# Revision History

The following table shows the revision history for this document.

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Revision</th>
</tr>
</thead>
<tbody>
<tr>
<td>04/01/2015</td>
<td>2015.1</td>
<td>Consolidated information related to creating block designs within a local project structure, and outside of a project directory under Creating a Block Design in Chapter 2. Updated Figure 2-11, page 14 to show the Details window of the Vivado IP Catalog in Adding IP Modules to the Design Canvas in Chapter 2. Referenced the Vivado Design Suite User Guide: Creating and Packaging Custom IP [Ref 11] for detailed information under Packaging a Block Design in Chapter 2. Added information related to out-of-context synthesis and using the IP Cache in Generating Output Products in Chapter 4. Consolidated and updated information related to adding existing block designs, either from an existing project or from a standalone directory, to a new project under Adding Existing Block-Designs in Chapter 4. Referenced the UltraFast Design Methodology Guide for the Vivado Design Suite (UG949) [Ref 9] for information related to Revision Control for Block Designs in Chapter 4. Referenced the Vivado Design Suite User Guide: Programming and Debugging (UG908) [Ref 7] for information related to Chapter 6, Using the ILA to Debug IP Integrator Designs. Added an example Tcl script to Chapter 8, Using IP Integrator in Non-Project Mode. Updated information in Chapter 10, Using the Platform Board Flow in IP Integrator. Updated information in Chapter 11, Using Third-Party Synthesis Tools in IP Integrator.</td>
</tr>
</tbody>
</table>
# Table of Contents

Revision History ................................................................. 2

## Chapter 1: Getting Started with Vivado IP Integrator

Introduction ................................................................. 5

## Chapter 2: Creating a Block Design

Introduction ................................................................. 6
Creating a Project .......................................................... 6
Designing with IP Integrator .............................................. 10
Making Connections .......................................................... 18
Re-arranging the Design Canvas ......................................... 38
Running Design Rule Checks .............................................. 43
Packaging a Block Design .................................................. 43

## Chapter 3: Creating a Memory-Map

Introduction ................................................................. 44
Using the Address Editor ..................................................... 45

## Chapter 4: Working with Block Designs

Overview ................................................................. 58
Generating Output Products ................................. 58
Integrating the Block Design into a Top-Level Design .................................................. 62
Adding Existing Block-Designs .............................................. 64
Revision Control for Block Designs ........................................ 67
Exporting a Hardware Definition to SDK ................................ 68
Adding and Associating an ELF File to an Embedded Design ........................................ 70

## Chapter 5: Propagating Parameters in IP Integrator

Introduction ................................................................. 79
Using Bus Interfaces .......................................................... 80
How Parameter Propagation Works ..................................... 85
Parameters in the Customization GUI ..................................... 86
Parameter Mismatch Example .............................................. 88
Chapter 6: Using the ILA to Debug IP Integrator Designs
  Overview ........................................................................................................... 90
  Using the HDL Instantiation Flow in IP Integrator ........................................ 91
  Using the Netlist Insertion Flow .................................................................... 97

Chapter 7: Using Tcl Scripts to Create Block Designs within Projects
  Overview ........................................................................................................... 108

Chapter 8: Using IP Integrator in Non-Project Mode
  Overview ........................................................................................................... 112
  Creating a Flow in Non-Project Mode .............................................................. 112

Chapter 9: Updating Designs for a New Release
  Overview ........................................................................................................... 115
  Upgrading a Block Design in Project Mode ...................................................... 115
  Upgrading a Block Design in Non-Project Mode ............................................. 123

Chapter 10: Using the Platform Board Flow in IP Integrator
  Overview ........................................................................................................... 124
  Select a Target Board ...................................................................................... 124
  Create a Block Design to use the Board Flow ............................................... 126
  Complete Connections in the Block Design .................................................. 131

Chapter 11: Using Third-Party Synthesis Tools in IP Integrator
  Overview ........................................................................................................... 133
  Setting the Block Design as Out-of-Context Module ..................................... 133
  Creating an HDL or EDIF Netlist in Synplify ............................................... 135
  Creating a Post-Synthesis Project in Vivado ................................................... 136
  Adding Top-Level Constraints ...................................................................... 138
  Adding an ELF File ......................................................................................... 138
  Implementing the Design ................................................................................ 141

Appendix A: Additional Resources and Legal Notices
  Xilinx Resources ............................................................................................. 144
  References ........................................................................................................ 144
  Training Resources ......................................................................................... 145
  Please Read: Important Legal Notices .......................................................... 146
Chapter 1

Getting Started with Vivado IP Integrator

Introduction

As FPGAs become larger and more complex, and as design schedules become shorter, use of third-party IP and design reuse is becoming mandatory. Xilinx® recognizes the challenges designers face, and to aid designers with design and reuse issues, has created a powerful feature within the Vivado® Design Suite called the Vivado IP integrator.

The Vivado IP integrator feature lets you create complex system designs by instantiating and interconnecting IP from the Vivado IP catalog on a design canvas. You can create designs interactively through the IP integrator canvas GUI or programmatically through a Tcl programming interface. Designs are typically constructed at the interface level (for enhanced productivity) but may also be manipulated at the port level (for precision design manipulation).

An interface is a grouping of signals that share a common function. An AXI4-Lite master, for example, contains a large number of individual signals plus multiple buses, which are all required to make a connection. If each signal or bus is visible individually on an IP symbol, the symbol will be visually very complex. By grouping these signals and buses into an interface, the following advantages can be realized. First, a single connection in IP integrator (or Tcl command) creates a master to slave connection. Next, the graphical representation of this connection is a simple, single connection. Finally, Design Rule Checks (DRCs) that are aware of the specific interface can be run to assure that all the required signals are connected properly.

A key strength of IP integrator is that it provides a Tcl extension mechanism for its automation services so that system design tasks such as parameter propagation, can be optimized per-IP or application domain. Additionally, IP integrator implements dynamic, runtime DRCs to ensure, for example, that connections between the IP in an IP integrator design are compatible and that the IP themselves are properly configured.
Creating a Block Design

Introduction

This chapter describes the basic features and functionality of Vivado IP integrator.

Creating a Project

While entire designs can be created using IP integrator, the typical design will consist of HDL, IP, and IP integrator block designs. This section is an introduction to creating a new IP integrator-based design.

As shown in the figure below, you start by clicking on Create New Project in the Vivado® IDE graphical user interface (GUI) to create a new project. The Vivado Design Suite supports many different types of design projects. Refer to this link in the Vivado Design Suite User Guide: System-Level Design Entry (UG895) [Ref 3] for more information.

![Create New Project Wizard](image-url)
To add or create a block design in a project, you must create an RTL project, or open an Example Project as shown in Figure 2-2. You can add VHDL or Verilog design files, IP from the Vivado IP catalog, and other types of design source files to the project using the New Project wizard.

![Create New Project Wizard](image)

**Figure 2-2: Create New Project Wizard**

After adding design sources, existing IP, and design constraints, you can also select the default Xilinx device or platform board to target for the project, as shown in Figure 2-3, page 8. For more information, see Chapter 10, Using the Platform Board Flow in IP Integrator.

**IMPORTANT:** The Vivado tools support multiple versions of Xilinx target boards, so carefully select your target hardware.

The Tcl equivalent commands for creating a project are:

```tcl
create_project <project_name> <dir_name>/xx -part xc7k325tffg900-2
set_property BOARD_PART xilinx.com:kc705:part0:1.2 [current_project]
set_property TARGET_LANGUAGE vhdl [current_project]
```

**Note:** When displaying the Tcl commands in this document, the <> characters are used to designate variables that are specific to your design. The <> symbols should not be included in the command string.

See the *Vivado Design Suite Tcl Command Reference Guide* (UG835) [Ref 1] for information on specific Tcl commands.
Chapter 2: Creating a Block Design

Creating a Block Design

You can create a block design inside the current project directory, or outside of the project directory structure. A common use case for creating the block design outside of a project is to use the block design in non-project mode, or to use it in multiple projects, or to use it in a team-based design flow.

You create a new block design in the Flow Navigator by clicking the Create Block Design under the IP Integrator heading, as shown in the following figure.

1. Select Flow Navigator > IP Integrator > Create Block Design.
Chapter 2: Creating a Block Design

The Create Block Design dialog box opens, as shown below.

2. In the Create Block Design dialog box, specify the **Design name** and the **Directory**.

![Create Block Design Dialog Box](image)

**Figure 2-5:** Create Block Design Dialog Box

The default value for the Directory field is `<Local to Project>`. You can override this default value by clicking the **Directory** field and selecting **Choose Location**.

3. Choose the location in which to create the block design, and click **OK**.

The entire block design directory is created local to the current project, or at the specified location with its own directory structure.

The equivalent Tcl command is:

```
create_bd_design <your_design_name>
```
Designing with IP Integrator

Once the block design is created, the Vivado IP integrator provides a design canvas that you can use to construct your design. This canvas can be re-sized in the Vivado IDE GUI. You can double-click the design canvas tab at the upper-left corner of the diagram to increase the size of the diagram. When you double-click the tab again, the view returns to the default layout. You can even move the design canvas to a separate monitor by clicking on the Float Window button in the upper-right corner of the diagram, and moving the window as needed.

Changing Layers

To display the layers, click the top-left icon in the Diagram window, as shown by the red circle in the following figure. You can select the Attributes, Nets and Interface connections that you want to view or hide by checking or un-checking the boxes against these.

Attributes

Several attributes of the block design can be displayed or hidden by checking or un-checking the options. The following attributes can be modified.

- Pin tie offs - Pins that have a tie-off value specified, for e.g. ‘0’ or ‘1’ can be displayed by checking the Pin tie offs option.
Chapter 2: Creating a Block Design

Figure 2-7: Viewing/Hiding Pin tie-offs on the pins of IP Symbols

- Pins without parameter propagation - Show or hide the pins that do not propagate parameters.
- Mark Debug - Show or hide pins that have been marked for debug. Nets marked for debug have a bug symbol placed on them.

Figure 2-8: Viewing/Hiding nets marked for debug

- Display pins of hidden nets and interfaces - This option works in conjunction with the Nets or Interface Connections option. If a net has been hidden by un-checking the appropriate net, then the pins that are connected by the net also are hidden. This option displays the pins in question, even though the nets may be hidden.
Chapter 2: Creating a Block Design

Nets

Several types of nets such as clock nets, reset nets, data nets or simply other unclassified type of nets can be hidden or shown on the block design canvas by selecting the appropriate check box.

Interface Connection

Interface connections can also be shown or hidden by selecting the options under this category.

Colors in a block design

You can change the background color of the diagram canvas and other objects from the default color. As shown in the following figure, you can click the Block Design Options > Colors button in the upper-left corner of the diagram to change the color.

![Figure 2-9: Changing the IP Integrator Background Color](image)

Notice that you can control the colors of almost every object displayed in an IP integrator diagram. For example, changing the Background color to 240,240,240 as shown above
makes the background light gray. To hide the **Block Design Options**, either click the close button in the upper-right corner, or click the **Block Design Options** button again.

