Parameters in NCD Mode

IVT Command Line Argument <sup>(1)</sup>	Description
<code>-group isolation_group instance_name</code>	Associates the isolated instance to the area group name.
<code>[-output output[.rpt]]</code>	Specifies the name of the output report.
<code>[-verbose]</code>	Writes out all IVT reporting information.
<code>[-f parameter_file]</code>	Specifies an external file that lists all the IVT command line arguments.
<code>user_design[.ncd]</code>	Specifies the fully routed design file (NCD).
<code>[-h]</code>	Displays a brief argument summary.
<code>[-license]</code>	Displays the license agreements.

#### Notes:

- Optional arguments are in brackets [ ].

The `lockstep_system_ncd.ivt` file for the dual-lockstep MicroBlaze processor system is located at `<reference design>\ivt\ncd` and has these contents:

```
-verbose
# Groups Isolation Group Instance Name
# -----
-group U2_mb0 U2_mb0
```

```
-group U3_mb1 U3_mb1
-group U4_peripherals U4_peripherals
-group U5_mb0_comp U5_mb0_comp
-group U6_mb1_comp U6_mb1_comp
```

```
# Combined Design
LOCKSTEP_SYSTEM_TOP_routed.ncd
```

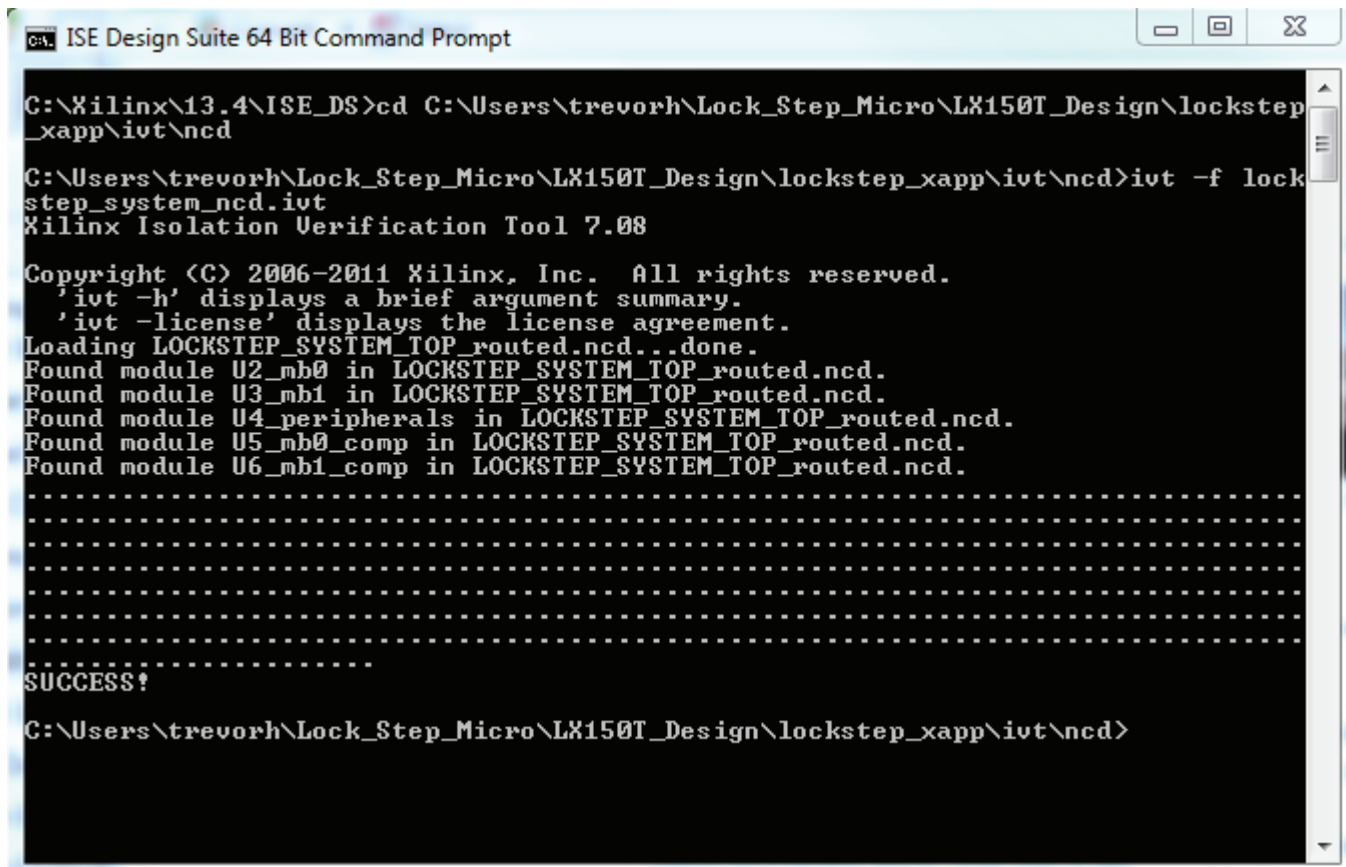
```
# Output file
-output lockstep_system_ncd.rpt
```

1. To access the routed design, in the Design Runs pane within the PlanAhead tool, left-click and then right-click **impl\_1**.
2. In the pop-up window, select **Open Run Directory....**
3. In the explorer window that appears, copy the `LOCKSTEP_SYSTEM_TOP_routed.ncd` file to the `<reference design>\ivt\ncd` directory.
4. Open up a ISE Design Suite command prompt.

**Start > All Programs > Xilinx ISE Design Suite 13.4 > Accessories > ISE Design Suite 64 Bit Command Prompt**

**Note:** This step launches the 64-bit Windows version of the tool.

5. Within the command prompt, change directory (**cd**) to `<reference design>\ivt\ncd`.
6. Within the command prompt, type `ivt -f lockstep_system_ncd.ivt` and press **Enter**.
7. After the IVT runs, verify the status displays **SUCCESS!** to show that no isolation violations were found (Figure 82).



```

C:\Xilinx\13.4\ISE_DS>cd C:\Users\trevorh\Lock_Step_Micro\LX150T_Design\lockstep_xapp\ivt\ncd
C:\Users\trevorh\Lock_Step_Micro\LX150T_Design\lockstep_xapp\ivt\ncd>ivt -f lockstep_system_ncd.ivt
Xilinx Isolation Verification Tool 7.008

Copyright (C) 2006-2011 Xilinx, Inc. All rights reserved.
'ivt -h' displays a brief argument summary.
'ivt -license' displays the license agreement.
Loading LOCKSTEP_SYSTEM_TOP_routed.ncd...done.
Found module U2_mb0 in LOCKSTEP_SYSTEM_TOP_routed.ncd.
Found module U3_mb1 in LOCKSTEP_SYSTEM_TOP_routed.ncd.
Found module U4_peripherals in LOCKSTEP_SYSTEM_TOP_routed.ncd.
Found module U5_mb0_comp in LOCKSTEP_SYSTEM_TOP_routed.ncd.
Found module U6_mb1_comp in LOCKSTEP_SYSTEM_TOP_routed.ncd.
.....
SUCCESS!
C:\Users\trevorh\Lock_Step_Micro\LX150T_Design\lockstep_xapp\ivt\ncd>

```

X584\_82\_041112

Figure 82: IVT in NCD Mode Command Line on Implemented Design

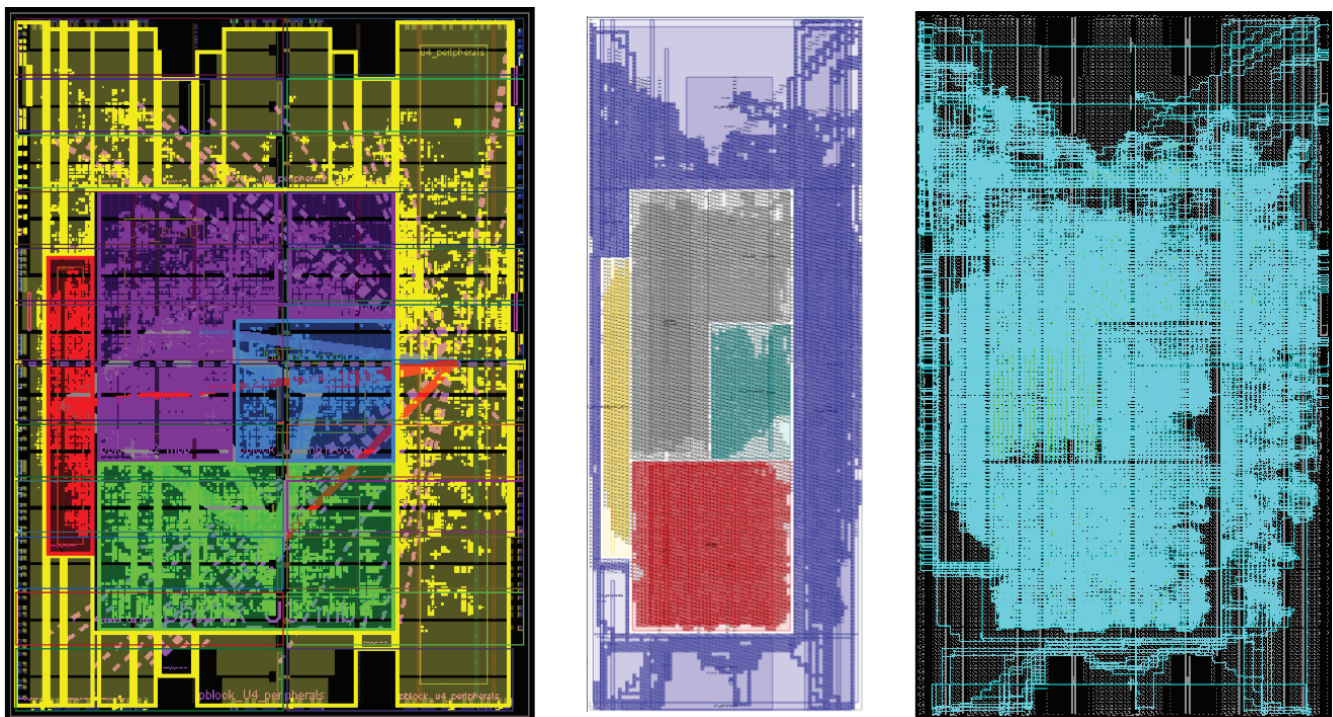
## Examining the Outputs of the IVT in NCD Mode

The IVT, in NCD mode, also outputs two files when complete, a text report file and a graphical report file. The text report file for the run is located at `<reference design>\ivt\ncd\lockstep_system_ncd.rpt` and contains these sections:

- **Provenance:** This section includes the date, IVT version, ISE tools version used for the run, ISE tools version against which the IVT was compiled, the location of the ISE tools installation, the command line, the current working directory, the output report location, and the part and package.
- **Isolated Modules:** This section lists the isolation groups and associated design blocks or partial NCD files.
- **Uncategorized User Global Nets:** This section lists nets (signals) in the design that are above the isolated modules in the design hierarchy and might connect isolated functions. All such nets must be examined for their impact on the data separation and independence requirements of the system. Ideally, the only nets listed in this section would be nets specifically intended to connect isolated functions. In practice, some global clock signals appear here as well. In the report, uncategorized global nets are said to be “found in multiple isolation groups.” This is an artifact of the original implementation of the IVT in which multiple partial NCD files were used to specify isolation groups. Global resources are duplicated in all the partial NCD files.
- **Categorized Nets:** This section has many possible subsections corresponding to various categories of nets that for one reason or another are benign with respect to isolation. Examples include constants, global clocks, trusted bus macros, and clocking inserted automatically to mitigate negative-bias temperature instabilities.
- **Trusted Bus Macros:** This section lists all the instances of trusted bus macros in the design and the nets connected to them.
- **Area Range Constraints:** This section lists the corresponding area groups and associated site ranges used to define the floorplan of the design for each isolation group.
- **Net Fault-Cost Violations (Failing Paths):** For Virtex-4 FPGA designs only, this section lists pairs of putatively isolated nets that cannot be shown to be sufficiently isolated with a routing search based on fault cost. Examples of low-cost paths between isolated nets are listed, and several customer designs have produced isolation violations. However, in all cases, the violations turned out to be due to insufficient information in the cost function, not an actual vulnerability.
- **Tiles with Net Content Violations:** This section lists the contents of all tiles that contain isolated logic or routing from more than one isolation group.
- **Tiles in the Fence Containing Nets:** This section lists the contents of tiles that are outside all the isolated area ranges. This list is advisory. It is permissible for inter-region signals and clocking to exist in the fence. Akin to the list of Uncategorized Global Nets, nets in tiles that are outside of all isolation groups must be vetted for their impact on data separation and independence requirements. This section provides low-level details of the specific nodes and wire segments used to implement routing in the fence.
- **Tiles in the Fence Containing Programming:** This section lists tiles in the fence that are associated with components and therefore are not entirely unused. All such tiles must be examined for their impact on the data separation and independence requirements of the system.
- **Tiles in the Fence Containing Used PIPs:** This section lists tiles containing programmable interconnect points (PIPs), which are nodes that have the potential to connect to other nodes. For example, suppose a certain type of node spans five horizontal tiles called A, B, C, D, and E, and that this node has PIPs in tiles A, C, and E. This node can be used to connect two isolated regions provided the only tiles in the fence are B or D.

- Tiles with Net Adjacency Violations: This section lists all pairs of tiles that are adjacent and contain isolated logic or routing resources from distinct isolation groups—in other words, pairs of tiles that should be separated by a fence tile.
- Package Pins, I/O Buffers, and I/O Banks: In this section, pin assignments of the design are presented with coordinates, I/O banks, isolation groups, and the net names.
- I/O Buffer Isolation Violations: This section lists pairs of I/O buffers from distinct isolation groups that are adjacent on the die.
- I/O Bank Violations: This section lists examples of pins from distinct isolation groups that are members of a single I/O bank.
- Package Pin Isolation Violations: This section lists pairs of pins from distinct isolation groups that are adjacent on the package.
- Isolation Verification Summary: This section lists the total number of constraints violated by category, reports completion, and reports the elapsed time to perform the analysis and generate the report. For each section of the report containing violations, there is a line in the summary with a tally of the violations in that section.

The graphical report file for the run is located at `<reference design>\ivt\ncd\lockstep_system_ncd.svg`. The format and properties are the same for NCD mode as UCF Mode. [Figure 83](#) shows the floorplan, SVG file output, and routed design for the implemented design.