### Using Mouse Strokes and the Left-Button Panel

- A northwest stroke (lower-right to upper-left) is **Zoom Fit**
- A southwest stroke (upper-right to lower-left) is **Zoom In**
- A northeast stroke (lower-left to upper-right) is **Zoom Out**
- A southeast stroke (lower-right to upper-left) is **Zoom Area**

The toolbar menu on the left side of the design canvas allow the following commands to be invoked:

![Zoom In](image)

---

**Figure 2-10: IP Integrator Action Buttons**

### Adding IP Modules to the Design Canvas

You can add IP modules to a diagram in the following ways:

1. Right-click in the diagram and select **Add IP**.

   A searchable IP Catalog opens.
2. By typing in the first few letters of the IP name in the Search filter at the top of the catalog, only IP modules matching the search string are displayed.

3. To add a single IP, you can either click on the IP name and press the Enter key on your keyboard, or double click on the IP name.

4. To add multiple IP to the Block Design, you can highlight the additional desired IP (Ctrl+Click) and press the Enter key.

5. You can also add IP to the block diagram by opening the IP Catalog from the Flow Navigator.
a. Click on IP Catalog under Project Manager in Flow Navigator.

b. In the Search field of the IP Catalog type the name of the IP.

c. Double-click on the IP to place on the block design.

d. In the Add IP dialog box select **Add IP to Block Design**.

6. Alternatively, float the IP Catalog by clicking on the Float icon at the upper right corner of the catalog window. Then drag and drop the IP of your choice from the IP Catalog in the block design canvas.

**TIP:** Different fields associated with an IP such as Name, Version, Status, License, Vendor VLNV etc. can be enabled by right-clicking in the displayed Header column of the IP Catalog and enabling and disabling the appropriate fields.

7. Multiple IP can be added to the block design canvas at once by selecting multiple IP in the IP catalog and using one of the methods described above.

**Hierarchical IP in IP integrator**

Some IP in the IP Catalog are hierarchical, and offer a child block design inside the top-level block design to display the logical configuration of the parent. These hierarchical blocks let you see the contents of the block, but do not let you edit the hierarchy. Changes cannot be made to the child block design. Changes can only be made in the Re-customize IP dialog box of the IP.

For example, the 10G Ethernet Subsystem and AXI 1G/2.5G Ethernet Subsystem are Hierarchical IP. You would instantiate these IP just as any other IP by searching and selecting the IP from the IP Catalog.
When the IP has been instantiated in the block design, double-click the IP to re-customize it. This opens the Re-customize IP dialog box where you can configure the IP parameters, as shown in the following figure.

![Figure 2-14: Adding Hierarchical IP to the Block Design](image)

You can see the child block design inside the AXI Ethernet subsystem IP by right-clicking and selecting **View Block Design** option, as shown in **Figure 2-16, page 17**.
This opens a block design window showing the child-level block design, as shown in the following figure. Again, the block design cannot be edited directly.

You can also view the block design by clicking on the View Block Design icon at the top left corner of the IP symbol.
TIP: If you re-customize the IP while the child-level block design is open, it will be closed.

Making Connections

When you create a design in IP integrator, you add blocks to the diagram, configure the blocks as needed, make interface-level or simple-net connections, and add interface or simple ports. Making connections in IP integrator is simple. As you move the cursor near an interface or pin connector on an IP block, the cursor changes into a pencil. You can then click on an interface or pin connector on an IP block, hold down the left-mouse button, and then draw the connection to the destination block.

As shown on the SLMB interface pin in the following figure, an interface-level connection is indicated by the more prominent connection box on a symbol.
Chapter 2: Creating a Block Design

Clicking the + symbol on a block expands the interface and displays the associated signals and buses.

A signal or bus-level connection is shown as a narrow connection line on a symbol. Buses are treated identically to individual signals for connection purposes. As shown in the following figure, when you are making a connection, a green check box appears near any possible destination connections, highlighting the potential connections.

As shown in the following figure, when signals are grouped as an interface, you can expand the interface to make connections to individual signal or bus pins.

Figure 2-19: Connection Box on a Symbol

Figure 2-20: Signal or Bus Connection on a Symbol
You can connect signals and interfaces to external I/O ports as follows:

- Make External
- Create Port
- Create Interface Port

These options are described in the following sections.

**Making Ports External**

1. To connect signals or interfaces to external ports on a diagram, first select a pin, bus, or interface connection, as shown in the following figure.

2. Right-click and select **Make External**.

   You can also use **Ctrl+Click** to select multiple pins and invoke the **Make External** command for all pins at one time.
This command ties a pin on an IP to an I/O port on the block design. IP integrator connects the port on the IP to an external I/O.

**Creating Ports**

1. To use the create port option, right-click and select **Create Port**, as shown in the following figure. This feature is used for connecting individual signals, such as a clock, reset, and uart_txd.

   **Create Port** gives you more control in specifying the input and output, the bit-width and the type (such as clk, reset, and data).

   When you specify a clock, you can also specify the input frequency.
Chapter 2: Creating a Block Design

The Create Port dialog box opens, as shown in the following figure:

![Create Port Dialog Box](image)

*Figure 2-23: Create Port Option*

The Create Port dialog box opens, as shown in the following figure:

![Create Port Dialog Box](image)

*Figure 2-24: Create Port Dialog Box*
2. Specify the port name, the direction such as input, output or bidirectional, and the type (such as clock, reset, interrupt, data, clock enable or custom type).

You can also create a bit-vector by checking the **Create Vector** field and then selecting the appropriate bit-width.

**Creating Interface Ports**

1. To use the create interface port option, right-click and select **Create Interface Port**, as shown in the following figure.

![Create Interface Port Option](image)

**Figure 2-25: Create Interface Port Option**

This command creates ports on the interface pins which are groupings of signals that share a common function. For example, the **LMB_M** and **LMB_SI_0** are interface pins in the figure above. The **Create Interface Port** command gives more control in terms of specifying the interface type and the mode (master/slave).

2. In the Create Interface Port dialog box, shown in the following figure, specify the interface name, the vendor, library, name and version (VLNV) field, and the mode field such as **MASTER** or **SLAVE**.
3. Double-click external ports to see their properties and modify them.

In the figure below, the port shown is a clock input source, so you can specify different properties such as frequency, phase, clock domain, any bus interface, the associated clock enable, associated reset and associated asynchronous reset (frequency).

![Create Interface Port Dialog Box](image1.png)

*Figure 2-26: Create Interface Port Dialog Box*

On an AXI interface, double-clicking the port shows the following configuration dialog box.

![Customize Clock Port Properties Dialog Box](image2.png)

*Figure 2-27: Customize Clock Port Properties Dialog Box*
Handling Interrupts

Interrupt handling in the Vivado Design Suite IP integrator tool depends on the processor being used. For a Zynq®-7000 processor, the Generic Interrupt Controller block within the Zynq-7000 processor handles the interrupt.

For a MicroBlaze™ processor, the AXI Interrupt Controller IP must be used to manage interrupt. Regardless of the processor used in the design, a Concat IP consolidates and drives the interrupt pins.
The inputs of the Concat IP are driven by different interrupt sources. Accordingly, the Concat IP must be configured to support the appropriate number of input ports. The **Number of Ports** field must be set to the number of interrupt sources in the design as shown in the following figure.

![Concat IP Driving Interrupt Input to AXI Interrupt Controller](image)

**Figure 2-29: Concat IP Driving Interrupt Input to AXI Interrupt Controller**

![Concat Re-customize IP Dialog Box](image)

**Figure 2-30: Concat Re-customize IP Dialog Box**
**TIP:** The width of the output (\(d_{out}\)) is set automatically during parameter propagation.

You can configure several of the parameters for the AXI Interrupt Controller. The following figure shows the parameters available from the Basic tab of the AXI Interrupt Controller, of which several are configurable:

- The **Number of Peripheral Interrupts** cannot be set by the user. This is automatically set during parameter propagation. This value is determined by the number of interrupt sources that are driving the inputs of the Concat IP.
- The **Fast Interrupt Mode** can be set by the user if low latency interrupt is desired.
- The **Peripheral Interrupts Type** is set to **Auto**, which can be overridden by the user by toggling the **Auto** setting to **Manual**. In manual mode, users can specify the custom values in these fields.

*Figure 2-31: AXI Interrupt Controller Basic Tab Parameters*
Chapter 2: Creating a Block Design

- The **Processor Interrupt Type** field offers two choices:
  - the **Interrupt Type**, and depending on the **Interrupt Type** setting:
  - the **Level Type** of **Edge Type**.

  For example, if the **Interrupt Type** is **Edge Interrupt**, the other choice is **Edge Type**. If the **Interrupt Type** is **Level Interrupt**, the other choice is **Level Type**.

  Users can select if the interrupt source is either **Edge-triggered** or **Level-triggered**. Accordingly, then can also select whether the interrupt is rising or falling edge and in case of Level triggered interrupt the interrupt is active-High or active-Low.

  In IP integrator, this value is normally automatically determined from the connected interrupt signals, but can be set manually.

---

**CAUTION!** You must ensure that the interrupt inputs of the Concat IP are all of the same type (for example Edge/Level triggered and Rising/Falling or active-High/active-Low respectively) when consolidating into the Interrupt Controller.

---

The following figure shows parameters on the Advanced tab of the AXI Interrupt Controller. See the *LogiCORE IP AXI Interrupt Controller (PG099)* [Ref 11] for details of these parameters.
One option to notice is the **Asynchronous Clocks** option. The AXI Interrupt Controller determines whether the interrupt sources in a design are from the same clock domain or different clock domains. In the case of interrupts being driven from different clock domains, the **Enable Asynchronous Clock operation** is enabled automatically. In this case, cascading synchronizing registers are added to the interrupt sources.

**TIP:** You can also override the automatic behavior by toggling the **Auto** button to **Manual** and setting this option manually.

---

**Using the Designer Assistance Feature**

IP integrator offers a feature called Designer Assistance, which includes **Block Automation** and **Connection Automation**, to assist you in putting together a basic microprocessor system by making internal connections between different blocks and making connections to external interfaces. The Block Automation Feature is provided when an embedded processor such as the Zynq® Processing System 7 or MicroBlaze™ processor, or some other hierarchical IP such as an Ethernet is instantiated in the IP integrator block design.
Click the **Run Block Automation** link in the banner of the design canvas, as shown in the following figure, for assistance in putting together a simple MicroBlaze system.

![Run Block Automation Feature](image1.png)

*Figure 2-33: Run Block Automation Feature*

The **Run Block Automation** dialog box opens, as shown in *Figure 2-34, page 30*, and lets you provide input about basic features that the microprocessor system needs.

![Run Block Automation Dialog Box](image2.png)

*Figure 2-34: Run Block Automation Dialog Box*

After you specify the necessary options, the Block Automation feature automatically creates a basic system as shown in the following figure.
Chapter 2: Creating a Block Design

The MicroBlaze System shown in Figure 2-35, page 31 consists of a MicroBlaze Debug Module, which is a hierarchical block called the `microblaze_1_local_memory` that has the Local Memory Bus, the Local Memory Bus Controller and the Block Memory Generator, a Clocking Wizard, an AXI Interconnect and an AXI Interrupt Controller.

Because the design is not connected to any external I/O at this point, IP integrator offers the **Connection Automation** feature as shown in the light green banner of the design canvas in the preceding figure. When you click **Run Connection Automation**, IP integrator provides assistance in hooking interfaces and/or ports to external I/O ports.