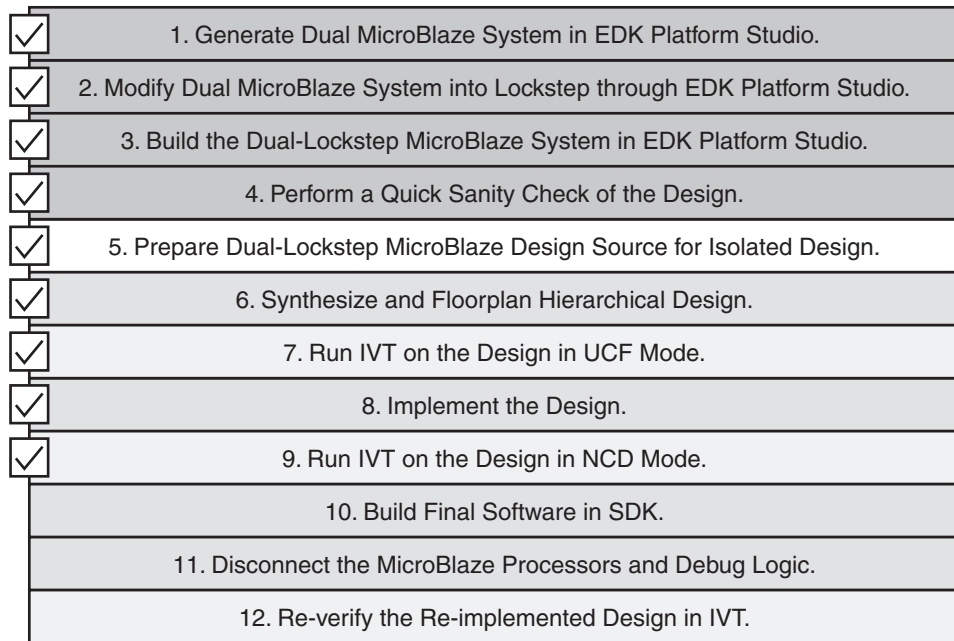


X584\_83\_040512

*Figure 83:* Floorplan, SVG, and FPGA Editor of Implemented Design

## Build Final Software In SDK

[Figure 84](#) shows the reference design progress to this point.



X584\_84\_04112

*Figure 84: Reference Design Progress*

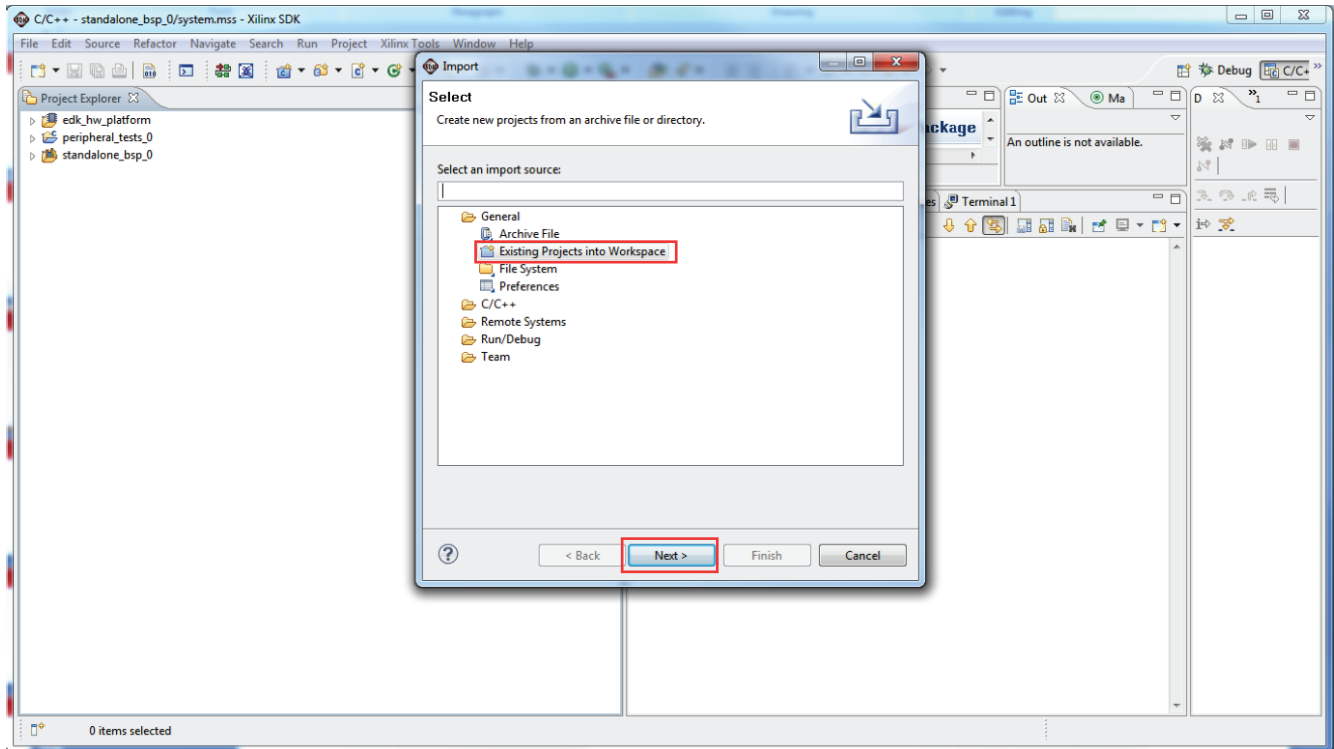
In [Importing the Final Demonstration Software into SDK](#), the final dual-lockstep MicroBlaze processor demonstration software is added to the SDK project. The software is compiled and programmed into the linear flash on the Avnet Spartan-6 FPGA LX150T development board. A bootloader is also created to load the software from the flash into both the MicroBlaze processors of the dual-lockstep MicroBlaze processor system. Finally, the software is executed to test the dual-lockstep MicroBlaze processor system.

### Importing the Final Demonstration Software into SDK

A full SDK environment has been archived in `<reference design>\final_demonstration_sw` that includes the dual-lockstep MicroBlaze processor demonstration software. These steps describe importing this demonstration software into the user's SDK project.

1. Start the EDK Software Development Kit:  
**Start > All Programs > Xilinx ISE Design Suite 13.4 > EDK > Xilinx Software Development Kit**
2. In the Workspace Launcher window, set the workspace location to `<reference design>\sdk`. Click **OK** to open the SDK workspace. The location of the workspace that was generated during the quick sanity check is `<reference design>\sdk`.
3. In the SDK window, click **File > Import...** to bring up the Import window to start the process of importing the demonstration software.
4. In the Import window ([Figure 85](#)), expand the General tree and select **Existing Projects into Workspace**. Click **Next >**.

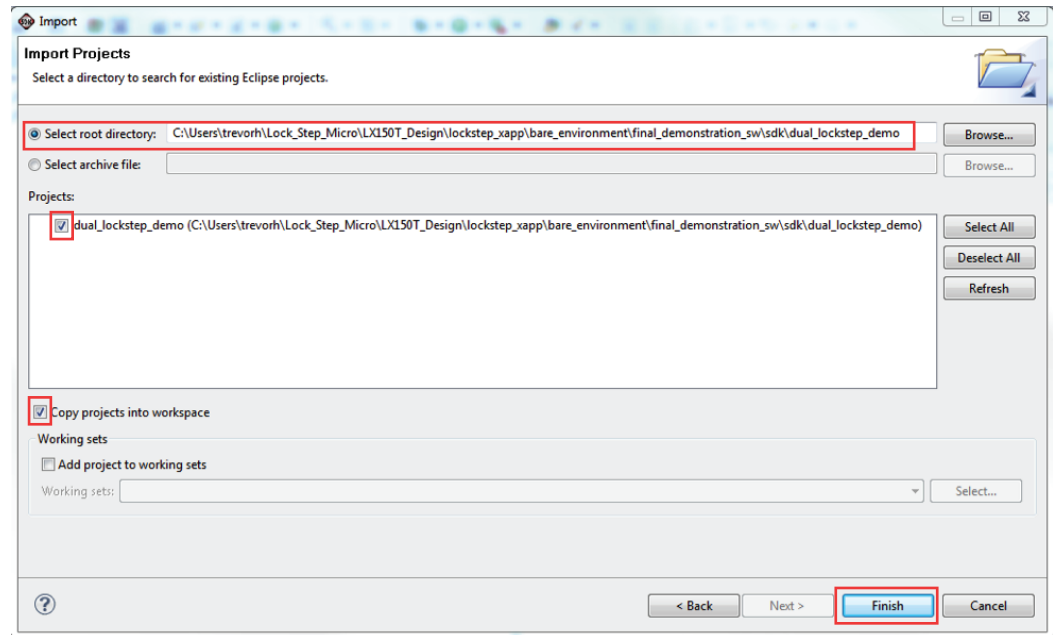




X584\_85\_040512

Figure 85: SDK Import Window

5. In the Import Projects window (Figure 86):
  - a. Choose **Select root directory** and browse to <reference design>\final\_demonstration\_sw\sdk\dual\_lockstep\_demo.
  - b. In the Projects area, check **dual\_lockstep\_demo**.
  - c. Check the **Copy projects into workspace** checkbox.
  - d. Click **Finish** to import the project and related code.



X584\_86\_040512

Figure 86: Import Projects Window

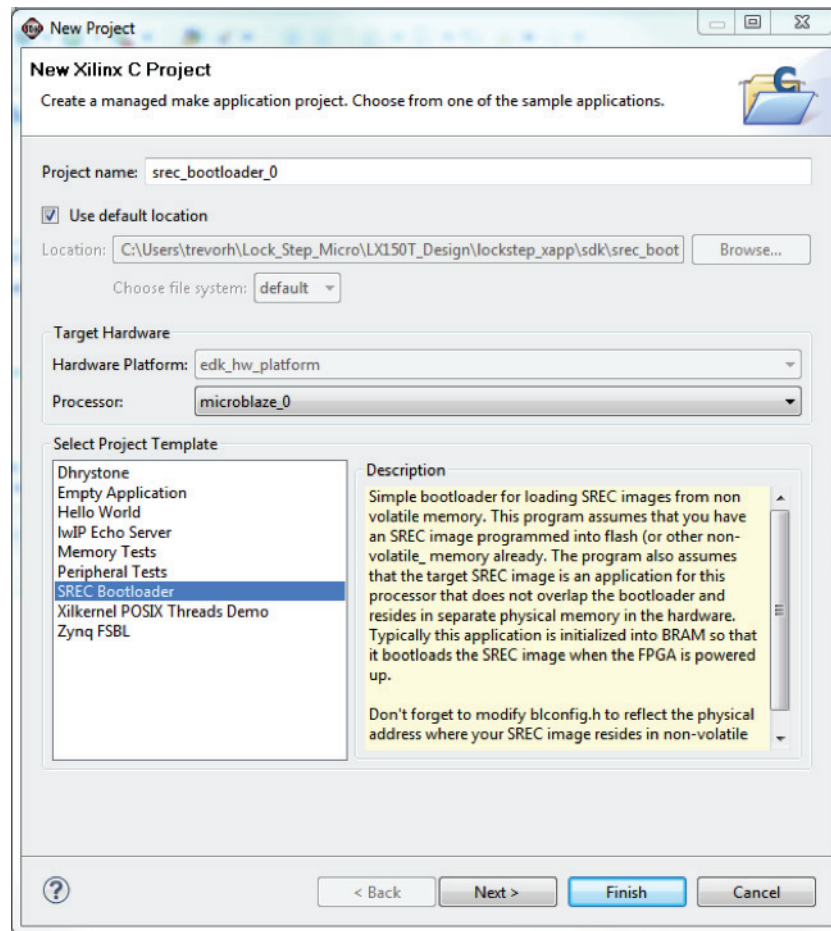
6. With the `dual_lockstep_demo` imported, in the SDK window's Project Explorer pane, right-click `dual_lockstep_demo` and select **Change Referenced BSP**.
7. Verify that the selected BSP to reference is the BSP that was created during the quick check, `standalone_bsp_0`. Click **OK**.

The dual-lockstep demonstration software has now been imported into SDK.

### Creating the Bootloader

As part of this application note, the demonstration software is loaded into the linear flash included in the dual-lockstep MicroBlaze processor system. A bootloader is created to load the demonstration software into the Data and Instruction block RAM for each MicroBlaze processor. These steps define how to create the bootloader:

1. In the SDK window, select **File > New > Xilinx C Project** to bring up the available pre-packaged applications.
2. In the New Xilinx C Project window (Figure 87), set these items to build the bootloader, then click **Next >**:
  - Use Default Location: **Checked**
  - Target Platform: **edk\_hw\_platform**
  - Processor: **microblaze\_0**
  - Select Project Template: **SREC Bootloader**



X584\_87\_040512

Figure 87: New Project Window - SREC Bootloader

3. In the next window, select **Target an existing Board Support Package** and click **Finish** to link this to the existing BSP. The application then compiles.
4. Verify that in the SDK console pane it says `elf check passed. Finished building: srec_bootloader_0.elfcheck` at the end of the compilation.
5. In the SDK window's Project Explorer pane, expand the code tree **srec\_bootloader\_0 > src > blconfig.h**.
6. Double-click `blconfig.h` in the Project Explorer pane to open the file in the SDK text editor.
7. Within the `blconfig.h` file, change the value of `FLASH_IMAGE_BASEADDR` from `0xF8000000` to `0x46000000`.  
`0x46000000` is the first address of the linear flash for the dual-lockstep MicroBlaze processor system. For this application note, the demonstration software is loaded into the flash starting at the first address of the linear flash. This address can be found either from the `lockstep_system.mhs` file that defines the EDK system, or within the `system.xml` file that SDK uses as the hardware platform definition and can be found within the Project Explorer pane at **edk\_hw\_platform > system.xml**.
8. Save the updated `blconfig.h` file.
9. In the SDK window's Project Explorer pane, double-click **srec\_bootloader\_0 > src > platform.c** to open the file in the SDK text editor.

10. Within the `platform.c` file, change the code at line 13 from:

```
XUartNs550_SetBaud(STDOUT_BASEADDR, XPAR_XUARTNS550_CLOCK_HZ, 9600);
```

to:

```
XUartNs550_SetBaud(STDOUT_BASEADDR, XPAR_XUARTNS550_CLOCK_HZ, 38400);
```

This change sets the RS-232 port of the dual-lockstep MicroBlaze processor system to operate at a baud rate of 38400. This is done to make it similar to the baud rate used in the demonstration software.

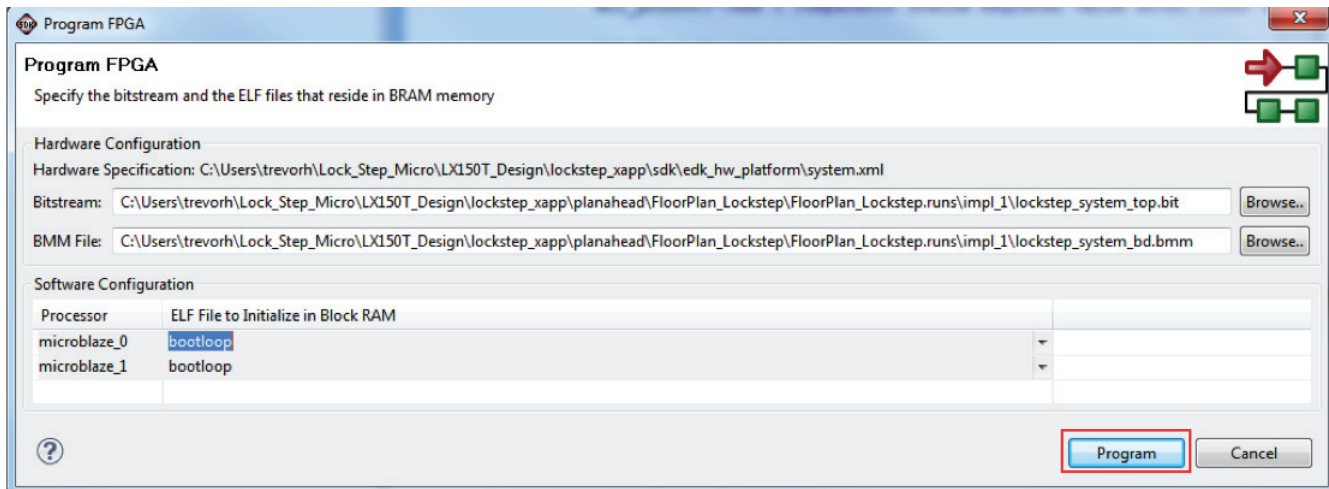
11. Save the updated `platform.c` file.
12. Before the software is loaded into the flash, perform a clean build of the demonstration software and bootloader by right-clicking each in the Project Explorer pane, **dual\_lockstep\_demo** and **src\_bootloader\_0**, and selecting **Build Configurations > Build All**.

### Loading the Demonstration Software into Flash

SDK provides a built-in tool to load software images into the flash on a board. The steps for loading the demonstration software into the linear flash are:

1. Connect a Xilinx Platform Cable to the computer and the Avnet Spartan-6 FPGALX150T development board. Then, do the same with a USB cable.
2. Power on the board.
3. To load the flash, the FPGA must be loaded with an image. The bitstream that was created in the PlanAhead tool is used as the image. Within the SDK window, click **Xilinx Tools > Program FPGA**.
4. In the Program FPGA window ([Figure 88](#)), set these settings, then click **Program**:
  - Hardware Configuration Bitstream:  
`<reference_design>\  
\planahead\FloorPlan_Lockstep\FloorPlan_Lockstep.runs\impl_1\lockstep_system_top.bit`
  - Hardware Configuration BMM File:  
`<reference_design>\planahead\FloorPlan_Lockstep\FloorPlan_Lockstep.runs\impl_1\lockstep_system_bd.bmm`
  - Software Configuration microblaze\_0: **bootloop**
  - Software Configuration microblaze\_1: **bootloop**

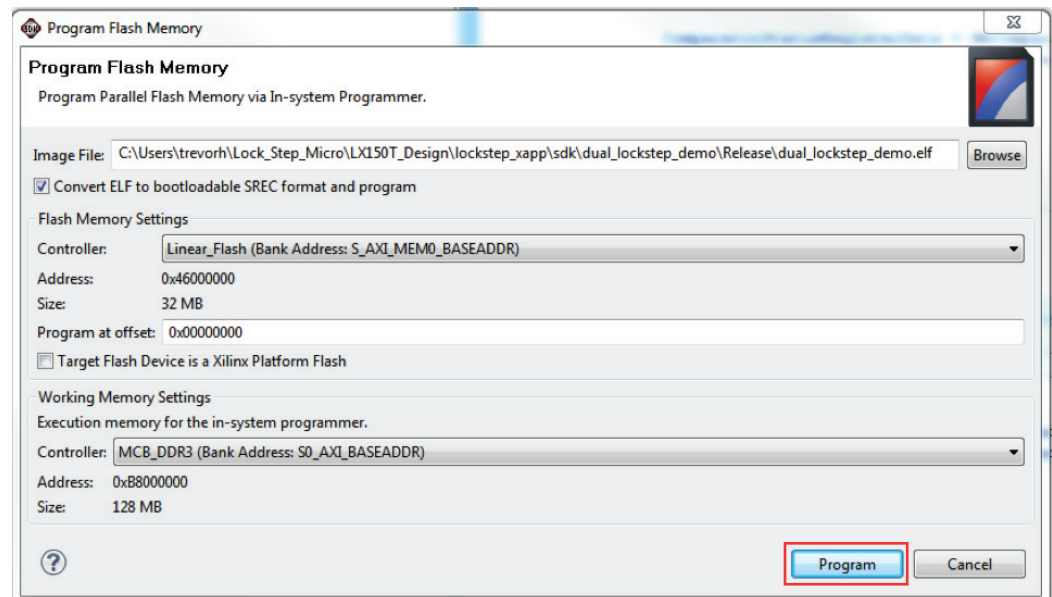
**Note:** The BIT and BMM file for the user implementation might be located in a directory other than `impl_1` if multiple implementations are executed within the PlanAhead tool. The bootloop application loaded into each MicroBlaze processor puts them in a standby state. An actual application is loaded later.



X584\_88\_040512

Figure 88: Program FPGA Window - Bootloop

5. After the FPGA has configured successfully, within the SDK window, click **Xilinx Tools > Program Flash** to start the flash programmer.
6. Click **OK** to the Flash Programmer Support Information window.
7. Within the Program Flash Memory window (Figure 89), make these settings and click **Program**:
  - Image File: <reference design>\sdk\dual\_lockstep\_demo\Release\dual\_lockstep\_demo.elf
  - Check **Convert ELF to bootloadable SREC format and program**.
  - Flash Memory Settings:
    - Controller: **Linear\_Flash**
    - Controller Program at offset: 0x00000000
  - Uncheck **Target Flash Device is a Xilinx Platform Flash**.
  - Working Memory Settings Controller: **MCB\_DDR3 (Bank Address S0\_AXI\_BASEADDR)**



X584\_89\_040512

Figure 89: Program Flash Memory Window

**Note:** The **Convert ELF to bootable SREC format and program** must be checked for the image to be used with the bootloader.

8. After the flash has programmed successfully, click **OK** in the Flash Programmed Successfully prompt.

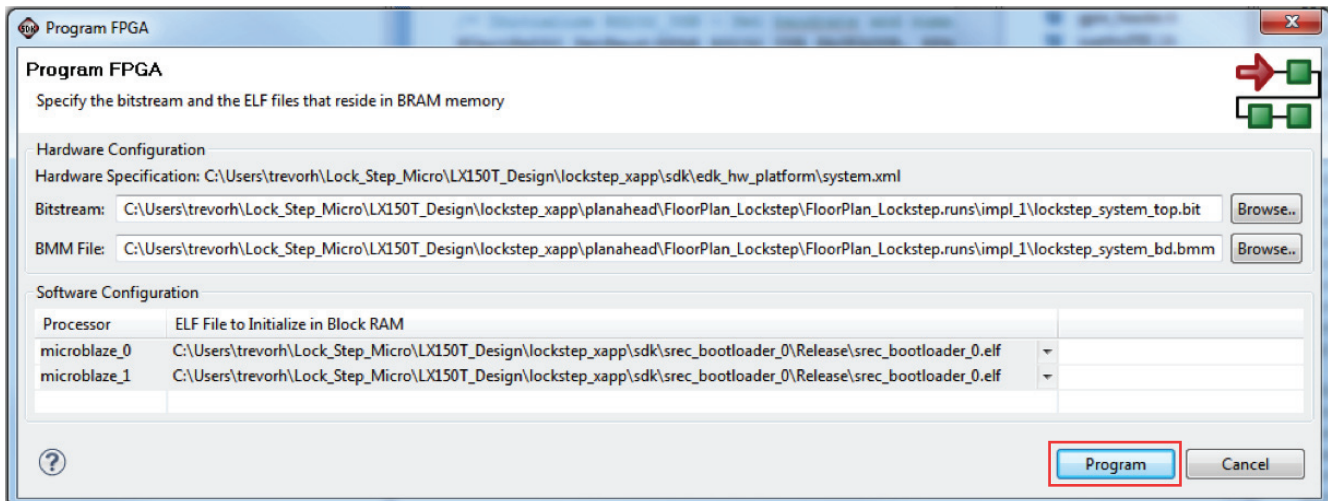
### Loading the Bootloader into the FPGA

With the demonstration software now loaded in the linear flash, the FPGA can be loaded with the bootloader embedded in each MicroBlaze processor.

1. Within the SDK window's Terminal pane, set up the terminal to have a serial connection with settings of **38400**, **8**, **1**, **none**, and **none**. Then connect the terminal.
2. Within the SDK window, click **Xilinx Tools > Program FPGA**.
3. In the Program FPGA window (Figure 90), set or verify these settings, then click **Program**:
  - Hardware Configuration Bitstream: <reference\_design>\planahead\FloorPlan\_Lockstep\FloorPlan\_Lockstep.runs\impl\_1\lockstep\_system\_top.bit
  - Hardware Configuration BMM File: <reference\_design>\planahead\FloorPlan\_Lockstep\FloorPlan\_Lockstep.runs\impl\_1\lockstep\_system\_bd.bmm
  - Software Configuration microblaze\_0: <reference\_design>\sdk\srec\_bootloader\_0\Release\srec\_bootloader\_0.elf
  - Software Configuration microblaze\_1: <reference\_design>\sdk\srec\_bootloader\_0\Release\srec\_bootloader\_0.elf

**Note:** The setting for Software Configuration microblaze\_1 is not found as a direct option. The user has to select **Browse...** and then browse to and select the `elf` file. Selecting the same file as microblaze\_0 ensures the MicroBlaze processors work in lockstep.

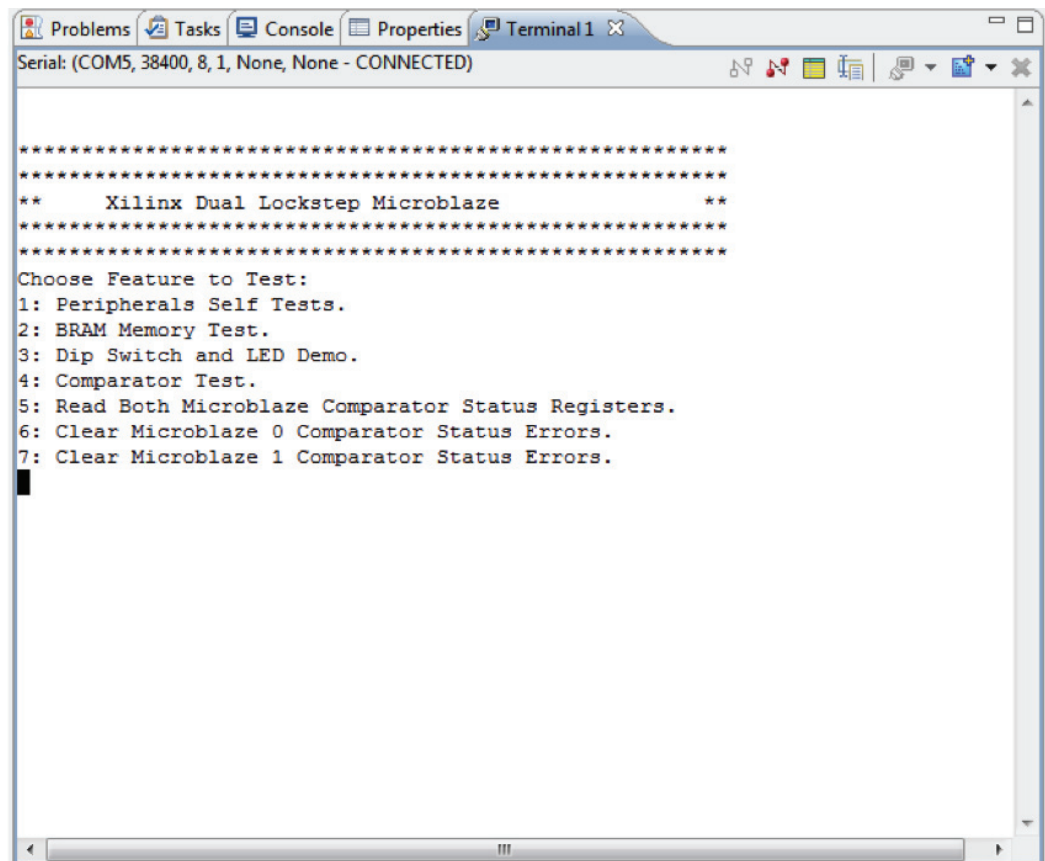




X584\_90\_040512

Figure 90: Program FPGA Window - Bootloader

- After the FPGA is programmed, go to the Terminal pane to view the serial output from the design. The SREC bootloader provides a status while it is loading the demonstration software. After the demonstration software is loaded from flash, the demonstration software prompt shows in the terminal screen (Figure 91).