The Run Connection Automation dialog box, as shown in the following figure, lists ports and interfaces that the Connection Automation feature supports, along with a brief description of the available automation, and available options for each automation.

---

**Figure 2-35:** MicroBlaze System Created by Block Automation

**Figure 2-36:** Ports and Interfaces that can use Connection Automation

For Xilinx Target Reference Platforms or evaluation boards, IP integrator has knowledge of the FPGA pins that are used on the target boards. Based on that information, the IP integrator connection automation feature can assist you in tying the ports in the design to external ports on the board. IP integrator then creates the appropriate physical constraints and other I/O constraints required for the I/O port in question.

In the MicroBlaze system design shown in Figure 2-35, page 31, the following connections need to be made:

- Processor System Reset IP needs to be connected to an external reset port.
- Clocking Wizard needs to be connected to an external clock source as well as an external reset.

By selecting the appropriate options, as shown in the following figure, you can tie the clock and the reset ports to the appropriate sources on the target board.

![Run Connection Automation](image)

**Figure 2-37: Run Connection Automation**

You can select the reset pin that already exists on the KC705 target board in this case, or you can specify a custom reset pin for your design. After the reset is specified, the reset pin is tied to the `ext_reset_in` pin of the `Proc_Sys_Rst IP`.

![Connecting the Reset Pin to the Board Reset Pin](image)

**Figure 2-38: Connecting the Reset Pin to the Board Reset Pin**

The Designer Assistance feature is constantly monitoring your design development in IP integrator.
For example, assume that you instantiate the AXI_GPIO IP into the design. The Run Connection Automation link reappears in the banner on top of the design canvas. You can then click Run Connection Automation and the S_AXI port of the newly added AXI GPIO can be connected to the MicroBlaze processor using the AXI Interconnect. Likewise the GPIO interface can be tied to one of the several interfaces present on the target board.

![Run Connection Automation](image)

Figure 2-39: Using Connection Automation to Show Potential Connections

The connection option are: GPIO interface port can be connected to either the Dip Switches that are 4-bits, or to the LCD that are 7-bits, LEDs that are 8-bits, 5-bits of Push Buttons, the Rotary Switch on the board, or can be connected to a Custom interface. Selecting any one of the choices connects the GPIO port to the existing connections on the board.

Selecting the S_AXI interface for automation, as shown in the following figure, informs you that the slave AXI port of the GPIO can be connected to the MicroBlaze master. If there are multiple masters in the design, then you have a choice to select between different masters. You can also specify the clock connection for the slave interface such as S_AXI interface of the GPIO.

![Run Connection Automation](image)

Figure 2-40: Connecting the Slave Interface S_AXI to the MicroBlaze Master
When you click the OK in the Run Connection Automation dialog box, the connections are made and highlighted as shown in the following figure.

Enhanced Designer Assistance is available for advanced users who want to connect an AXI4-Stream interface to a memory-mapped interface. In this case IP integrator instantiates the necessary sub-components and makes appropriate connections between them to implement this functionality. See this link in the Vivado Design Suite User Guide: Embedded Hardware Design (UG898) [Ref 5] for more information on this feature.

Using the Signals View to Make Connections

IP integrator provides for an easy way to make connections to clocks and resets using the Signals view. After a block design is open, the Signals view displays, as shown in Figure 2-42, page 35, with two tabs listing the Clocks and Reset signals present in the design.
Selecting the appropriate tab shows all the clocks or resets in the design.

Clocks are listed in the Clocks tab based on the clock domain name. In the figure above, the clock domain is `design_1_clk_wiz_1_0_clk_out1` and the output clock is called `clk_out1` with a frequency of 100 MHz, and is driving several clock inputs of different IP.

When you select a clock from the Unconnected Clocks folder, IP integrator highlights the respective clock port in the block design. Right-clicking the selected clock presents you with several options.

In the case shown in Figure 2-33, page 30, the Designer Assistance is in the form of the Run Connection Automation command which can be used to connect the `CLK_IN1_D` input interface of the Clocking Wizard to the clock pins on the board.

You can also select the Make Connection option, and connect the input to an existing clock source in the design. Finally, you can tie the pin to an external port by selecting the Make External option.

Other options for switching the context to the diagram and running design validation are also available.
When you select Make Connection, a dialog box opens, as shown in the following figure, if a valid connection can be made.

Selecting the appropriate clock source makes the connection between the clock source and the port or pin.

Connections can similarly be made from the Resets tab. Using the Clocks and Resets tab of the Signals view provides you with a visual way to manage and connect clocks in the design.
Using Make Connections to Connect Ports and Pins

Connections to unconnected ports or pins can be made by selecting a port or pin and then selecting Make Connection from the right-click menu, as shown in the following figure.

![Make Connection Option](image)

*Figure 2-45: Make Connection Option*

If a valid connection to the selected pin exists, the Make Connection dialog box opens that shows all the possible sources to which that net can be connected. From this dialog box you can select the appropriate source to drive the port/pin can be selected.

Making Connections with Start Connection Mode

You can make multiple connections at once by clicking on a pin and when the pencil shows up dragging and releasing the mouse, as shown in the following figure.

![Start Connection Mode](image)

*Figure 2-46: Start Connection Mode*

After the connection is made to the s_axi_aclk pin of the AXI BRAM Controller in the figure above, the Start Connection Mode will offer to connect the pin to the s_axi_aclk pin of AXI IIC.

In this way connections from the same source pin can be made to multiple different destinations at once.
Interfacing with AXI IP Outside of the Block Design

There are situations when an AXI master is outside of the block design. These external masters are typically connected to the block design using an AXI Interconnect. Once the ports on the AXI interconnect are made “external”, the address editor is available and memory mapping can be done in these cases.

As an example, consider the block design shown in the following figure.

Re-arranging the Design Canvas

IP blocks placed on the canvas can be re-arranged to get a better layout of the block design, and connections between blocks. To arrange a completed diagram or a diagram in progress, you can click the **Regenerate Layout** button.

You can also move blocks manually by clicking on a block, holding the left-mouse button down, and moving the block with the mouse, or with the arrow keys. The diagram only allows specific column locations, indicated by the dark gray vertical bars that appear when moving a block. A grid appears on the diagram when moving blocks, which assists you in making better block and pin alignments.
It is also possible to manually place the blocks where desired and then click **Optimize Routing**. This preserves the placement of the blocks (unlike the Regenerate Layout function) and only modifies the routing to the various blocks. In other words, the optimize routing function keeps the location of different blocks intact and only modifies the nets connecting different blocks.

**Showing Interface Level Connectivity Only**

To see only the connectivity between interfaces present on the block design select the **Show interface connections only** button from the block design toolbar. This shows only the interface level connections, and hides all the other connections on the block design.

![Interface Connections Only](image.png)

*Figure 2-48: Interface Connections Only*

Clicking the **Show interface connections only** button again restores all the connections in the block design.

**Creating Hierarchies**

As shown in the following figure, you can create a hierarchical block in a diagram by using **Ctrl+Click** to select the desired IP blocks, right-click and select **Create Hierarchy**. IP integrator creates a new level of hierarchy containing the selected blocks.
Creating multiple levels of hierarchy is supported. You can also create an empty level of hierarchy, and later drag existing IP blocks into that empty hierarchical block.

When you click the + sign in the upper-left corner of an expandable block you can expand the hierarchy. You can traverse levels of hierarchy in a diagram using the Explorer type path information displayed in the upper-left corner of the IP integrator diagram.

Clicking Create Hierarchy opens the Create Hierarchy dialog box, as shown in the following figure, where you can specify the name of the new hierarchy.

This action groups the selected IP blocks under one block, as shown below.

- Click the + sign of the hierarchy to view the components underneath.
- Click the – sign on the expanded hierarchy to collapse it back to the grouped form.
Adding Pins and Interfaces to Hierarchies

As mentioned above, you can create an empty hierarchy and you can define the pin interface on that hierarchy before moving blocks of IP under the hierarchy.

Right-click on the IP integrator canvas, with no IP blocks selected, and select **Create Hierarchy**. In the Create Hierarchy dialog box, you specify the name of the hierarchy. Once the empty hierarchy has been created, the block design should look like the following figure.

```
create_bd_pin -dir I -type rst /hier_0/rst
```

In the above command, an input pin named reset of rst type was added to the hierarchy. You can add other pins using similar commands. Likewise, you can add a clock pin to the hierarchy using the following Tcl command:

```
create_bd_pin -dir I -type clk /hier_0/clock
```

You can also add interfaces to a hierarchy by using the following Tcl commands. First set the block design instance to the appropriate hierarchy where the interface is to be added, using the following command:

```
current_bd_instance /hier_0
```

Next, create the interface using command as specified below:

```
create_bd_intf_pin -mode Master -vlnv xilinx.com:interface:gpio_rtl:1.0 gpio
```
It is assumed that the right type of interface has been created prior to using the above command. After executing the commands shown above the hierarchy should look as shown in the following figure.

![Hierarchy Diagram](image)

**Figure 2-53: Create Pins**

After you have created the appropriate pin interfaces, different blocks can be dropped within this hierarchical block and pin connections from those IP to the external pin interface can be made.

![Connected IP to Hierarchical Pin Interface](image)

**Figure 2-54: Connected IP to Hierarchical Pin Interface**

**Cutting and Pasting**

You can use **Ctrl-C** and **Ctrl-V** to copy and paste blocks in a diagram. This lets you quickly copy IP blocks that have been customized, or copy IP into new hierarchical blocks.
Running Design Rule Checks

IP integrator runs basic design rule checks in real time as the design is being assembled. However, there is a potential for something to go wrong during design creation. As an example, the frequency on a clock pin may not be set right. As shown in the following figure, you can run a comprehensive design check on the design by clicking Validate Design.

You can also click the Validate Design button in the toolbar on the IP integrator canvas. If the design is free of Warnings and/or Errors, a confirmation dialog box displays, as shown in the following figure.

Packaging a Block Design

When you have created an IP integrator block design, implemented it, validated it, and tested it on target hardware, and you are satisfied with the functionality of the block design, you can package the block design to create an IP that can be reused in another design.

For more information on packaging a block design for use in the Vivado IP Catalog, see this link in the Vivado Design Suite User Guide: Creating and Packaging Custom IP (UG1118) [Ref 11].
Creating a Memory-Map

Introduction

Master interfaces reference an assigned memory range container called *address spaces*, or *bd_address_space* objects. Slave interfaces reference a requested memory range container, called a memory map.

By convention:

- Memory-maps are named after the slave interface pins that reference them, for example the *S_AXI* interface references the *S_AXI* memory-map, though that is not required.
- Address space names are related to its usage; for example, the MicroBlaze™ processor has a *Data* address space and an *Instruction* address space.

The memory-map for each slave interface pin contains slave segments, or *bd_address_seg* objects. These address segments correspond to the address decode window for that slave.

A typical AXI4-Lite slave has only one address segment, representing a range of memory. However, some slaves, like a bridge, have multiple address segments, or a range of addresses for each address decode window.

When a slave segment is mapped to the master address space, a master *bd_address_seg* object is created, mapping the address segments of the slave to the master. The Vivado® IP integrator can automatically assign addresses for all slaves in the design. However, you can also manually assign the addresses using the Address Editor.