X584\_91\_040512

Figure 91: Dual-Lockstep MicroBlaze Processor Demonstration Software Prompt in Terminal

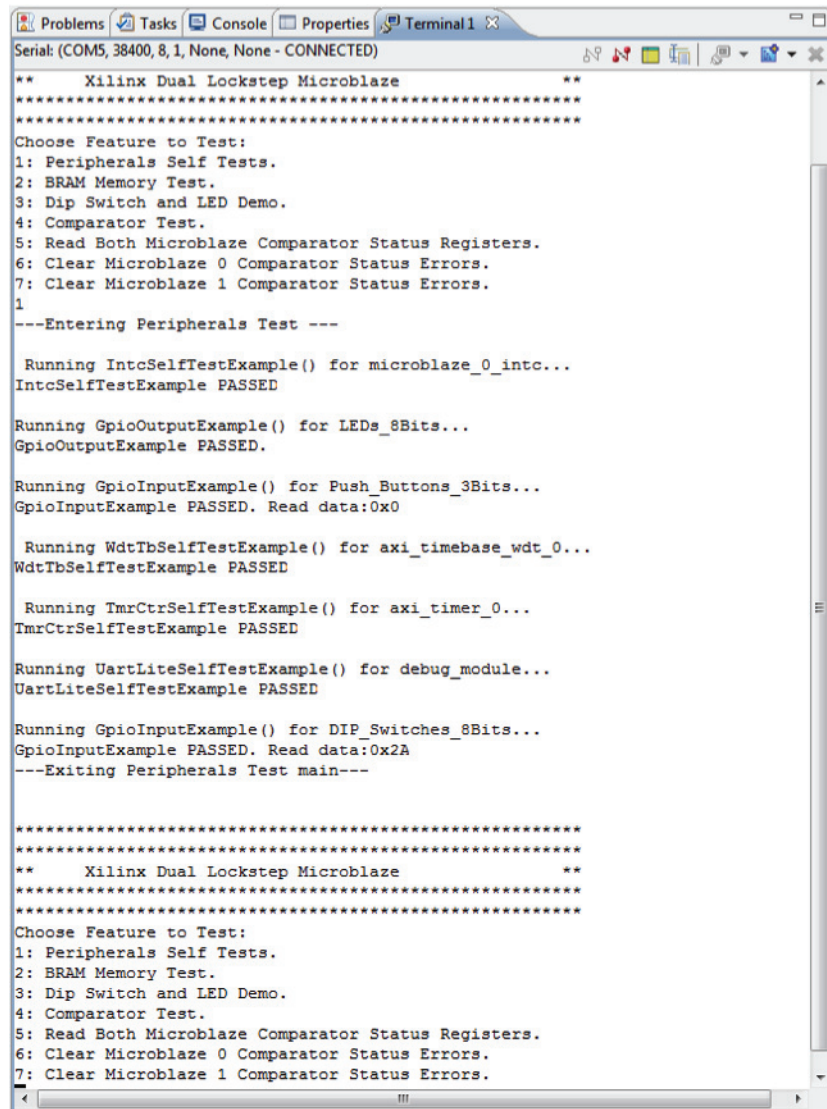
The Comparator Error outputs are routed to the two LEDs nearest the System ACE™ technology card slot on the board. If these two LEDs are not lit, the comparators are not indicating a comparison error (Figure 1, page 2).

## Running the Demonstration Software

The demonstration software provided with this application note runs seven different sub-applications to test and demonstrate the dual-lockstep MicroBlaze processor. To select a sub-application, the number of the sub-application should be entered into the terminal window.

### Peripherals Self-tests

The Peripherals Self-Test runs the peripherals test that was executed as part of the quick sanity check. Self-tests are run on the interrupt controller, six LEDs connected to the LED controller, pushbuttons, Timebase, timer, MDM, and DIP switches. A screen capture of the executed application is shown in Figure 92.



```

Serial: (COM5, 38400, 8, 1, None, None - CONNECTED)
**      Xilinx Dual Lockstep Microblaze      **
*****
*****
Choose Feature to Test:
1: Peripherals Self Tests.
2: BRAM Memory Test.
3: Dip Switch and LED Demo.
4: Comparator Test.
5: Read Both Microblaze Comparator Status Registers.
6: Clear Microblaze 0 Comparator Status Errors.
7: Clear Microblaze 1 Comparator Status Errors.
1
---Entering Peripherals Test ---

Running IntcSelfTestExample() for microblaze_0_intc...
IntcSelfTestExample PASSED

Running GpioOutputExample() for LEDs_8Bits...
GpioOutputExample PASSED.

Running GpioInputExample() for Push_Buttons_3Bits...
GpioInputExample PASSED. Read data:0x0

Running WdtTbSelfTestExample() for axi_timebase_wdt_0...
WdtTbSelfTestExample PASSED

Running TmrCtrSelfTestExample() for axi_timer_0...
TmrCtrSelfTestExample PASSED

Running UartLiteSelfTestExample() for debug_module...
UartLiteSelfTestExample PASSED

Running GpioInputExample() for DIP_Switches_8Bits...
GpioInputExample PASSED. Read data:0x2A
---Exiting Peripherals Test main---

*****
*****
**      Xilinx Dual Lockstep Microblaze      **
*****
*****
Choose Feature to Test:
1: Peripherals Self Tests.
2: BRAM Memory Test.
3: Dip Switch and LED Demo.
4: Comparator Test.
5: Read Both Microblaze Comparator Status Registers.
6: Clear Microblaze 0 Comparator Status Errors.
7: Clear Microblaze 1 Comparator Status Errors.

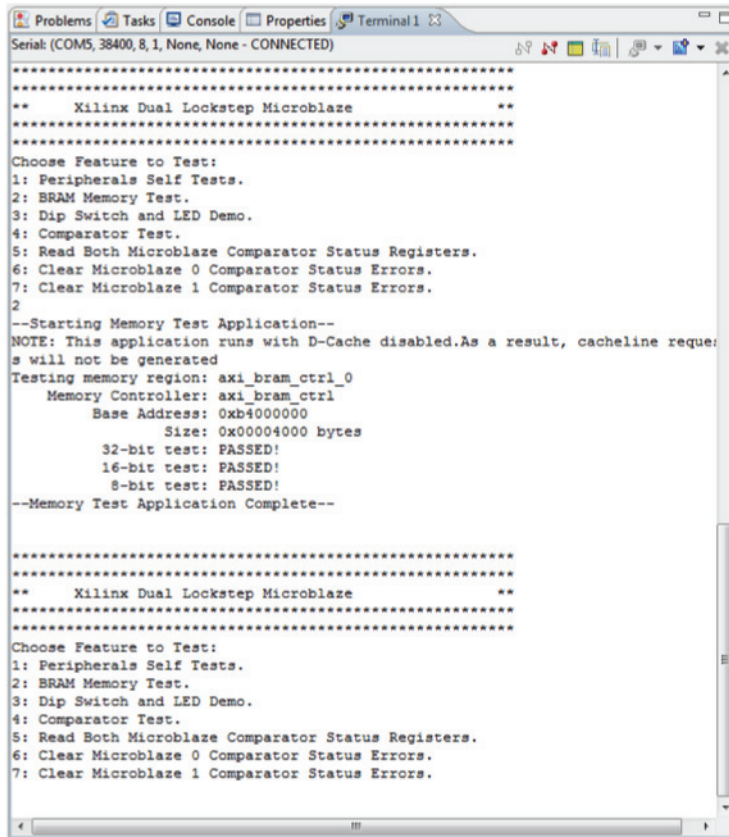
```

X584\_92\_040512

Figure 92: Peripherals Self-Tests

### BRAM Memory Test

The block RAM (BRAM) Memory test runs write and read tests on the AXI\_BRAM. A screen capture of the executed application is shown in [Figure 93](#).



```
Serial: (COM5, 38400, 8, 1, None, None - CONNECTED)
*****
**      Xilinx Dual Lockstep Microblaze      **
*****
Choose Feature to Test:
1: Peripherals Self Tests.
2: BRAM Memory Test.
3: Dip Switch and LED Demo.
4: Comparator Test.
5: Read Both Microblaze Comparator Status Registers.
6: Clear Microblaze 0 Comparator Status Errors.
7: Clear Microblaze 1 Comparator Status Errors.
2
--Starting Memory Test Application--
NOTE: This application runs with D-Cache disabled.As a result, cacheline requests will not be generated
Testing memory region: axi_bram_ctrl_0
Memory Controller: axi_bram_ctrl
Base Address: 0xb4000000
Size: 0x00004000 bytes
32-bit test: PASSED!
16-bit test: PASSED!
8-bit test: PASSED!
--Memory Test Application Complete--

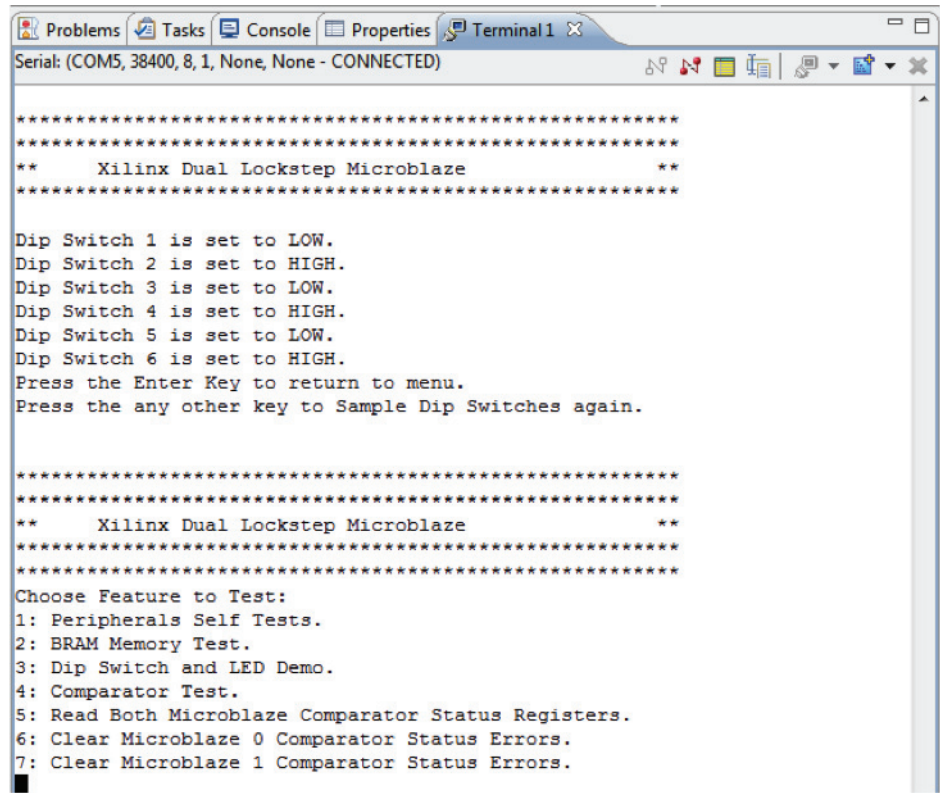
*****
**      Xilinx Dual Lockstep Microblaze      **
*****
Choose Feature to Test:
1: Peripherals Self Tests.
2: BRAM Memory Test.
3: Dip Switch and LED Demo.
4: Comparator Test.
5: Read Both Microblaze Comparator Status Registers.
6: Clear Microblaze 0 Comparator Status Errors.
7: Clear Microblaze 1 Comparator Status Errors.
```

X584\_93\_040512

Figure 93: BRAM Memory Test

### DIP Switch and LED Demonstration

The DIP switch and LED Demonstration reads DIP switches 1 through 6 (SW6) on the Avnet Spartan-6 FPGA LX150T development board and outputs the setting to the LEDs. After reporting the DIP switch settings, the application prompts the user to re-run the application or return to the demonstration application prompt. A screen capture of the executed application is shown in [Figure 94](#).



```
Serial: (COM5, 38400, 8, 1, None, None - CONNECTED)

*****
*****
**      Xilinx Dual Lockstep Microblaze      **
*****

Dip Switch 1 is set to LOW.
Dip Switch 2 is set to HIGH.
Dip Switch 3 is set to LOW.
Dip Switch 4 is set to HIGH.
Dip Switch 5 is set to LOW.
Dip Switch 6 is set to HIGH.
Press the Enter Key to return to menu.
Press the any other key to Sample Dip Switches again.

*****
*****
**      Xilinx Dual Lockstep Microblaze      **
*****

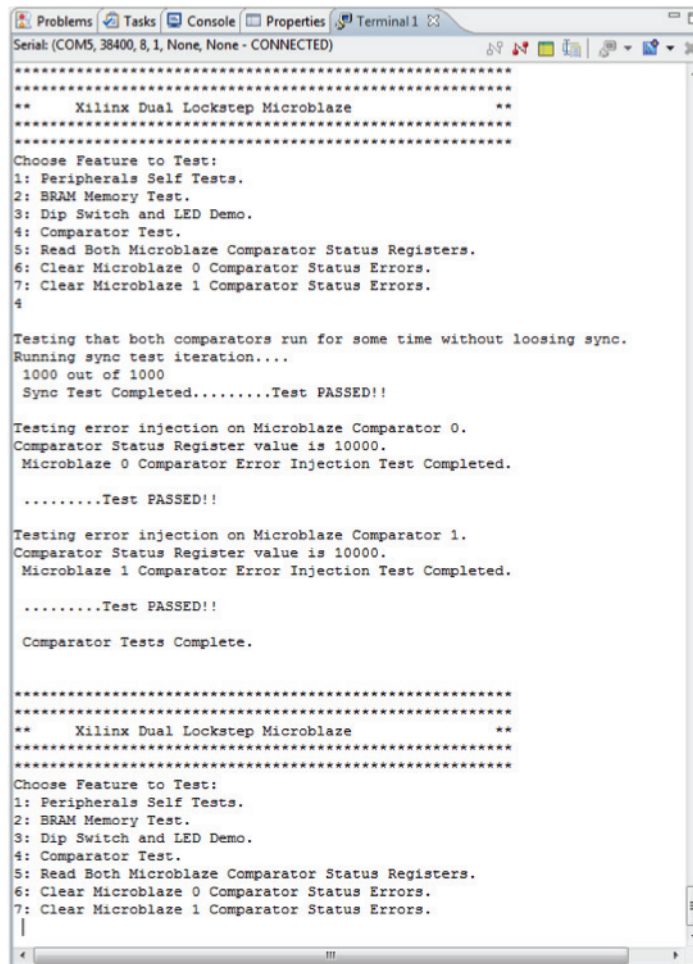
Choose Feature to Test:
1: Peripherals Self Tests.
2: BRAM Memory Test.
3: Dip Switch and LED Demo.
4: Comparator Test.
5: Read Both Microblaze Comparator Status Registers.
6: Clear Microblaze 0 Comparator Status Errors.
7: Clear Microblaze 1 Comparator Status Errors.
█
```

X584\_94\_040512

Figure 94: DIP Switch and LED Demo

### Comparator Test

The Comparator test checks the overall function of the two MicroBlaze Comparators, including error injection. The first part of the test runs code for a time and monitors both MicroBlaze Comparators to see that no comparator error is indicated. This part of the test basically checks that the MicroBlaze processors remain in lockstep. The second part of the test checks that the error injection on all the bits in all the connected buses works and that the status register clears when commanded. In this system, the connected buses are AXI\_IC, AXI\_DC, AXI\_IP, AXI\_DP, ILMB, and DLMB. This check is done for both interfaces of each MicroBlaze Comparator. A screen capture of the executed application is shown in [Figure 95](#).



```
*****  
**      Xilinx Dual Lockstep Microblaze      **  
*****  
Choose Feature to Test:  
1: Peripherals Self Tests.  
2: BRAM Memory Test.  
3: Dip Switch and LED Demo.  
4: Comparator Test.  
5: Read Both Microblaze Comparator Status Registers.  
6: Clear Microblaze 0 Comparator Status Errors.  
7: Clear Microblaze 1 Comparator Status Errors.  
4  
  
Testing that both comparators run for some time without loosing sync.  
Running sync test iteration...  
1000 out of 1000  
Sync Test Completed.....Test PASSED!!  
  
Testing error injection on Microblaze Comparator 0.  
Comparator Status Register value is 10000.  
Microblaze 0 Comparator Error Injection Test Completed.  
  
.....Test PASSED!!  
  
Testing error injection on Microblaze Comparator 1.  
Comparator Status Register value is 10000.  
Microblaze 1 Comparator Error Injection Test Completed.  
  
.....Test PASSED!!  
  
Comparator Tests Complete.  
  
*****  
**      Xilinx Dual Lockstep Microblaze      **  
*****  
Choose Feature to Test:  
1: Peripherals Self Tests.  
2: BRAM Memory Test.  
3: Dip Switch and LED Demo.  
4: Comparator Test.  
5: Read Both Microblaze Comparator Status Registers.  
6: Clear Microblaze 0 Comparator Status Errors.  
7: Clear Microblaze 1 Comparator Status Errors.  
|
```