**TIP:** *The Address Editor tab only appears if the diagram contains an IP block that functions as a bus master (such as the MicroBlaze processor) or if an external bus master (outside of IP integrator) is present.*

Click the *Address Editor* tab above the design canvas. In the Address Editor, you can see the address segments of the slaves, and can map them to address spaces in the masters.

If you generate the RTL from an IP integrator block design without first generating addresses, the IP integrator prompts you to automatically assign addresses at that point.
You can also set addresses manually by entering values in the Offset Address and Range columns.

A master such as a processor communicates with peripheral devices through device registers. Each of the peripheral devices is allocated a block of memory within a master’s overall memory space. IP Integrator follows the industry standard IP-XACT data format for capturing memory requirements and capabilities of endpoint masters and slaves.

IP integrator provides an Address Editor to allocate these memory ranges to the master/slave interfaces of different peripherals. Master and slave interfaces each reference specific memory objects.

### Using the Address Editor

The Address Editor in the Vivado IP Integrator tool is used to allocate memory ranges to peripherals from the perspective of a master interface. The Address Editor tab becomes available when a master with an address space, such as a MicroBlaze processor or a Zynq-7000 processor is instantiated in the Diagram canvas.

![Address Editor Tab](image)

As the peripherals are instantiated and connected to the processor in the block design canvas using connection automation, the IP integrator automatically enters a corresponding memory assignment that peripheral in the Address Editor, as shown in Figure 3-2, page 46.
Chapter 3: Creating a Memory-Map

The columns of the Address Editor are as follows:

- **Cell**: Describes the master and the connected peripherals that can be addressed by that master.

  You can expand the tree by clicking the **Expand All** button, or by clicking the + sign to expand the selection.

  As shown in **Figure 3-2**, the instance name of the “master” is *microblaze_0* which addresses the Data and Instruction address spaces.

  The peripherals *microblaze_0_local_memory/dlmb_bram_if_cntlr* and *microblaze_0_local_memory/ilmb_bram_if_cntlr* are mapped into the Data and Instruction address spaces respectively, where the rest of the peripheral are only accessible by the Data address space.

- **Slave Interface**: Lists the name of the slave interface pin of the peripheral instance.

  As an example, the peripheral instances *microblaze_0_local_memory/dlmb_bram_if_cntlr* and *microblaze_0_local_memory/ilmb_bram_if_cntlr* each have an interface called **SLMB** as shown in **Figure 3-3**, page 47.
Chapter 3: Creating a Memory-Map

Figure 3-3: Interface Names listed in the Slave Interface column

- **Base Name**: Specifies the name of the slave segment.

By convention, the two names that IP integrator creates “on-the-fly” are Mem (memory) and Reg (register), as shown in Figure 3-5, which shows a design with multiple memory instantiations.

Figure 3-4: Multiple Memory Instantiations in a block design

These are given the base names in the address editor as shown in Figure 3-5, page 48.
Chapter 3: Creating a Memory-Map

Figure 3-5: Base Names given to multiple memory instantiations

- **Offset Address**: Describes the offset from the start of the address block.

As an example the addressable range for data and instruction address spaces are 4G each in Figure 3-5. The address space starts at 0x00000000 and ends at 0xFFFFFFFF. Within this address space the axi_uartlite_0 can be addressed to a range starting at offset 0x40600000, axi_gpio_0 can be addressed starting at offset 0x40000000 and so forth. This field is automatically populated as the slaves are mapped in the address space of the master. However, they can also be changed by the user.

The **Offset Address** and the **Range** fields are interdependent. The Offset Address field must be aligned with the Range field. Alignment implies that for a range of 2^N the least significant bits of the starting offset must have at least N 0's. As an example, if the range of a slave segment happens to be 64K or 2^16, the Offset Address must be in the form 0xXXXX0000. This means the lowest 16-bits need to be 0's. If this field is not set correctly, the error message, shown in Figure 3-6 displays.
In Figure 3-6, page 49, the user set an offset address with only 12 0’s in the least significant bits. Only a range of 4K or $2^{12}$ can be accommodated by the proposed offset address. Therefore, the message pops up informing the user that the address is misaligned. The message also tells the user where the next offset address can be set based on the current memory map.

- **Range**: Specifies the total range of the address block for a particular slave. This field is typically populated based on a parameter in the `component.xml` file for an IP. This can also be changed by clicking the drop-down menu and selecting the appropriate value for this field.

The Range and the Offset Address fields are interdependent, and as described in the **Offset Address** field previously, the $2^N$ Range field must be aligned with the N number of least significant bits of the Offset Address field. Setting the Range field such that it exceeds this number can cause the message, shown in the following figure, to display.
In Figure 3-7, the user tried to set the range to 128K or $2^{17}$, for an offset in the form $0xXXXX0000$, which is an offset with only 16 least significant bits (LSBs). To accommodate a range of 128K, the form of the address must be at least $0xXXX20000$, that is with at least 17-bits in the LSB of the starting offset.

- **High Address**: Adjusts itself based on the **Offset Address** and the **Range** value. This is the last addressable address in a particular assigned segment.

### Memory Mapping Using the Address Editor

While memory block assignments happens automatically as the slave interfaces are connected to master interfaces in the block design, the mapping can also be done manually in the Address Editor.

#### Auto Assigning Addresses

To map all the slave segments at once, right-click anywhere in the Address Editor and select **Auto Assign Address** or click the **Auto Assign** button on the block design tool bar as shown in the following figure.
This maps the slave segments as shown in Figure 3-9, page 51.

After the slave segments are mapped, several options are presented to the user for other actions using a right-click on a mapped address segment, as shown in the following figure.
Address Segment Properties

The Address Segment Properties shows the details of the address segment in the Address Segment Properties window, shown in the following figure.

![Address Segment Properties Window](image)

**Figure 3-11: Address Segment Properties Window**

The fields that this window shows are as follows:

- **Name**: Shows the name of the master segment that was automatically assigned. This name can be changed by the user if desired.
- **Full Name**: Is not editable, and shows the full name of the mapped slave segment.
- **Slave Interface**: Shows the slave interface of the peripheral that references the slave segment.

**Unmap Segment**

A mapped address segment can be unmapped by selecting **Unmap Segment** from the context menu. This address segment then shows up in the Unmapped Slaves folder as shown in **Figure 3-12, page 53**. The user can also right-click and select **Assign Address**
Designing IP Subsystems Using IP Integrator

Chapter 3: Creating a Memory-Map

(which maps only the selected address) or **Auto Assign Address** (which assigns all unmapped address segments in the design).

---

**Exclude Segment**

Excluding a segment makes a mapped segment un-addressable to the master in question. This is typically done when multiple masters are present in the design and the user wants to control which masters should access which slaves. See **Sparse Connectivity, page 54** for more information.

**Group by Master Interfaces**

Selecting the **Group by Master Interfaces** groups the master segments within an address space by the master interfaces through which they are accessed by the master.

For example, the MicroBlaze in the following block design has three different master interfaces accessing various address segments: DLMB, ILMB, and M_AXI_DP within the **Data Address Space**.
Selecting the Group by Master Interfaces re-arranges the different address segments in the table under the master interfaces tree.

**Sparse Connectivity**

In a multiple master design users might want to specify slaves that could potentially be accessed by all masters or by certain masters only. This feature of memory mapping in IP Integrator is called sparse connectivity.
Excluding Address Segment from a Memory-Mapped Master

The following figure is a block design with two masters.

![Block Design with Two Masters](image)

**Figure 3-15:** Multiple Master and Slave Example

In the following figure, there are two masters, *Master_1* and *Master_2*, accessing two slaves, *Slave_1* and *Slave_2* using the same interconnect.

![Address Editor with Multiple Master Memory-Map](image)

**Figure 3-16:** Address Editor with Multiple Master Memory-Map

For an example, assume that *Master_1* must access *Slave_2* only, and *Master_2* needs to access both *Slave_1* and *Slave_2*. To exclude *Slave_1* from the memory-map of *Master_1*, right-click M00_AXI and select **Exclude Segment**, as shown in **Figure 3-17**, page 56.
This action excludes the segment by showing the segment under the Excluded Address Segments folder as shown in the following figure.

**Figure 3-17: Exclude Segment Option**

You can exclude both mapped and unmapped slaves.

**IMPORTANT:** An excluded master segment still occupies a range within the address space despite it being inaccessible by the master.

If, after excluding a slave within a master address space, one attempts to manually place another slave to address that overlaps with the excluded slave, an error occurs during validation.
Including an address segment

An excluded segment can be added back to the Master by selecting Include Segment from the context menu as shown in the following figure.

![Diagram](image)

**Figure 3-19:** Including an Excluded Segment into Master Memory-Map

Common Addressing-Related Critical Warnings and Errors

[BD 41-971] “Segment <name of segment> mapped into <address space> at Offset [ Range ] overlaps with <name of segment> mapped at Offset [ Range].

This message is typically thrown during validation. Each peripheral must be mapped into a non-overlapping range of memory within an address space.

[BD 41-1356] Address block <name of slave segment> is not mapped into <name of address space>. Please use Address Editor to either map or exclude it.

This message is typically thrown during validation. If a slave is accessible to a master, it should be either mapped into the master’s address space or excluded from it.

[BD 41-1353] <name of slave segment> is mapped at disjoint segments in master <name of address space> at <memory range> and in master <name of address space> at <memory range>. It is illegal to have the same peripheral mapped to different addresses within the same network. Peripherals must either be mapped to the same offset in all masters, or into addresses that are apertures of each other or to contiguous addresses that can be combined into a single address with a range that is a power of 2.

This message is typically thrown during validation. Within a network defined as a set of masters accessing the same set of slaves connected through a set of interconnects, each slave must be mapped to the same address within every master address space, or apertures or subsets of the largest address range.
Chapter 4

Working with Block Designs

Overview

At this point, you should know how to create a block design, populate it with IP, make connections, assign memory address spaces, and validate the design. This chapter of the guide describes how to work with block designs, creating the necessary output files for synthesis and simulation, adding a block design to a top-level design, and exporting the block design to the software development toolkit (SDK) for embedded processor designs.

Generating Output Products

After the block design is complete and the design is validated, you must generate output products for synthesis and simulation, in order to integrate the block design into a top-level RTL design. The source files and the appropriate constraints for all the IP are generated and made available in the Vivado® Integrated Design Environment (IDE) Sources window. Output files are generated for a block design based upon the Target Language that you specified during project creation, or in the Project Settings dialog box. If the source files for a particular IP cannot be generated in the specified target language a message displays in the Tcl Console.

To generate output products, in the Vivado sources pane, right-click the block design and select Generate Output Products, as shown in the following figure.

Figure 4-1: Generate Output Products Option
Alternatively, click **Flow Navigator > IP Integrator Generate Block Design**, as shown in **Figure 4-2**.

![Generate Block Design Option](image)

Generating the output products generates the top-level netlist of the block design. The netlist is generated in either VHDL or Verilog based upon the Target Language settings in Project Settings.

### The Generate Output Products dialog box

Output products can be generated for three different modes:

- **Global**: Used for generating output products used in top down synthesis of the whole design. This is essentially disable out-of-context synthesis for the block design, and simply synthesizes it with the whole design.