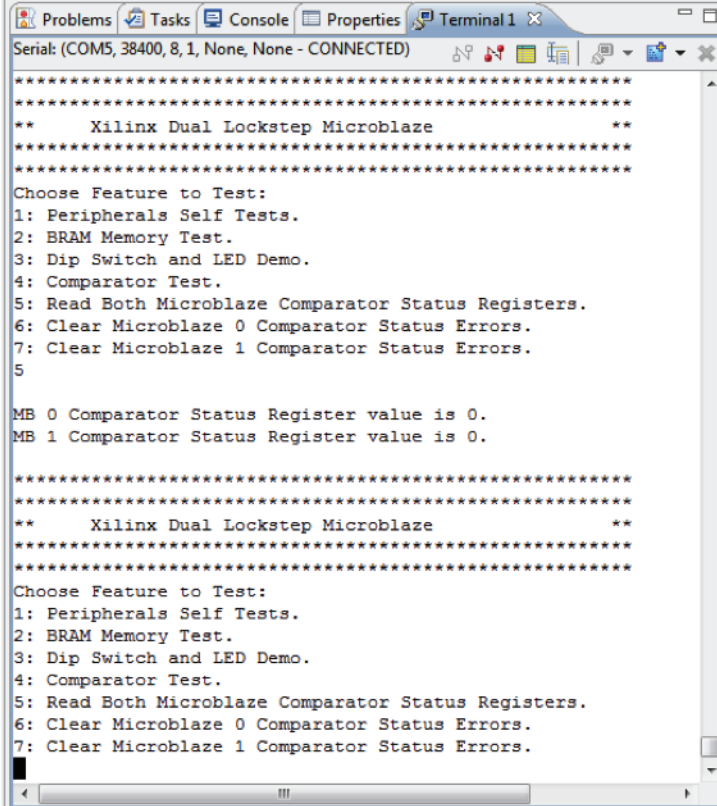
X584\_95\_040512

Figure 95: Comparator Test

### ***Read Both MicroBlaze Comparator Status Registers***

When Read Both MicroBlaze Comparator Status Registers is executed, the status register for each MicroBlaze comparator is read and printed to the screen. A value other than zero indicates that a comparator error was detected. A screen capture of the executed application is shown in [Figure 96](#).





```
Serial: (COM5, 38400, 8, 1, None, None - CONNECTED)
*****
**      Xilinx Dual Lockstep Microblaze      **
*****
Choose Feature to Test:
1: Peripherals Self Tests.
2: BRAM Memory Test.
3: Dip Switch and LED Demo.
4: Comparator Test.
5: Read Both Microblaze Comparator Status Registers.
6: Clear Microblaze 0 Comparator Status Errors.
7: Clear Microblaze 1 Comparator Status Errors.
5

MB 0 Comparator Status Register value is 0.
MB 1 Comparator Status Register value is 0.

*****
**      Xilinx Dual Lockstep Microblaze      **
*****
Choose Feature to Test:
1: Peripherals Self Tests.
2: BRAM Memory Test.
3: Dip Switch and LED Demo.
4: Comparator Test.
5: Read Both Microblaze Comparator Status Registers.
6: Clear Microblaze 0 Comparator Status Errors.
7: Clear Microblaze 1 Comparator Status Errors.
```

X584\_96\_040512

Figure 96: Read Both MicroBlaze Comparator Status Registers

#### ***Clear MicroBlaze X Comparator Status Errors.***

Running the Clear MicroBlaze 0 or 1 Comparator Status Error application writes to the Clear Fault bit of the control register and verifies that the status register is cleared for the corresponding MicroBlaze processor. A screen capture of the executed application run on MicroBlaze 0 is shown in [Figure 97](#).



```

Serial: (COM5, 38400, 8, 1, None, None - CONNECTED)

*****
*****
**      Xilinx Dual Lockstep Microblaze      **
*****
*****
Choose Feature to Test:
1: Peripherals Self Tests.
2: BRAM Memory Test.
3: Dip Switch and LED Demo.
4: Comparator Test.
5: Read Both Microblaze Comparator Status Registers.
6: Clear Microblaze 0 Comparator Status Errors.
7: Clear Microblaze 1 Comparator Status Errors.
6

MB 0 Comparator Status Register value after clear is 0.

*****
*****
**      Xilinx Dual Lockstep Microblaze      **
*****
*****
Choose Feature to Test:
1: Peripherals Self Tests.
2: BRAM Memory Test.
3: Dip Switch and LED Demo.
4: Comparator Test.
5: Read Both Microblaze Comparator Status Registers.
6: Clear Microblaze 0 Comparator Status Errors.
7: Clear Microblaze 1 Comparator Status Errors.

```

X584\_97\_040512

Figure 97: Clear MicroBlaze 0 Comparator Status Errors

## Disconnect the MicroBlaze Processors and Debug Logic

Figure 98 shows the reference design progress to this point.

- |                                     |   |
|-------------------------------------|---|
| <input checked="" type="checkbox"/> | 1. Generate Dual MicroBlaze System in EDK Platform Studio.                  |
| <input checked="" type="checkbox"/> | 2. Modify Dual MicroBlaze System into Lockstep through EDK Platform Studio. |
| <input checked="" type="checkbox"/> | 3. Build the Dual-Lockstep MicroBlaze System in EDK Platform Studio.        |
| <input checked="" type="checkbox"/> | 4. Perform a Quick Sanity Check of the Design.                              |
| <input checked="" type="checkbox"/> | 5. Prepare Dual-Lockstep MicroBlaze Design Source for Isolated Design.      |
| <input checked="" type="checkbox"/> | 6. Synthesize and Floorplan Hierarchical Design.                            |
| <input checked="" type="checkbox"/> | 7. Run IVT on the Design in UCF Mode.                                       |
| <input checked="" type="checkbox"/> | 8. Implement the Design.  |
| <input checked="" type="checkbox"/> | 9. Run IVT on the Design in NCD Mode.                                       |
| <input checked="" type="checkbox"/> | 10. Build Final Software in SDK.  |
| <input type="checkbox"/>            | 11. Disconnect the MicroBlaze Processors and Debug Logic.                   |
| <input type="checkbox"/>            | 12. Re-verify the Re-implemented Design in IVT.                             |

X584\_98\_041112

Figure 98: Reference Design Progress

The Lockstep Master bus that connects the two MicroBlaze processors can be removed so that the MicroBlaze processors are completely independent of each other. For the next phase of the application note, design preservation techniques are used to lock down and preserve all the isolated functions but disconnect the Lockstep Master bus between the two MicroBlaze processors. For added effect, the MDM is disconnected from the first MicroBlaze processor.

More information about design preservation can be found in *Repeatable Results with Design Preservation* [Ref 7] and *Xilinx Hierarchical Design Methodology Guide* [Ref 6]. A tutorial is also available in *Design Preservation Tutorial: PlanAhead Design Tool* [Ref 8].

**Note:** When this application note was started, the initial plan was to completely remove the MDM from the design at this phase. Locking down all the isolated regions and completely removing the MDM would have required that the MDM be included in its own isolated function and region. As a result of the Spartan-6 FPGA LX150T Development Board pinout and the limitation it put on the floorplan, the MDM had to be included in the U4\_peripherals isolated function. In this case, completely removing the MDM would have required that the U4\_peripherals isolated function be reimplemented, which would have defeated the purpose of executing preservation on all the isolated regions.

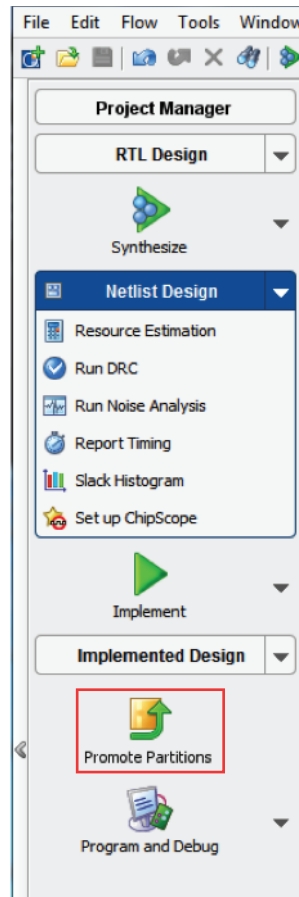
### Promoting Partitions

The first step of design preservation is promoting the partitions within the PlanAhead tool so that the netlists of the promoted partitions (isolated regions) can be pulled back in when the design is re-implemented. These steps describe how to promote the partitions for all the isolated functions and regions within the dual-lockstep MicroBlaze processor system:

1. If it is closed, reopen the PlanAhead tool project that implemented the isolated dual-lockstep MicroBlaze processor system.

**Note:** If the PlanAhead tool was closed, the netlist design has to be reopened, and the SCC\_ISOLATED attribute has to be re-enabled on each of the isolated functions. This is a bug in the PlanAhead tool. If the SCC\_ISOLATED attribute shows that it is enabled after opening the netlist, disable it, click **Apply**, re-enable it, and click **Apply** to ensure that the PlanAhead tool's attribute database is in sync with the design.

2. To lock down the isolated regions, click **Promote Partitions** on the left side of the PlanAhead tool window (see [Figure 99](#)).

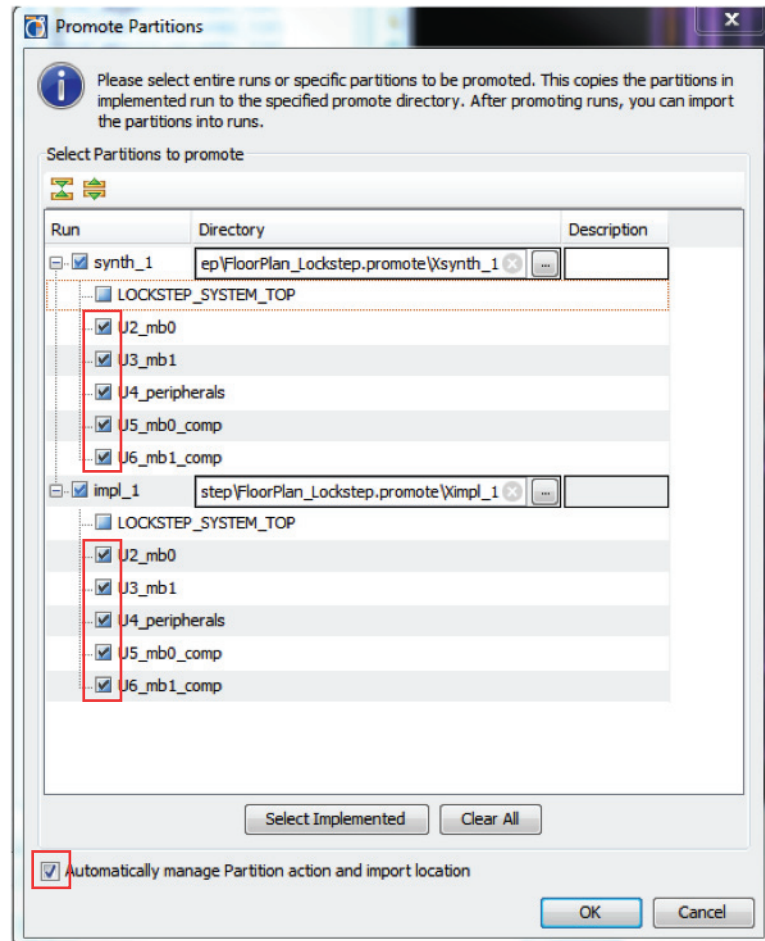


X584\_99\_040512

Figure 99: Promote Partitions Button

3. In the Promote Partitions window (see Figure 100), verify that **U2\_mb0**, **U3\_mb1**, **U4\_peripherals**, **U5\_mb0\_comp**, and **U6\_mb1\_comp** are all selected under the `synth` and `imp` runs to promote the synthesis and implementation runs. Check **Automatically manage Partition action and Import location** to have the PlanAhead tool automatically manage the files.

**Note:** Do not select the **LOCKSTEP\_SYSTEM\_TOP**. The disconnection of the buses occurs at the top level of the design, so the top has to be re-implemented.



X584\_100\_040512

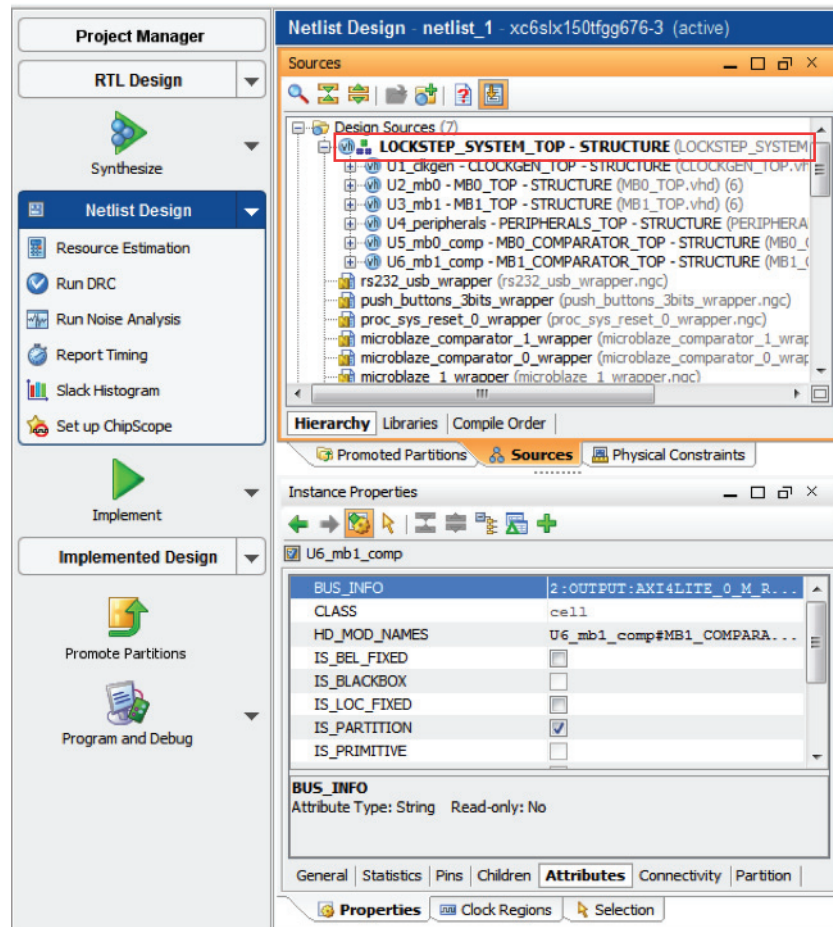
Figure 100: Promote Partitions Window

4. Click **OK** to promote and store the isolated region netlists.

### Removing the Lockstep Master and MDM Connections

Removing the Lockstep Master Bus and the MDM connections requires disconnecting the buses in the `LOCKSTEP_SYSTEM_TOP.vhd` file and adding some constraints to the design's UCF. The next section defines the changes to the `LOCKSTEP_SYSTEM_TOP.vhd` file and the additions to the `lockstep_system.ucf` file.