- **Out of context per Block Design**: This lets you synthesize the complete block design separately from, or out of the context of the top-level design. This option is generally selected when third party synthesis is used. A design checkpoint (DCP) is generated for the block design when this option is selected.

- **Out of context per IP**: Generates the output product for each individual IP used in the block design. A DCP is created for every IP used in the block design. This option can significantly reduce synthesis run times because the IP cache can be used with this option to prevent Vivado synthesis from regenerating output products for specific IP if they have not been changed.
You can enable the IP Cache in the Project Setting dialog box. Click **IP** and then select the **IP Cache** tab, as seen in Figure 4-4. IP caching can be **Disabled**, or can be set as **Local** or **Remote**. For more information on setting the IP Cache, refer to this link in the Vivado Design Suite User Guide: Designing with IP (UG896) [Ref 4].
Global Synthesis

When this mode is chosen no design checkpoint is created. The entire block design is generated in the top down synthesis mode. You can see this in the Design Runs window.

![Design Runs window for Global Synthesis](image)

Out of context per Block Design

Typically used with third-party synthesis tools, this option creates a design checkpoint for the entire block design. As can be seen from the figure below, the Sources window shows that a Design Checkpoint file (DCP) was created for the block design. The Design Runs window shows the Out-of-Context Module runs for the top level block design.

![Design Runs window for Global Synthesis](image)

A module marked as out-of-context has a square symbol placed against it so it can be easily recognized from the Sources window.

If the block design is added as a synthesized netlist in other designs, this DCP file can be added to the project where the block design is instantiated.

**RECOMMENDED:** Add the block design to the project. Adding the block design brings in the DCP file as well.

Out of context per IP

This mode creates a DCP for every IP that is instantiated in the block design. When IP Caching is enabled, this mode can provide with significant run time improvements. When this mode is used only IP that has been modified, either because of re-customization or because of an impact from parameter propagation, is synthesized.
Generation of output products in out-of-context per IP takes a little longer to run. However, run time improvements can be seen in subsequent runs.

**Integrating the Block Design into a Top-Level Design**

An IP integrator block design can be integrated into a higher-level design or it can be the highest level in the design hierarchy. To integrate the IP integrator design into a higher-level design, instantiate the design in the top-level HDL file.

You can perform a higher-level instantiation of the block design by selecting the block design in the Vivado IDE Sources window and selecting **Create HDL Wrapper** (in the following figure). This generates a top-level HDL file for the IP integrator sub-system.

![Create HDL Wrapper Option](Image)

The Create HDL Wrapper dialog box opens, as shown in the following figure.
Chapter 4: Working with Block Designs

The Create HDL Wrapper options are as follows:

- **Copy generated wrapper to allow user edits.** When a block design is a subset of an overall design hierarchy, you need to have the option to manually edit the wrapper file so you can then instantiate other design components within the wrapper file.

  **IMPORTANT:** Ensure that this file is updated any time an I/O interface of the block design changes.

When you manually create a wrapper file it is placed in the `<project_name>.srcs/sources_1/imports/hdl` directory.

- **Let Vivado tools manage wrapper and auto-update.** If the block design is the only design component in the project or if edits to the wrapper file are not necessary, use this option.

  When the Vivado tools manage the wrapper file, that file is updated every time you generate output products. The wrapper file is located in the directory `<project_name>.srcs/sources_1/bd/<bd_name>/hdl`.

You are now ready to take the design through elaboration, synthesis, and implementation.

**Instantiating I/O Buffers**

When generating the wrapper, IP integrator looks for I/O interfaces that are made external in the design. If the tool finds I/O interfaces that are made external, it reviews the port maps of that interface. If three ports which match the pattern `<name>_I`, `<name>_O`, and `<name>_T` are found, then the tool instantiates an I/O buffer and connects the signals appropriately. If any of the three ports are not found, then an I/O buffer is not inserted.

Other conditions in which I/O buffers are not inserted include the following:

- If any of the _I, _O, and _T ports are manually connected by the user, either by making them external or by connecting it to another IP in the design.
• If the interface has the `BUFFER_TYPE` parameter set to `NONE`.

If you want to hand-instantiate I/O buffers in the block design, use the **Utility Buffer** IP that is available in the Vivado IP Catalog. This IP can be configured as different kinds of I/O buffers as shown in the following figure.

![Utility Buffer IP](image)

*Figure 4-10: Utility Buffer IP*

### Adding Existing Block-Designs

You can add an existing block design that was created outside of the current project, either from an existing project or from a remote directory location.

Assuming that a block design was created using a project-based flow, and all the directory structure including and within the block design folder is available, the block design can be added to a new Vivado project. The only limitation is that the target part or platform board for the current project must be the same as the original project in which the block design was created.
To add a remote block design:

1. Select **Flow Navigator > Add Sources**. Alternatively, you can right-click in the Sources window and select **Add Sources**.

2. In the Add Sources dialog box, select **Add Existing Block Design Sources**, as shown in the following figure, and click **Next**.

3. In the Add Existing Block Design Sources page, click **Add Files**.

4. In the Add Sources File window, navigate to the folder where the block design is located, select the `.bd` file, and click **OK**.

---

**IMPORTANT:** If the target devices of the projects are different, even within the same device family, the IP used in the block design will be locked, and the design must be re-generated. In that case the behavior of the new block design might not be the same as the original block design.
5. In the Add Sources dialog box you can enable or disable the **Copy sources into project** check box as needed for your current project.

![Add Source Files](image)

*Figure 4-12: Add block Design Source*

You can reference the block design from its original location, or copy it into the local project directory. Managing the block design remotely is the recommended practice.

![Add Existing Block Design Sources](image)

*Figure 4-13: Add Existing Block Design Sources*
when working with revision control systems. See Revision Control for Block Designs, page 67. However, if someone edits the remote block design, they may inadvertently change your referenced copy. To avoid that, you can enable the Copy sources into project check box, as seen in Figure 4-13, page 66, so that you can change the block design when needed, but remote users won’t be able to affect your design.

**TIP:** When adding a block design from a remote location, ensure that the design is locked for your project by copying the remote block design locally into the project.

6. Click Finish to close the Add Sources dialog box and add the block design to your project.

In the Sources window, you can see the block design added under Design Sources.

![Figure 4-14: Imported Block Design in the Sources Window](image)

7. Double-click the block design to open it in the Vivado IP integrator.

**TIP:** You might need to update the IP used in the block design, or validate the block design, generate a wrapper, and synthesize and implement the design. These topics were previously described in this document.

---

### Revision Control for Block Designs

Revision control systems can be used to manage the various source files associated with Vivado IP integrator block designs, in both Project and Non-Project Mode. As block designs are developed, and get more complex, it is a challenge to keep track of the different iterations of the design, and to facilitate project management and collaboration in a team-design environment.

Refer to this link in the UltraFast Design Methodology Guide for the Vivado Design Suite (UG949) [Ref 9] for more information on using the Vivado Design Suite with revision control software.
Exporting a Hardware Definition to SDK

To start software development before a bitstream is created, you can export the hardware definition to the software development toolkit (SDK) after generating the design. This exports the necessary XML files needed for SDK to interpret the IP used in the design and also the memory-mapping from the processor perspective.

After a bitstream is generated and the design is exported, then the device can be downloaded and the software can run on the processor.

When the output products for the block design are generated, an archive that contains the pertinent information for exporting the hardware to SDK is created. This archive is called `<top_level_design_name>.hwdef`, and is located in the synthesis directory such as `<project_name>.runs/synth_1`.

For a Zynq®-7000 processor-based design, an archive contains the following files:

- `ps7_init.c`
- `ps7_init.h`
- `ps_init.html`
- `ps7_init.tcl`
- `hwdef.xml`
- `<bd_name>.hwh`

For a MicroBlaze™-based processor design, an archive contains the following files:

- `hwdef.xml`
- `<bd_name>.hwh`

After you implement the design and generate the bitstream, a `<top_level_design_name>.sysdef` archive is generated.

For a Zynq-7000 processor-based design, the `/sysdef` archive contains the following files:

- `ps7_init.c`
- `ps7_init.h`
- `ps_init.html`
- `ps7_init.tcl`
- `sysdef.xml`
- `<bd_name>.hwh`
- `<top_level_design_name>.bit`
• `<top_level_design_name>.mmi`

For a MicroBlaze processor-based design, the archive contains the following files:

• `<bd_name>.hwh`
• `<top_level_design_name>.bit`
• `<top_level_design_name>.mmi`
• `sysdef.xml`

When you export the hardware, the IP integrator copies the `sysdef.xml` file into a specified location. Use the following steps:

1. Select **File > Export > Export Hardware** to open the Export Hardware dialog box.

![Export Hardware Dialog Box](image)

*Figure 4-15: Export Hardware Dialog Box*

The options in the Export Hardware dialog box are, as follows:

• **Include bitstream**: Includes the bitstream as a part of the exported data to SDK.
• **Export to**: Lets you select a location to which to export the hardware definitions.
  • In a typical project based flow, leave this setting to the default value of `<Local to Project>`.
  • In a project-based flow, the hardware is exported to `<project_name>/<project_name>.sdk`.

2. To launch SDK after the hardware has been exported, click **File > Launch SDK**.

The Launch SDK dialog box opens, as shown in the following figure.
Chapter 4: Working with Block Designs

In a typical project based flow, leave the default options as defined `<Local to Project>`. If you chose a different location while exporting the hardware, the Exported location and Workspace must be set to that location.

After SDK launches, you can create a custom application project using the exported hardware definitions. SDK creates the necessary drivers and board support package for the target hardware.

---

### Adding and Associating an ELF File to an Embedded Design

In a microprocessor-based design such as a MicroBlaze design, an ELF file generated in the SDK (or in other software development tool) can be imported and associated with a block design in the Vivado IDE. A bitstream can then be generated that includes the ELF contents from the Vivado IDE and run on target hardware. There are two ways in which you can add the ELF file to an embedded object.

#### Adding ELF and Associating it With an Embedded Processor

To add an ELF and associate it with an embedded processor, do the following:

1. In the Sources window, select and right-click Design Sources.
2. Select Add Sources.

   Alternatively, you can add design sources by selecting Flow Navigator > Add Sources.

The Add Sources dialog box opens.

**Add or Create Design Sources** is selected by default. This option adds the ELF file as a design file as well as a simulation source.
TIP: If you are adding an ELF file for simulation purposes only, select Add or Create Simulation Sources.

3. Click Next.

The Add or Create Design Sources dialog box opens as shown in Figure 4-17, page 71.

4. In the Add or Create Design Sources dialog box, click the + icon to Add Files.

5. The Add Source Files dialog box opens.

6. Navigate to the ELF file, select it, and click OK.
In the Add or Create Sources dialog box, you see the ELF file added to the project.

7. Either copy the ELF file into the project by checking **Copy sources into project** or leave the option unchecked if you want to work with the original ELF file. Then click **Finish**.

Under the Sources window, you see the ELF file added under the ELF folder, as shown in the following figure.

![Figure 4-18: Add Sources Dialog Box](image1)

You then associate the ELF file with the microprocessor design. To do so:

8. In the Sources window, select and right-click the block design and select **Associate ELF Files** as shown in **Figure 4-20, page 73**.
You can associate an ELF for synthesis or simulation by clicking the appropriate browse icon (Under Design Sources or Simulation Sources) and browse to the newly added ELF file to the design.
The Select ELF files dialog box opens.