1. Open the Sources tab within the PlanAhead tool window (Figure 101) by clicking **Window > Sources**.
2. Double-click the **LOCKSTEP\_SYSTEM\_TOP - STRUCTURE** line under the **Design Sources** tree within the Sources tab to open the `LOCKSTEP_SYSTEM_TOP.vhd` file within the PlanAhead tool's text editor (Figure 101).



X584\_101\_040512

Figure 101: LOCKSTEP\_SYSTEM\_TOP within Sources Tab

3. Within the LOCKSTEP\_SYSTEM\_TOP.vhd file, scroll down to Line 1024 and change lines 1024 to 1026 from:

```
EXT_BRK_IN           => EXT_BRK_MB0,
EXT_NM_BRK_IN       => EXT_NM_BRK_MB0,
LOCKSTEP_MASTER_OUT => LOCKSTEP_MASTER,
```

to:

```
EXT_BRK_IN           => '0',
EXT_NM_BRK_IN       => '0',
LOCKSTEP_MASTER_OUT => OPEN,
```

This disconnects the two External Break inputs and the Lockstep Master output of U2\_mb0.

**Note:** **Ctrl+g** can be used within the text editor to jump to lines.

4. Within the LOCKSTEP\_SYSTEM\_TOP.vhd file, scroll down to Line 1189 and change lines 1189 through 1196 from:

```
MB_DBG_CLK_IN       => MB_DBG_CLK,
MB_DBG_TDI_IN       => MB_DBG_TDI,
MB_DBG_TDO_OUT      => MB_DBG_TDO,
MB_DBG_REG_EN_IN    => MB_DBG_REG_EN,
MB_DBG_CAPTURE_IN   => MB_DBG_CAPTURE,
MB_DBG_SHIFT_IN     => MB_DBG_SHIFT,
MB_DBG_UPDATE_IN    => MB_DBG_UPDATE,
MB_DEBUG_RST_IN     => MB_DEBUG_RST
```

to:

```

MB_DBG_CLK_IN           => '0',
MB_DBG_TDI_IN           => '0',
MB_DBG_TDO_OUT         => OPEN,
MB_DBG_REG_EN_IN       => x"00",
MB_DBG_CAPTURE_IN      => '0',
MB_DBG_SHIFT_IN        => '0',
MB_DBG_UPDATE_IN       => '0',
MB_DEBUG_RST_IN        => '0'

```

This disconnects the MDM connections of U2\_mb0.

5. Within the LOCKSTEP\_SYSTEM\_TOP.vhd file, scroll down to Line 1198 and add in this code line to tie the LOCKSTEP\_MASTER input to U3\_mb1:

```
LOCKSTEP_MASTER <= (OTHERS => '0');
```

6. Within the LOCKSTEP\_SYSTEM\_TOP.vhd file, scroll down to Line 1205 and change line 1205 and 1206 from:

```

EXT_BRK_IN              => EXT_BRK_MB1,
EXT_NM_BRK_IN           => EXT_NM_BRK_MB1,

```

to:

```

EXT_BRK_IN              => '0',
EXT_NM_BRK_IN           => '0',

```

This disconnects the two External Break inputs and the Lockstep Master output of U3\_mb1.

7. Within the LOCKSTEP\_SYSTEM\_TOP.vhd file, scroll down to Line 1277 and change lines 1277 through 1280 from:

```

EXT_BRK_MB0_OUT        => EXT_BRK_MB0,
EXT_BRK_MB1_OUT        => EXT_BRK_MB1,
EXT_NM_BRK_MB0_OUT     => EXT_NM_BRK_MB0,
EXT_NM_BRK_MB1_OUT     => EXT_NM_BRK_MB1,

```

to:

```

EXT_BRK_MB0_OUT        => OPEN,
EXT_BRK_MB1_OUT        => OPEN,
EXT_NM_BRK_MB0_OUT     => OPEN,
EXT_NM_BRK_MB1_OUT     => OPEN,

```

This disconnects the External Break outputs of U4\_peripherals.

8. Within the LOCKSTEP\_SYSTEM\_TOP.vhd file, scroll down to Line 1288 and change line 1288 and 1289 from:

```

DEBUG_MODULE_DRCK_IN   => DEBUG_MODULE_DRCK_IN,
DEBUG_MODULE_DRCK_OUT  => DEBUG_MODULE_DRCK_OUT,

```

to:

```

DEBUG_MODULE_DRCK_IN   => '0',
DEBUG_MODULE_DRCK_OUT  => OPEN,

```

This disconnects the U4\_peripherals component from the MDM DRCK\_BUFG instantiated within LOCKSTEP\_SYSTEM\_TOP.vhd.

9. Within the LOCKSTEP\_SYSTEM\_TOP.vhd file, scroll down to Line 1538 and change lines 1538 through 1545 from:

```

MB_DBG_CLK_OUT         => MB_DBG_CLK,
MB_DBG_TDI_OUT         => MB_DBG_TDI,
MB_DBG_TDO_IN          => MB_DBG_TDO,
MB_DBG_REG_EN_OUT      => MB_DBG_REG_EN,
MB_DBG_CAPTURE_OUT     => MB_DBG_CAPTURE,
MB_DBG_SHIFT_OUT       => MB_DBG_SHIFT,
MB_DBG_UPDATE_OUT      => MB_DBG_UPDATE,

```



```

MB_DEBUG_RST_OUT      => MB_DEBUG_RST,
to:
MB_DBG_CLK_OUT        => OPEN,
MB_DBG_TDI_OUT        => MB_DBG_TDI,
MB_DBG_TDO_IN         => MB_DBG_TDI,
MB_DBG_REG_EN_OUT     => OPEN,
MB_DBG_CAPTURE_OUT    => OPEN,
MB_DBG_SHIFT_OUT      => OPEN,
MB_DBG_UPDATE_OUT     => OPEN,
MB_DEBUG_RST_OUT      => OPEN,

```

This disconnects the MDM outputs of U2\_mb0 and loops the TDI output to TDO for JTAG connectivity.

10. Within the LOCKSTEP\_SYSTEM\_TOP.vhd file, scroll down to Line 1578 and comment out the BUFG\_DRK instance.
11. Save the updated LOCKSTEP\_SYSTEM\_TOP.vhd file.
12. Within the Sources tab, scroll down and expand the Constraints tree and double-click lockstep\_system.ucf to open the file within the PlanAhead tool's text editor.
13. At the bottom of lockstep\_system.ucf, add in these constraints:

```

PIN "U3_mb1.LOCKSTEP_MASTER_IN(*)" ALLOW_CONSTANT_PUSHING = FALSE;
PIN "U3_mb1.EXT_BRK_IN"           ALLOW_CONSTANT_PUSHING = FALSE;
PIN "U2_mb0.EXT_BRK_IN"           ALLOW_CONSTANT_PUSHING = FALSE;
PIN "U3_mb1.EXT_NM_BRK_IN"        ALLOW_CONSTANT_PUSHING = FALSE;
PIN "U2_mb0.EXT_NM_BRK_IN"        ALLOW_CONSTANT_PUSHING = FALSE;
PIN "U2_mb0.MB_DBG_CLK_IN"        ALLOW_CONSTANT_PUSHING = FALSE;
PIN "U2_mb0.MB_DBG_TDI_IN"        ALLOW_CONSTANT_PUSHING = FALSE;
PIN "U2_mb0.MB_DBG_REG_EN_IN(*)"  ALLOW_CONSTANT_PUSHING = FALSE;
PIN "U2_mb0.MB_DBG_CAPTURE_IN"    ALLOW_CONSTANT_PUSHING = FALSE;
PIN "U2_mb0.MB_DBG_SHIFT_IN"      ALLOW_CONSTANT_PUSHING = FALSE;
PIN "U2_mb0.MB_DBG_UPDATE_IN"     ALLOW_CONSTANT_PUSHING = FALSE;
PIN "U2_mb0.MB_DEBUG_RST_IN"      ALLOW_CONSTANT_PUSHING = FALSE;
PIN "U4_peripherals.DEBUG_MODULE_DRCK_IN" ALLOW_CONSTANT_PUSHING = FALSE;

```

These constraints prevent the Xilinx Implementation tools from modifying the isolated function due to the inputs that are now connected to constants (High or Low) instead of active drivers. If these constraints were not added, **ngdbuild** would report an error stating that the isolated function had changed.

The constraint has the format:

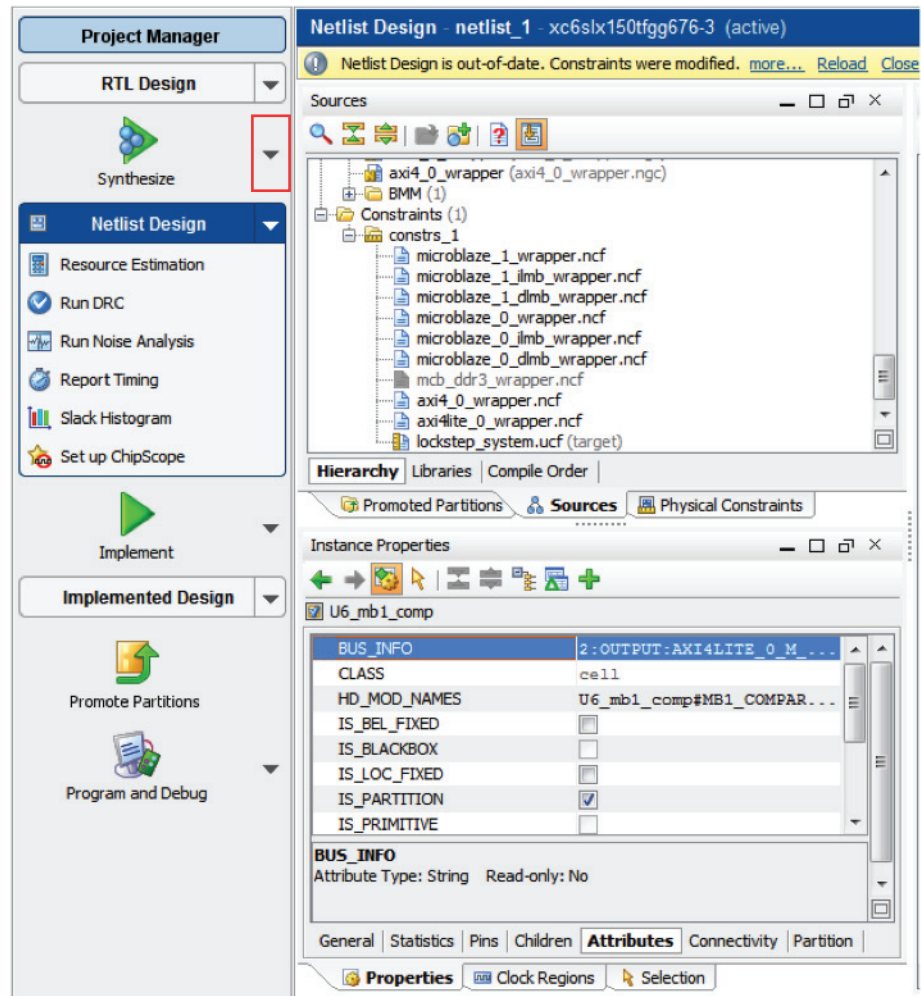
```
PIN "Partition_Instance.Partition_Pin" ALLOW_CONSTANT_PUSHING = FALSE;
```

14. Save the updated lockstep\_system.ucf file.
15. If prompted that the Netlist Design is out of date, reload the Netlist by clicking **Reload**.