9. Click on **Add Files**.

10. Navigate to the file that has been added as Design Sources.
11. Highlight the file, as shown in the following figure, and click **OK**.

![Figure 4-23: Associating ELF Files with a Microprocessor](image)

12. Make sure that the ELF file is populated in the **Associated ELF File** column and click **OK**.

![Figure 4-24: Highlight the ELF File to Associate](image)
Adding and Associating an ELF File in a Single Step

You can add and associate an ELF file as follows:

1. In the Sources window, right-click the block design, and select **Associate ELF files**.

2. In the Associate ELF Files dialog box, click the browse button on either the processor instance under Design Sources or Simulation Sources, as shown in the following figure.

3. In the Associate ELF File dialog box, click **Add Files**, as shown in Figure 4-27, page 77.
4. In the Add Source Files dialog box, shown in the following figure, navigate to the folder where the ELF file is located. Select the file and click **OK**.

![Add Source Files](figure4-28.png)

*Figure 4-28: Navigate to the Directory where the ELF File is Located*

5. In the Associate ELF file dialog box, select the newly-added ELF file and click **OK**.

![Select ELF Files](figure4-27.png)

*Figure 4-27: Select and Add ELF File*

6. In the Associate ELF Files dialog box, verify that the new ELF file is shown in the processor instance field and click **OK**.
With the ELF file added to the project, the Vivado tools automatically merge the Block RAM memory information (MMI file) and the ELF file contents with the device bitstream (BIT) when generating the bitstream to program the device.

**CAUTION!** The ELF file is not copied as a part of the design sources when this method is used.

You can also do the association in command-line mode using the UpdateMEM utility. See this link in the Vivado Design Suite User Guide: Embedded Processor Hardware Design (UG898) for more information.
Propagating Parameters in IP Integrator

Introduction

Parameter propagation is one of the most powerful features available in IP integrator. The feature enables an IP to auto-update its parameterization based on how it is connected in the design. IP can be packaged with specific propagation rules, and IP integrator will run these rules as the diagram is generated.

For example, in the following figure, IP0 has a 64-bit wide data bus. IP1 is then added and connected, as is IP2.

In this case, IP2 has a default data bus width of 32 bits.

When you run the parameter propagation rules, you are alerted to the fact that IP2 has a different bus width. Assuming that the data bus width of IP2 can be changed through a change of parameter, IP integrator can automatically update IP2.

If the IP cannot be updated to match properties based on its connection, then an error displays, alerting you of potential issues in the design. This simple example demonstrates the power of parameter propagation. The types of errors that can be corrected or identified by parameter propagation are often errors not found until simulation.
Using Bus Interfaces

A bus interface is a grouping of signals that share a common function. An AXI4-Lite master, for example, contains a large number of individual signals plus multiple buses, which are all required to make a connection. One of the important features of IP integrator is the ability to connect a logical group of bus interfaces from one IP to another, or from the IP to the boundary of the IP integrator design or even the FPGA IO boundary. Without the signals being packaged as a bus interface, the IP’s symbol will show an extremely long and unusable list of low-level ports, which will be difficult to connect one by one.

A list of signals can be grouped in IP-XACT using the concept of a bus Interface with its constituent port map that maps the physical port (available on the IP’s RTL or netlist) to a logical port as defined in the IP-XACT abstraction Definition file for that interface type.

Common Internal Bus Interfaces

Some common examples of bus interfaces are buses that conform to the AXI specification such as AXI4, AXI4-Lite and AXI-Stream. The AXI-MM interface includes all three subsets (AXI4, AXI3, and AXI4-Lite). Other interfaces include block RAM.

I/O Bus Interfaces

Some Bus Interfaces that group a set of signals going to IO ports are called I/O interfaces. Examples include UART, I2C, SPI, Ethernet, PCIe, and DDR.

Special Signals

Special signals include:

- Clock
- Reset
- Interrupt
- Clock Enable
- Data (for traditional arithmetic IP which do not have any AXI interface (for example adders and subtractors, multipliers)

These special signals are described in the following sections.
Chapter 5: Propagating Parameters in IP Integrator

Clock

The clock interface can have the following parameters associated with them. These parameters are used in the design generation process and are necessary when the IP is used with other IP in the design.

- **ASSOCIATED_BUSIF**: The list contains names of bus interfaces, which run at this clock frequency. This parameter takes a colon-separated list (:) of strings as its value. If all interface signals at the boundary do not run at this clock rate, this field is left blank.

- **ASSOCIATED_RESET**: The list contains names of reset ports (not names of reset container interfaces) as its value. This parameter takes a colon-separated (:) list of strings as its value. If there are no resets in the design, this field is left blank.

- **ASSOCIATED_CLKEN**: The list contains names of clock enable ports (not names of container interfaces) as its value. This parameter takes a colon-separated (:) list of strings as its value. If there are no clock enable signals in the design, this field is left blank.

- **FREQ_HZ**: This parameter captures the frequency in hertz at which the clock is running in positive integer format. This parameter needs to be specified for all output clocks only.

- **PHASE**: This parameter captures the phase at which the clock is running. The default value is 0. Valid values are 0 to 360. If you cannot specify the PHASE in a fixed manner, then you must update it in bd.tcl, similar to updating FREQ_HZ.

- **CLK_DOMAIN**: This parameter is a string ID. By default, IP integrator assumes that all output clocks are independent and assigns a unique ID to all clock outputs across the block design. This is automatically assigned by IP integrator, or managed by IP if there are multiple output clocks of the same domain.

To see the properties on the clock net, select the source clock port or pin and analyze the properties on the object. The following figure shows the Clocking Wizard.

![Clocking Wizard](image)

*Figure 5-2: Analyzing the Clock Properties in IP Integrator*
The following figure shows the clock properties.

![Clock Properties](image)

*Figure 5-3: Clock Properties*

These properties can also be reported by the following Tcl command:

```
report_property [get_bd_intf_ports sys_diff_clock]
```

The following figure shows the results of the Tcl command.

![Tcl Command Results](image)

*Figure 5-4: Reporting Clock Properties using Tcl Command*

You can also double-click a port or pin to see the customization dialog box for the selected object.

**Reset**

This container bus interface includes the POLARITY parameter. Valid values for this parameter are `ACTIVE_HIGH` or `ACTIVE_LOW`. The default is `ACTIVE_LOW`. 
To see the properties on the clock net, select the reset port or pin and analyze the properties on the object, as shown in the following figure.

The following figure shows the Properties window.

These properties can also be reported by the following Tcl command:

```tcl
report_property [get_bd_ports reset]
```

This command writes the following output to the Tcl Console.
Chapter 5: Propagating Parameters in IP Integrator

Interrupt

This bus interface includes the parameter, SENSITIVITY: Valid values for this parameter are LEVEL_HIGH, LEVEL_LOW, EDGE_RISING, and EDGE_FALLING. The default is LEVEL_HIGH.

To see the properties on the interrupt pin, highlight the pin as shown in the following figure, and look at the properties window.

Figure 5-7: Results from get_bd_ports /reset Command

```
report_property [get_bd_ports /reset]
```

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Read-only</th>
<th>Visible</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASS</td>
<td>string</td>
<td>true</td>
<td>true</td>
<td>bd_port</td>
</tr>
<tr>
<td>CONFIG.POLARITY</td>
<td>string</td>
<td>false</td>
<td>true</td>
<td>ACTIVE_HIGH</td>
</tr>
<tr>
<td>DIR</td>
<td>string</td>
<td>true</td>
<td>true</td>
<td>I</td>
</tr>
<tr>
<td>INTF</td>
<td>string</td>
<td>true</td>
<td>true</td>
<td>FALSE</td>
</tr>
<tr>
<td>LEFT</td>
<td>string</td>
<td>false</td>
<td>true</td>
<td></td>
</tr>
<tr>
<td>LOCATION</td>
<td>string</td>
<td>false</td>
<td>true</td>
<td>140 #10</td>
</tr>
<tr>
<td>NAME</td>
<td>string</td>
<td>false</td>
<td>true</td>
<td>reset</td>
</tr>
<tr>
<td>PATH</td>
<td>string</td>
<td>true</td>
<td>true</td>
<td>/reset</td>
</tr>
<tr>
<td>RIGHT</td>
<td>string</td>
<td>false</td>
<td>true</td>
<td></td>
</tr>
<tr>
<td>TYPE</td>
<td>string</td>
<td>true</td>
<td>true</td>
<td>intr</td>
</tr>
</tbody>
</table>

Figure 5-8: Interrupt Properties: Block Diagram and Properties Window

These properties can also be reported by using the following Tcl command:

```
report_property [get_bd_pins /axi_uartlite_0/interrupt]
```

The interrupt command returns the following information.
Chapter 5: Propagating Parameters in IP Integrator

Clock Enable

There are two parameters associated with Clock Enable: \texttt{FREQ\_HZ} and \texttt{PHASE}.

How Parameter Propagation Works

In IP integrator, parameter propagation takes place when you choose to run Validate Design. You can do this in one of the following ways:

- Click \textbf{Validate Design} in the Vivado® IDE toolbar.
- Click \textbf{Validate Design} in the Design Canvas toolbar.
- Select \textbf{Tools} > \textbf{Validate Design} from the Vivado Menu.
- Use the Tcl command: \texttt{validate_bd_design}

The propagation Tcl provides a mechanism to synchronize the configuration of an IP instance with that of other instances to which it is connected. The synchronization of configuration happens at bus interface parameters.

IP integrator’s parameter propagation works primarily on the concept of assignment strength for an interface parameter. An interface parameter can have a strength of USER, CONSTANT, PROPAGATED, or DEFAULT. When the tool compares parameters across a connection, it always copies a parameter with higher strength to a parameter with lower strength.
Parameters in the Customization GUI

In the Non-Project Mode, you must configure all user parameters of an IP. However, in the context of IP integrator, any user parameters that are auto updated by parameter propagation are grayed out in the IP customization dialog box. A greyed-out parameter indicates that you cannot set the specific-user parameters directly on the IP; instead, the property values are auto-computed by the tool.

There are situations when the auto-computed values might not be optimal. In those circumstances, you may override these propagated values.

The cases in which you encounter parameter propagation are as follows:

- **Auto-computed parameters**: Parameters are auto-computed by the IP integrator and you cannot override them. For example, the **Ext Reset Logic Level** parameter in the following figure is grey, and **Auto** is placed next to the parameter to indicate you cannot change this parameter.

The following figure shows the Processor System Reset page.

![Auto-Computed Parameter](image)

*Figure 5-10: Auto-Computed Parameter*
Override-able parameters: Auto-computed parameters that you can override. For example, you can change the SLMB Address Decode Mask for the LMB BRAM Controller. When you hover the mouse on top of the slider button, it will tell you that the parameter is controlled by the system; but, you can change it by toggling the button from Auto to User. The following figure shows these settings.

User configurable parameters: User configurable only. The following figure shows such parameters outlined in red.

Constants: Parameters that cannot be set.
Parameter Mismatch Example

The following is an example of a parameter mismatch on the `FREQ_HZ` property of a clock pin. In this example, the frequency does not match between the `S01_AXI` port and the `S_AXI` interface of the AXI Interconnect. This error is revealed when the design is validated.

- The `S01_AXI` port has a frequency of 500 MHz as can be seen in the properties window.
- The `S01_AXI` interface of the AXI Interconnect is set to a frequency of 50 MHz.

You can fix this error by changing the frequency in the property, or by double-clicking the `S01_AXI` port and correcting the frequency in the **Frequency** field of the customization dialog box.
After you change the frequency, re-validate the design to ensure there are no errors.

Figure 5-14: Change Frequency Port in Properties Window
Overview

In-system debugging lets you debug your design in real-time on your target hardware. This is an essential step in design completion. Invariably, one comes across a situation which is extremely hard to replicate in a simulator. Therefore, there is a need to debug the problem in the FPGA. In this step, you place an instrument into your design with special debugging hardware to provide you with the ability to observe and control the design. After the debugging process is complete, you can remove the instrumentation or special hardware to increase performance and reduce logic.

The Vivado® IP integrator provides ways to instrument your design for debugging which is explained in the following sections. There are two flows explained:

- The HDL instantiation flow
- The netlist insertion flow

Choosing the flow depends on your preference and types of nets and signals that you are interested in debugging.

As an example:

- If you are interested in performing hardware-software co-verification using the cross-trigger feature of a MicroBlaze™ or Zynq®-7000 processor, then you can use the HDL instantiation flow.
- If you are interested in verifying interface level connectivity then you can use the HDL instantiation flow.
- If you are interested in analyzing I/O ports and internal nets, then use netlist insertion.

In most of the cases, you will be using a combination of both flows to debug your design.
Chapter 6: Using the ILA to Debug IP Integrator Designs

Using the HDL Instantiation Flow in IP Integrator

You can instantiate an Integrated Logic Analyzer (ILA) in the IP integrator design and connect nets to the ILA that you are interested in probing. You can instantiate an ILA by following the steps described below.

1. Right-click on the block design canvas and select Add IP, as shown in the following figure.

![Add IP from Context Menu](image1)

*Figure 6-1: Add IP from Context Menu*

2. In the IP catalog, type ILA in the search field, select and double-click the ILA core to instantiate it on the IP integrator canvas.

The following figure shows the ILA core instantiated on the IP integrator canvas.

![Instantiated ILA Core](image2)

*Figure 6-2: Instantiated ILA Core*

3. Double-click the ILA core to reconfigure it.

The Re-Customize IP dialog box opens, as shown in Figure 6-3, page 92.
Chapter 6: Using the ILA to Debug IP Integrator Designs

The default option under the General Options tab shows AXI as the Monitor Type.

- If you are monitoring an entire AXI interface, keep the Monitor Type as AXI.
- If you are monitoring non-AXI interface signals, change the Monitor Type to Native.

You can change the Sample Data Depth and other fields as desired. For more information, see this link in the Vivado Design Suite User Guide: Programming and Debugging (UG908) [Ref 7].

**CAUTION!** You can only monitor one AXI interface using an ILA. Do not change the value of the C Num Monitor Slots. If more than one AXI interface is desired to be debugged, then instantiate more ILA cores as needed.

When you set the Monitor Type to Native, you can set the Number of Probes value, as shown in Figure 6-4, page 93. Set this value to the number of signals you want to be monitored.

Figure 6-3: Re-customized IP Dialog Box for the ILA Core
In Figure 6-4, the **Number of Probes** is set to 2 in the General Options tab. You can see under the Probe_Ports tab that two ports are displayed. The width of these ports can be set to the desired value.

4. Assuming that you want to monitor a 32-bit bus, set the Probe Width for Probe0 to 32.

After you configure the ILA, the changes are reflected on the IP integrator canvas as shown in the following figure.
Chapter 6: Using the ILA to Debug IP Integrator Designs

5. After configuring the ILA, make the required connections to the pins of the ILA on the IP integrator canvas, as shown in the following figure.

![Diagram of ILA connections](image)

**Figure 6-6: Instantiated ILAs to Monitor AXI and Non-AXI Signals**

**CAUTION!** If a pin connected to an I/O port is to be debugged, use `MARK_DEBUG` to mark the nets for debug. The following section describes this action.

6. Follow on to synthesize, implement and generate bitstream.

**Connecting I/O Ports to an ILA or VIO Debug Core**

Often, the I/O ports of a block design need to be probed for debugging. If the I/O ports of interest are bundled into interface ports then you must take care when connecting these I/O ports or pins to the ILA or VIO debug core. You must pull the ports of interest out of the bundled interface port.

As an example, consider the MicroBlaze processor design for the KC705 board, shown in the following figure. This design has a GPIO which is configured for using both the 8-bit LED interface and the 4-bit dip switches on the KC705 board.
To monitor these I/O interfaces, do the following:

1. Expand the GPIO interface pins so that you can see the pins that make up the interface pin.

   As you can see in Figure 6-7, page 95, the GPIO interface consists of an 8-bit output pin (gpio_io_o[7:0]), and the GPIO2 interface consists of a 4-bit input pin (gpio2_io_i[3:0]).

   To monitor these pins using debug probes you need to make them external to the block design. In other words, you must tie the pins inside the interface pin to an external port.

2. Right-click the pin, and select Make External.

   ![Figure 6-7: Monitoring I/O interfaces in a Block Design](image)

   ![Figure 6-8: Using Make External Option to Connect I/O Pin to an I/O Port](image)

You can see in the following figure that the pins that make up the GPIO and GPIO2 interface pins have been tied to external ports in the block design. Next you must connect these pins to an ILA debug core.
Chapter 6: Using the ILA to Debug IP Integrator Designs

CAUTION! When you make the I/O pins of an interface external, by connecting the input or output pins to external ports, do not delete the connection between the top-level interface pin and the I/O port. As shown in Figure 6-10, leave the existing top-level interface pin connected externally to the appropriate interface. You will see a warning as shown below:

---

CAUTION! WARNING: [BD 41-1306] The connection to interface pin /axi_gpio_0/gpio2_io_i is being overridden by the user. This pin will not be connected as a part of interface connection GPIO2

3. Use the Add IP command to instantiate an ILA core into the design, and configure it to support either Native or AXI mode.

   Note: In this case you must configure the ILA to support Native mode because you are not monitoring an AXI interface.

   You also need to configure two probes on the ILA core:
   - One that is 8-bits wide to monitor the LCD
   - One that is 4-bits wide to monitor the DIP Switches

4. Connect the ILA probes to the appropriate input/output pins, and connect the ILA clock to the same clock domain as that of the I/O pins, as shown in the following figure.

---

Send Feedback
With the debug cores inserted into the block design, the generated output products will include the necessary logic and signal probes to debug the design in the Vivado hardware manager. For more information on working with the Vivado hardware manager, and programming and debugging devices, see this link in the Vivado Design Suite User Guide: Programming and Debugging (UG908) [Ref 7].

Using the Netlist Insertion Flow

In this flow, you mark nets for debug that you are interested in analyzing in the block design. Marking nets for debug in the block design offers more control in terms of identifying debug signals during coding and enabling/disabling debugging later in the flow.

Marking Nets for Debug in the Block Design

To mark nets for debug, in the block design, highlight the net, right-click and select Mark Debug.

The nets that are marked for debug show a small bug icon placed on top of the net in the block design.

Also, a bug icon is placed on the nets to be debugged in the Design Hierarchy window, as shown in the following figure.

**Figure 6-11: Mark Debug Option**
Chapter 6: Using the ILA to Debug IP Integrator Designs

TIP: You can mark multiple nets for debug at the same time by highlighting them together, right-clicking and selecting Mark Debug.

Generating Output Products

You can Generate Output Products as follows:

1. In the Flow Navigator, click **Generate Block Design**.
   
   Alternatively, you can highlight the block design in the sources window, right-click and select **Generate Output Products**, as shown in the following figure.

   ![Generate Output Products Option](image)

   **Figure 6-13:** Generate Output Products Option

2. In the Generate Output Products dialog box, click **Generate**.
When you mark the nets for debug, the **MARK_DEBUG** attribute is placed on the net, which can be seen in the generated top-level HDL file.

This prevents the Vivado tools from optimizing and renaming the nets.

```
1799 signal VCC_1 : STD_LOGIC;
1800 signal axi_gpio_0_gpio_TRI_0 : STD_LOGIC_VECTOR ( 7 downto 0 );
1801 attribute MARK_DEBUG : boolean;
1802 attribute MARK_DEBUG of axi_gpio_0_gpio_TRI_0 : signal is true;
1803 signal axi_uartlite_i_uart_rx : STD_LOGIC;
1804 signal axi_uartlite_i_uart_TXD : STD_LOGIC;
1805 signal axi_uartlite_i_uart_TXD : STD_LOGIC;
1806 signal clk_wiz_i_locked : STD_LOGIC;
1807 signal mdm_1_debug_sys_clk : STD_LOGIC;
```

**Figure 6-15: MARK_DEBUG Attributes in the Generated HDL File**

### Synthesize the Design and Insert the ILA Core

The next step is to synthesize the top-level design. To do so:

1. From the **Flow Navigator > Synthesis > click Run Synthesis**.

   After synthesis finishes, the Synthesis Completed dialog box opens.

2. Select **Open Synthesized Design** to open the netlist design, and click OK.
Chapter 6: Using the ILA to Debug IP Integrator Designs

The Schematic and the Debug window opens. If the Debug window at the bottom of the GUI is not open, you can always open that window by choosing Windows > Debug from the menu. The following figure shows the Debug window.

![Debug Window Schematic](image)

**Figure 6-16**: Schematic and Debug Window View in the Vivado IDE

You can see all the nets that were marked for debug in the Debug window under the folder **Unassigned Debug Nets**. These nets need to be connected to the probes of an Integrated Logic Analyzer (ILA). This is the step where you insert an ILA core and connect these unassigned nets to the probes of the ILA.

2. Click the **Setup Debug** button in the Debug window toolbar.
   
   Alternatively, you can also select **Tools > Setup Debug**.
3. Click **Next**. The Set Up Debug dialog box opens, as shown in the following figure.

![Setup Debug Option](image1.png)

*Figure 6-17: Setup Debug Option*

![Set Up Debug Dialog Box](image2.png)

*Figure 6-18: Set Up Debug Dialog Box*
The Specify Nets to Debug page opens, as shown in the following figure.

![Specify Nets to Debug](image)

**Figure 6-19:** Select Nets Marked for Debug

4. Select a subset (or all) of the nets to debug. Every signal must be associated with the same clock in an ILA. If the clock domain association cannot be found by the tool, manually associate those nets to a clock domain by selecting all the nets that have the **Clock Domain** column specified as **undefined**.

**CAUTION!** You need to mark the entire interfaces that you are interested in debugging; however, if you are concerned with device resource usage, then the nets you do not need for debugging can be deleted while setting up the debug core.

4. To associate a clock domain to the signals that have the **Clock Domain** column specified as **undefined**, select the nets, right-click, and choose **Select Clock Domain**.

**TIP:** One ILA is inferred per clock domain by the **Set up Debug** dialog box.
5. In the Select Clock Domain dialog box, select the clock for the nets, and click **OK**.

6. In the Specify Nets to Debug dialog box, click **Next**.
Chapter 6: Using the ILA to Debug IP Integrator Designs

7. In the ILA (Integrated Logic Analyzer) General Options dialog box, shown in the following figure, select the appropriate options for triggering and capturing data, and click **Next**.