## Re-implementing the Design with Promoted Partitions

With the Lockstep Master and MDM now disconnected in the code, the design can now be re-implemented. The design has to go through the full implementation flow including synthesis because of the changes to the top and constraint files. These steps define how to indicate to use the promoted partitions and re-implement the design:

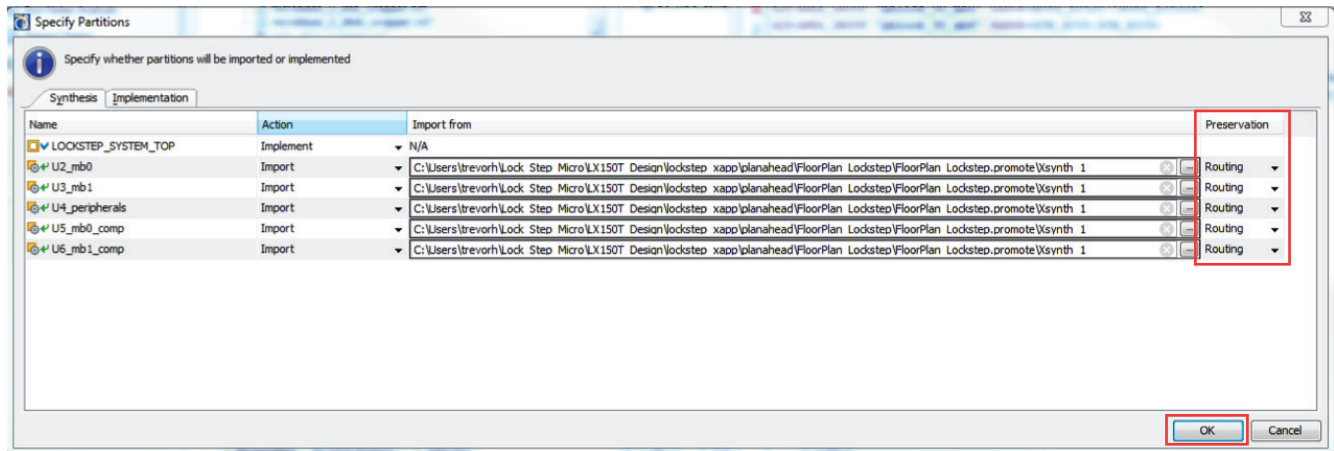
1. In the PlanAhead tool window ([Figure 102](#)), click the down arrow next to Synthesize and select **Specify Partitions....**



X584\_102\_040512

Figure 102: Synthesize Menu Selections

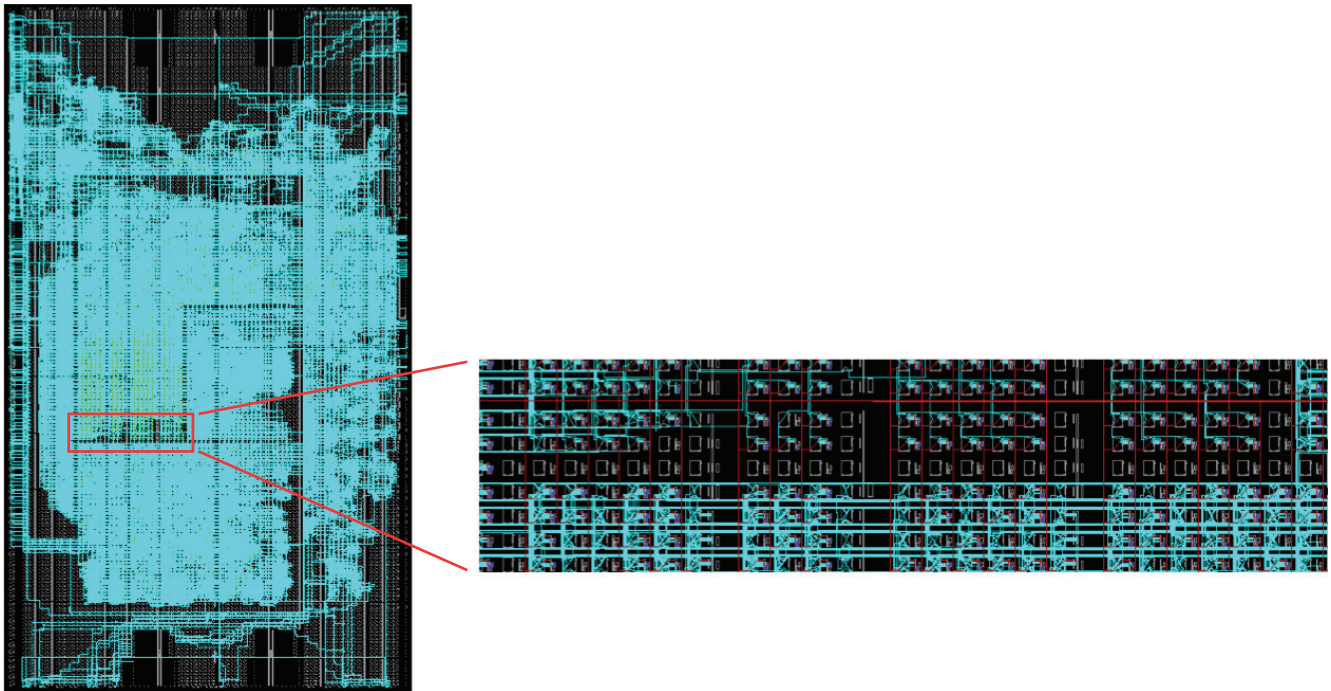
- In the Specify Partitions window (Figure 103), verify that all the partitions **U2\_mb0**, **U3\_mb1**, **U4\_peripherals**, **U5\_mb0\_comp**, and **U6\_mb1\_comp** have an action of **Import** and a preservation of **Routing** on both the Synthesis and Implementation tabs. Click **OK**. The import location is automatically set to where the promoted partitions were stored.



X584\_103\_040512

Figure 103: Promote Partitions Window

3. Synthesize the design by clicking the **Synthesize** button on the left side of the PlanAhead tool window.
4. When prompted whether to rerun synthesis, click **OK**.
5. After Synthesis is completed, open the **Netlist Design**.
6. With the Netlist Design open, click **Implement** on the left side of the PlanAhead tool window to implement the design.
7. When implementation is complete, verify that the design met timing. The NCD can also be opened in FPGA Editor to see that the Lockstep Master bus has been disconnected between the two MicroBlaze processors and that only the global clock crosses the two isolated functions. [Figure 104](#), as an example, shows where the Lockstep Master bus used to connect between the two processors. The signal highlighted in red is the global 50 MHz clock, which is the only signal that remains.



X584\_104\_040512

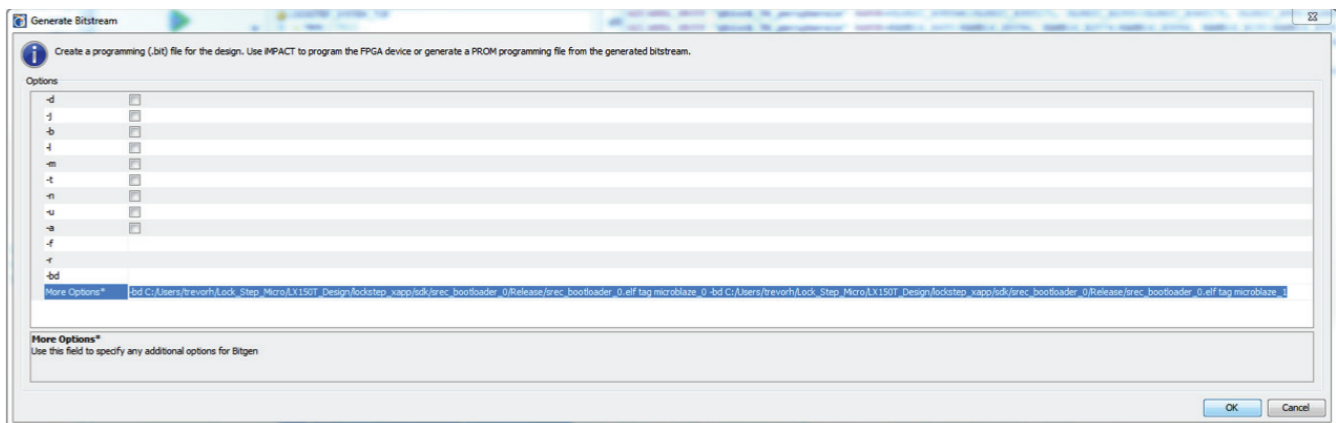
Figure 104: u2\_mb0 and U3\_mb1 Isolated Function Boundary with Master Lockstep Removed

### Building the Bitstream with the Bootloader Embedded

The bitstream can be built through the PlanAhead tool so that the bootloader is added to the MicroBlaze processor Data and Instruction memory without having to go through SDK. This is done through BitGen using the `-bd` flag identifying the ELF file coupled with the tag command to identify the MicroBlaze processor. These steps describe this process:

1. With the design re-implemented, click the down arrow next to Program and Debug on the left side of the PlanAhead tool window. Select **Generate Bitstream....**
2. In **Generate Bitstream**, use the **More Options** field (Figure 105), which allows multiple flag inputs. Add in these lines, then click **OK**:

```
-bd <reference design>/sdk/src_bootloader_0/Release/src_bootloader_0.elf tag microblaze_0
-bd <reference design>/sdk/src_bootloader_0/Release/src_bootloader_0.elf tag microblaze_1
```

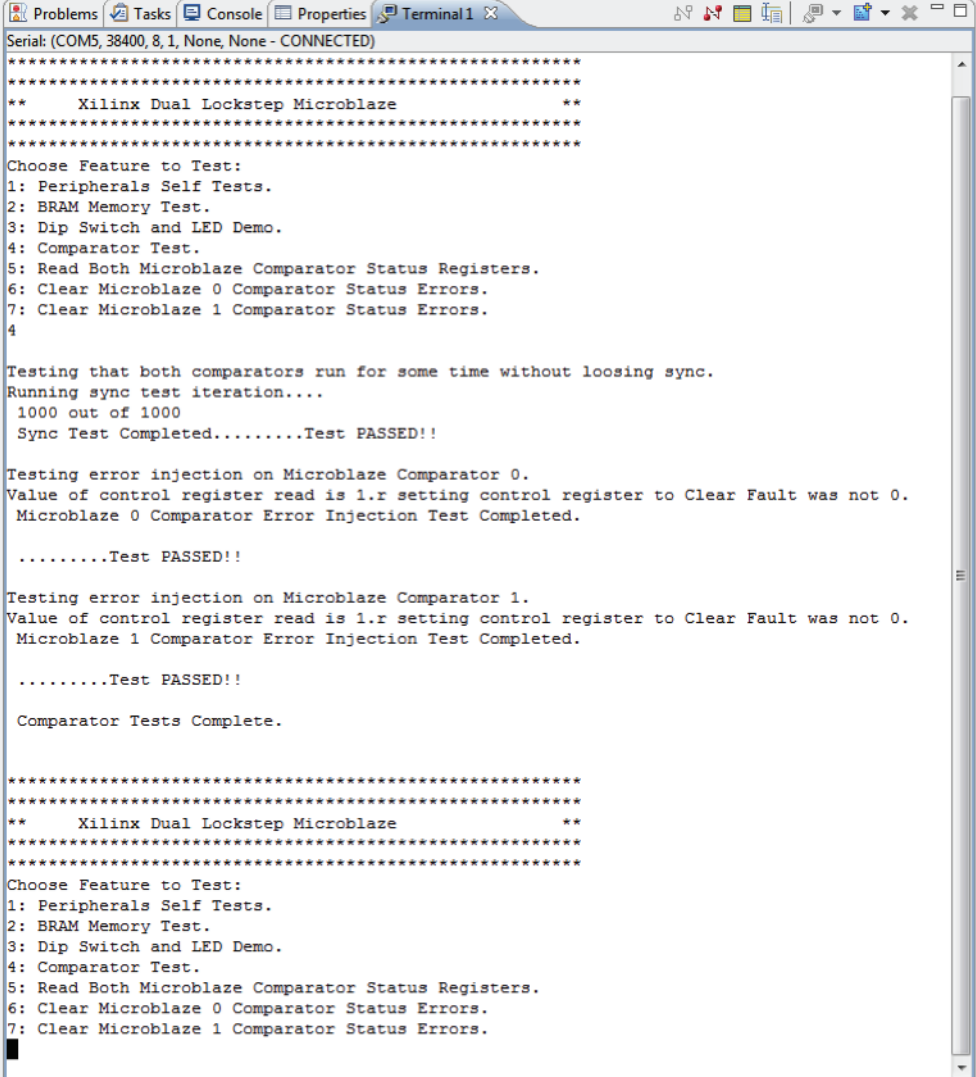


X584\_105\_040512

Figure 105: Generate Bitstream with ELF Callout

- After BitGen is complete, the generated file can be loaded into the FPGA through the iMPACT tool (a foundational tool in the ISE Design Suite). This FPGA load brings up the demonstration code, which is still stored in the linear flash.

Figure 16 shows a screen capture of the demonstration software's Comparator test successfully executed on the re-implemented design.



```
Serial: (COM5, 38400, 8, 1, None, None - CONNECTED)
*****
**      Xilinx Dual Lockstep Microblaze      **
*****
Choose Feature to Test:
1: Peripherals Self Tests.
2: BRAM Memory Test.
3: Dip Switch and LED Demo.
4: Comparator Test.
5: Read Both Microblaze Comparator Status Registers.
6: Clear Microblaze 0 Comparator Status Errors.
7: Clear Microblaze 1 Comparator Status Errors.
4

Testing that both comparators run for some time without loosing sync.
Running sync test iteration....
1000 out of 1000
Sync Test Completed.....Test PASSED!!

Testing error injection on Microblaze Comparator 0.
Value of control register read is 1.r setting control register to Clear Fault was not 0.
Microblaze 0 Comparator Error Injection Test Completed.

.....Test PASSED!!

Testing error injection on Microblaze Comparator 1.
Value of control register read is 1.r setting control register to Clear Fault was not 0.
Microblaze 1 Comparator Error Injection Test Completed.

.....Test PASSED!!

Comparator Tests Complete.

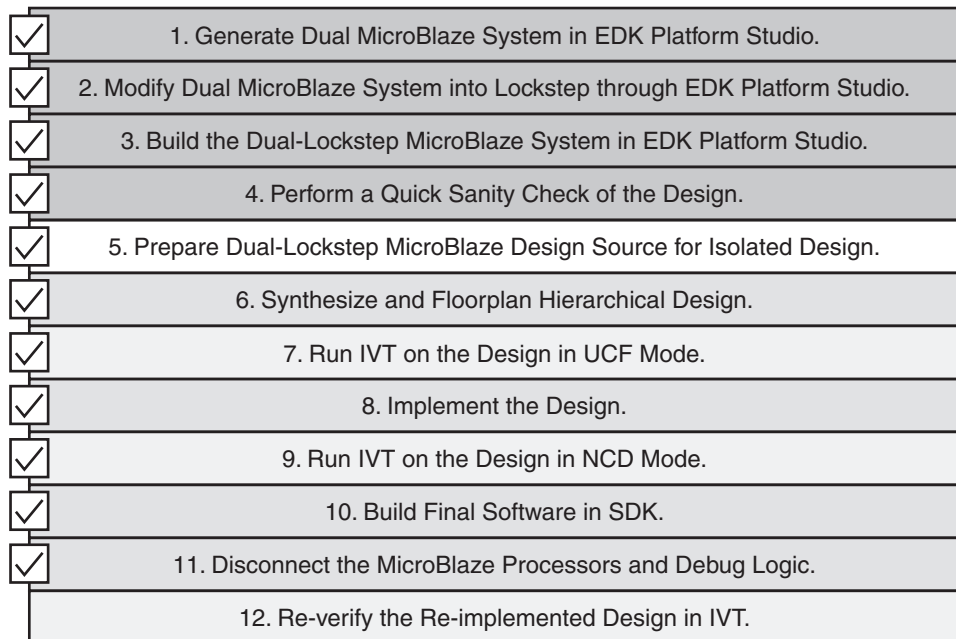
*****
**      Xilinx Dual Lockstep Microblaze      **
*****
Choose Feature to Test:
1: Peripherals Self Tests.
2: BRAM Memory Test.
3: Dip Switch and LED Demo.
4: Comparator Test.
5: Read Both Microblaze Comparator Status Registers.
6: Clear Microblaze 0 Comparator Status Errors.
7: Clear Microblaze 1 Comparator Status Errors.
```

X584\_106\_040512

Figure 106: Snapshot of Comparator Test Run on Preserved Build with Embedded ELF

## Re-verify the Re-implemented Design in the IVT

Figure 107 shows the reference design progress to this point.



X584\_107\_041112

Figure 107: Reference Design Progress

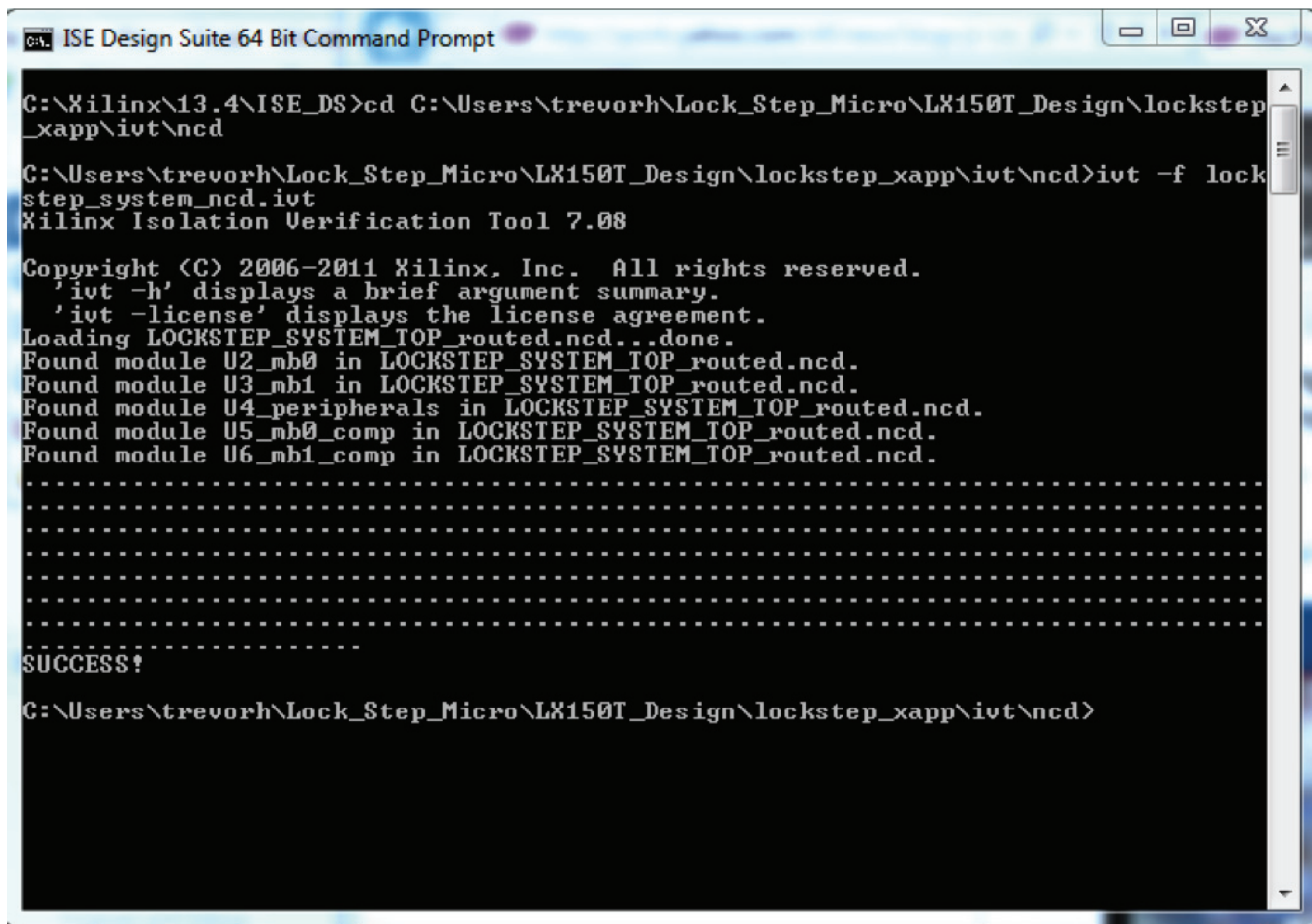
With the design re-implemented, run the NCD through the IVT one last time to verify that there are no isolation violations.

1. In the PlanAhead tool, in the Design Runs pane, left-click and then right-click **impl\_1**.
2. In the pop-up window, select **Open Run Directory....**
3. In the Explorer window that appears, copy the `LOCKSTEP_SYSTEM_TOP_routed.ncd` file to `<reference design>\ivt\ncd`.
4. Open up a ISE Design Suite command prompt.
 

**Start > All Programs > Xilinx ISE Design Suite 13.4 > Accessories > ISE Design Suite 64 Bit Command Prompt.**

**Note:** This command launches the 64-bit Windows version of the tool.
5. Within the command prompt, change directory (**cd**) to `<reference design>\ivt\ncd`.
6. Within the command prompt, enter `ivt -f lockstep_system.ncd.ivt`. Press **Enter**.
7. After the IVT runs, verify that the status says `SUCCESS!`, indicating that no isolation violations were found (Figure 108).



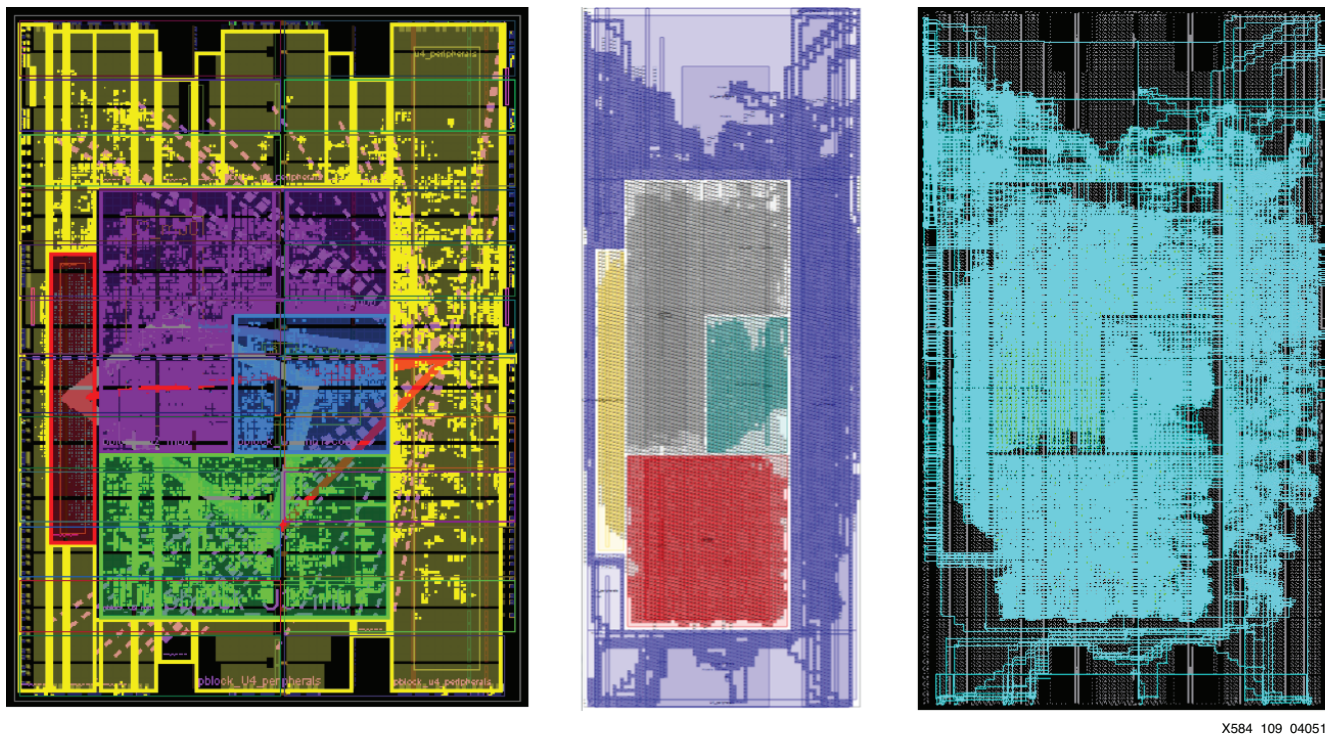
A screenshot of a Windows Command Prompt window titled "ISE Design Suite 64 Bit Command Prompt". The window shows the execution of the "ivt" command in NCD mode. The command prompt is at "C:\Users\trevorh\Lock\_Step\_Micro\LX150T\_Design\lockstep\_xapp\ivt\ncd". The user enters "ivt -f lockstep\_system\_ncd.ivt". The output shows "Xilinx Isolation Verification Tool 7.08", copyright information for Xilinx, Inc. (2006-2011), and instructions for using the tool. It then lists several modules found in the "LOCKSTEP\_SYSTEM\_TOP\_routed.ncd" file, including "U2\_mb0", "U3\_mb1", "U4\_peripherals", "U5\_mb0\_comp", and "U6\_mb1\_comp". After a series of dots, it displays "SUCCESS!". The prompt then returns to "C:\Users\trevorh\Lock\_Step\_Micro\LX150T\_Design\lockstep\_xapp\ivt\ncd>".

```
C:\Xilinx\13.4\ISE_DS>cd C:\Users\trevorh\Lock_Step_Micro\LX150T_Design\lockstep_xapp\ivt\ncd
C:\Users\trevorh\Lock_Step_Micro\LX150T_Design\lockstep_xapp\ivt\ncd>ivt -f lockstep_system_ncd.ivt
Xilinx Isolation Verification Tool 7.08
Copyright (C) 2006-2011 Xilinx, Inc. All rights reserved.
'ivt -h' displays a brief argument summary.
'ivt -license' displays the license agreement.
Loading LOCKSTEP_SYSTEM_TOP_routed.ncd...done.
Found module U2_mb0 in LOCKSTEP_SYSTEM_TOP_routed.ncd.
Found module U3_mb1 in LOCKSTEP_SYSTEM_TOP_routed.ncd.
Found module U4_peripherals in LOCKSTEP_SYSTEM_TOP_routed.ncd.
Found module U5_mb0_comp in LOCKSTEP_SYSTEM_TOP_routed.ncd.
Found module U6_mb1_comp in LOCKSTEP_SYSTEM_TOP_routed.ncd.
.....
SUCCESS!
C:\Users\trevorh\Lock_Step_Micro\LX150T_Design\lockstep_xapp\ivt\ncd>
```

X584\_108\_040512

*Figure 108: IVT in NCD Mode Command Line on Re-implemented Design*

**Figure 109** shows the floorplan, the SVG file output, and routed design for the re-implemented design.



X584\_109\_040512

Figure 109: Floorplan versus SVG File versus FPGA Editor of Re-implemented Design

The dual-lockstep MicroBlaze processor system is now created, IDF is applied, and design preservation is used. Figure 110 shows the completed steps for this reference design.

<input checked="" type="checkbox"/>	1. Generate Dual-MicroBlaze System in EDK Platform Studio.
<input checked="" type="checkbox"/>	2. Modify Dual-MicroBlaze System into Lockstep through EDK Platform Studio.
<input checked="" type="checkbox"/>	3. Build the Dual-Lockstep MicroBlaze System in EDK Platform Studio.
<input checked="" type="checkbox"/>	4. Perform a Quick Sanity Check of the Design.
<input checked="" type="checkbox"/>	5. Prepare Dual-Lockstep MicroBlaze Design Source for Isolated Design.
<input checked="" type="checkbox"/>	6. Synthesize and Floorplan Hierarchical Design.
<input checked="" type="checkbox"/>	7. Run IVT on the Design in UCF Mode.
<input checked="" type="checkbox"/>	8. Implement the Design.
<input checked="" type="checkbox"/>	9. Run IVT on the Design in NCD Mode.
<input checked="" type="checkbox"/>	10. Build Final Software in SDK.
<input checked="" type="checkbox"/>	11. Disconnect the MicroBlaze Processors and Debug Logic.
<input checked="" type="checkbox"/>	12. Re-verify the Re-implemented Design in IVT.

X584\_110\_041112

Figure 110: Reference Design Complete

## Conclusion

The Xilinx All-Programmable technologies and solutions enables generation of a cycle-for-cycle, highly available, highly reliable, repeatable dual-lockstep MicroBlaze processor system. A cycle-for-cycle dual-lockstep MicroBlaze processor system can be derived from a dual MicroBlaze processor system using the EDK tools. Using IDF, the system can be divided into isolated functions and regions within a single device, providing increased system reliability and availability. Design preservation can be applied to lock down isolated functions and guarantee repeatable results for future implementation iterations.

## References

This application note uses the following references:

1. [UG081](#), *MicroBlaze Processor Reference Guide Embedded Development Kit*
2. XAPP1145, *Developing Secure Designs with the Spartan-6 Family Using the Isolation Design Flow*
3. XAPP1104, *Implementation of a Fail-Safe Design in the Spartan-6 Family using ISE Design Suite 12.4*
4. Avnet Spartan-6 LX150T Development Board website:  
<http://www.em.avnet.com/en-us/design/drc/Pages/Xilinx-Spartan-6-LX150T-Development-Kit.aspx>
5. Avnet driver software for the USB to RS-232 converter website:  
[https://www.em.avnet.com/Support%20And%20Downloads/CP2102\\_USB\\_Drivers01.zip](https://www.em.avnet.com/Support%20And%20Downloads/CP2102_USB_Drivers01.zip)
6. [UG748](#), *Xilinx Hierarchical Design Methodology Guide*
7. [WP362](#), *Repeatable Results with Design Preservation*
8. [UG747](#), *Design Preservation Tutorial: PlanAhead Design Tool*

To find addition documentation, see the Xilinx website at <http://www.xilinx.com/support>.

## Revision History

The following table shows the revision history for this document.

Date	Version	Description of Revisions
07/10/2012	1.0	Initial Xilinx release.

## Notice of Disclaimer

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of the Limited Warranties which can be viewed at <http://www.xilinx.com/warranty.htm>; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in Critical Applications: <http://www.xilinx.com/warranty.htm#critapps>.

## **Automotive Applications Disclaimer**

XILINX PRODUCTS ARE NOT DESIGNED OR INTENDED TO BE FAIL-SAFE, OR FOR USE IN ANY APPLICATION REQUIRING FAIL-SAFE PERFORMANCE, SUCH AS APPLICATIONS RELATED TO: (I) THE DEPLOYMENT OF AIRBAGS, (II) CONTROL OF A VEHICLE, UNLESS THERE IS A FAIL-SAFE OR REDUNDANCY FEATURE (WHICH DOES NOT INCLUDE USE OF SOFTWARE IN THE XILINX DEVICE TO IMPLEMENT THE REDUNDANCY) AND A WARNING SIGNAL UPON FAILURE TO THE OPERATOR, OR (III) USES THAT COULD LEAD TO DEATH OR PERSONAL INJURY. CUSTOMER ASSUMES THE SOLE RISK AND LIABILITY OF ANY USE OF XILINX PRODUCTS IN SUCH APPLICATIONS.