![Figure 6-22: Specify Nets to Debug Dialog Box](image1)

**Figure 6-22:** Specify Nets to Debug Dialog Box

![Figure 6-23: Setup the Trigger and Capture Modes in the ILA](image2)

**Figure 6-23:** Setup the Trigger and Capture Modes in the ILA
Chapter 6: Using the ILA to Debug IP Integrator Designs

The advanced triggering capabilities provide additional control over the triggering mechanism. Enabling advanced trigger mode enables a complete trigger state machine language that is configurable at runtime.

There is a three-way branching per state and there are 16 states available as part of the state machine. Four counters and four programmable counters are available and viewable in the Analyzer as part of the advanced triggering.

In addition to the basic capture of data, capture control capabilities let you capture the data at the conditions where it matters. This ensures that unnecessary block RAM space is not wasted and provides a highly efficient solution.

8. In the Summary page, verify that all the information looks correct, and click Finish.

The Debug window looks like the following figure after the ILA core has been inserted.

Note: All the buses (and single-bit nets) have been assigned to different probes.

The probe information also shows how many signals are assigned to that particular probe.

For example, in the following figure, probe0 has 32 signals (the 32 bits of the microblaze_1_axi_periph_m02_axi_WDATA) assigned.
Chapter 6: Using the ILA to Debug IP Integrator Designs

You are now ready to implement your design and generate a bitstream.

5. In the Flow Navigator > Program and Debug, click Generate Bitstream.

Because you made changes to the netlist (by inserting an ILA core), a dialog box, as shown in the following figure, displays asking if the design should be saved prior to generating bitstream.

![Save Project](image)

*Figure 6-26: Save Modified Constraints after ILA Insertion*

You can choose to save the design at this point, which writes the appropriate constraints in an active constraints file (if one exists), or create a new constraints file.

The constraints file contains all the commands to insert the ILA core in the synthesized netlist as shown in *Figure 6-27, page 107.*
Chapter 6: Using the ILA to Debug IP Integrator Designs

The benefit of saving the project is that if the signals marked for debug remain the same in the original block design, then there is no need to insert the ILA core after synthesis manually as these constraints will take care of it. Therefore, subsequent iteration of design changes will not require a manual core insertion.

If you add more nets for debug (or unmark some nets from debug) then you must open the synthesized netlist and make appropriate changes using the Set up Debug wizard.

If you do not chose to save the project after core insertion, none of the constraints show up in the constraints file and you must insert the ILA core manually in the synthesized netlist in subsequent iterations of the design.

With the debug cores and signal probes inserted into the top-level design, you are ready to debug the design in the Vivado hardware manager. For more information on working with the Vivado hardware manager, and programming and debugging devices, see this link in the Vivado Design Suite User Guide: Programming and Debugging (UG908) [Ref 7].

Figure 6-27: XDC Constraints for ILA Core Insertion

```
C:/temp/my_migration/my_migration.srcs/constra_1/new/design_1_wrapper.xdc
create_debug_core u_ila_0 labtools_ila_v3
set_property ALL_PROBESAME_MU true [get_debug_cores u_ila_0]
set_property ALL_PROBESAME_MU_CNT 4 [get_debug_cores u_ila_0]
set_property C_ADV_TRIGGER true [get_debug_cores u_ila_0]
set_property C_DATA_DEPTH 1024 [get_debug_cores u_ila_0]
set_property C_EN_STRG_QUAL true [get_debug_cores u_ila_0]
set_property C_INPUT_FIPE_STAGES 0 [get_debug_cores u_ila_0]
set_property C_TRIGIN_EN false [get_debug_cores u_ila_0]
set_property C_TRIGOOUT_EN false [get_debug_cores u_ila_0]
set_property port_width 1 [get_debug_ports u_ila_0/clk]
connect_debug_port u_ila_0/clk [get nets [list design_1_i/clk_out1]]
set_property port_width 32 [get_debug_ports u_ila_0/probe0]
```
Using Tcl Scripts to Create Block Designs within Projects

Overview

Typically, you create a new design in a project-based flow in the Vivado® Integrated Design Environment (IDE). After you assemble the initial design, you might want to re-create the design using a scripted flow in the GUI or in batch mode. This chapter guides you through creating a scripted flow for block designs.

Creating a Design in the Vivado IDE

Create a project and a new block design in the Vivado IDE as described in Chapter 2, Creating a Block Design. When the block design is complete, your canvas contains a design like the example in the following figure.

![Block Design Canvas](image)

*Figure 7-1: Block Design Canvas*

With the block design open, type the following command in the Tcl Console:

```
write_bd_tcl <path to file/filename>
```

This creates a Tcl file that can be sourced to re-create the block design.
This Tcl file has embedded information about the version of the Vivado tools in which it was created, and, consequently this design cannot be used across different releases of the Vivado Design Suite. The Tcl file also contains information about the IP present in the block design, their configuration, and the connectivity.

**CAUTION!** Use the script produced by `write_bd_tcl` in the release in which it was created only. The script is not intended for use in other versions of the Vivado Design Suite.

---

**Saving the Vivado Project Information in a Tcl File**

The overall project settings can be saved by using the `write_project_tcl` command, as follows:

```
write_project_tcl <path to file/filename>
```

For a Vivado project that consists of a block diagram, the Tcl file generated from `write_project_tcl` command could look like the following:
Chapter 7: Using Tcl Scripts to Create Block Designs within Projects

In the preceding Tcl file, the block design file (.bd) is read explicitly as shown by the highlighted code.

There are cases in which you may want to re-create the block design, rather than simply read the block design file. In this case, you will need to modify the Tcl file generated by the write_project_tcl command as shown in Figure 7-3, page 111:

**TIP:** If you choose to simply read the existing block design file, and not re-create the block design, then the Tcl file does not need to be modified.
Figure 7-3: Code Snippet from the Tcl File to Recreate the Block Design using the Output File

As can be seen in the figure above, the Tcl file from the write_project_tcl file has been modified to source another Tcl script that was created using the write_bd_tcl command. The sourced block design Tcl script re-creates the block design every time the project Tcl script is run, rather than reading a block design file.
Using IP Integrator in Non-Project Mode

Overview

Non-Project Mode is for users who want to manage their own design data or track the design state. In this flow, Vivado® tools read the various source files and implement the design through the entire flow in-memory. At any stage of the implementation process, you can generate a variety of reports based on your script. When running in Non-Project Mode, it is also important to note that this mode does not enable project-based features such as source file and run management, cross-probing back to source files, and design state reporting. Essentially, each time a source file is updated on the disk, you must know about it and reload the design. There are no default reports or intermediate files created within the Non-Project Mode. You need to have your script control the creation of reports with Tcl commands. For details of working in Non-Project Mode see this link in Vivado Design Suite User Guide: Design Flows Overview (UG892) [Ref 2].

Creating a Flow in Non-Project Mode

Vivado tools can be invoked in Tcl mode instead of the usual Project Mode by issuing the following commands in the Tcl Console. The recommended approach in this mode is to create a Tcl script and source it from the Vivado prompt:

```
Vivado% vivado -mode tcl
```

In non-project mode, you have to create an in-memory project, and set your project options as shown below:

```
create_project -in_memory -part xc7k325tffg900-2
set_property target_language VHDL [current_project]
set_property board_part xilinx.com:kc705:part0:0.9 [current_project]
set_property default_lib work [current_project]
```

In non-project mode, there is no project file saved to disk. Instead, an in-memory Vivado project is created. The device/part/target-language of a block design is not stored as a part of the block design sources. Therefore, it is recommended that you specify the `create_project` -in_memory command to define the desired part settings before issuing the read_bd command.
Chapter 8: Using IP Integrator in Non-Project Mode

After the project has been created the source file for the block design can be added to the project.

This can be done in two different ways. First, assuming that there is an existing block design that has been created in a Project Mode with the entire directory structure of the block design intact, you can add the block design using the read_bd tcl command as follows:

```
Vivado% read_bd <absolute path to the bd file>
```

**CAUTION!** You must have the project settings (board, part, and user repository) match with the project settings of the original project in which the bd was created. Otherwise, the IP in the block design will be locked.

After the block design is added successfully, you need to add your top-level RTL files and any top-level XDC constraints.

```
Vivado% read_verilog <top-level>.v
Vivado% read_xdc <top-level>.xdc
```

You can also create top-level HDL wrapper file using the command below because a BD source cannot be synthesized directly.

```
Vivado% make_wrapper -files [<path to bd>/<bd instance name>.bd] –top
add_files -norecurse <path to bd>/< bd instance name >_wrapper.vhd
update_compile_order -fileset sources_1
```

This creates a top-level HDL file and adds it to the source list.

If setting the block design as an out-of-context module is desired, then use the commands below to generate a synthesized design check point (DCP) for the block design.

```
create_fileset -blockset -define_from <block_design_name> <block_design_name>
reset_run <block_design_name>_synth_1 -from_step synth_design
```

For a MicroBlaze™-based processor design, populate the I-LMB with either a Bootloop or your own executable in ELF format. You then needs to add the ELF and associate it with the MicroBlaze instance. The following steps will do this.

```
vivado% add_files <ELF file Targeted to BRAM with .elf extension>
vivado% set_property MEMDATA.ADDR_MAP_CELLS {<bd instance name>/microblaze_0}
[get_files <BRAM Targeted ELF File>]
```

If the design has multiple levels of hierarchy, you need to ensure that the correct hierarchy is provided. After this, go through the usual synthesis, place, and route steps to implement the design.

**TIP:** One aspect that needs to be kept in mind is that for the synthesis (synth_design) step, you need to provide the target part as the default target part, which might not be the same as the desired one.
synth_design -top <path to top level wrapper file> -part <part>
opt_design
place_design
route_design
write_bitstream top

To export the implemented hardware system to SDK, use the following command:

write_sysdef -hwdef “C:/Data/ug940/lab1/zynq_design.hwdef” \\
-bitfile “<project_name>/<project_name>.runs/impl_1/zynq_design.bit” \\
-meminfo “<project_name>/<project_name>.runs/impl_1/zynq_design.bmm” \\
-file “C:/Data/ug940/lab1/zynq_design.sysdef”

See the Vivado Design Suite Tcl Command Reference (UG835) [Ref 1] for more on the write_sysdef or write_hwdef commands.

Non-Project Script

The following is a sample script for creating a block design in Non-Project Mode.

create_project -in_memory -part xc7k325tffg900-2
set_property target_language VHDL [current_project]
set_property board_part xilinx.com:kc705:part0:0.9 [current_project]
set_property default_lib work [current_project]
read_bd ./bd/mb_ex_1/mb_ex_1.bd
open_bd ./bd/mb_ex_1/mb_ex_1.bd
generate_target all [get_files ./bd/mb_ex_1/mb_ex_1.bd]
read_vhdl ./bd/mb_ex_1/hdl/mb_ex_1_wrapper.vhd
write_hwdef -file mb_ex_1_wrapper.hwdef
set_property source_mgmt_mode All [current_project]
update_compile_order -fileset sources_1
update_compile_order -fileset sources_1
update_compile_order -fileset sim_1
synth_design -top mb_ex_1_wrapper -part xc7k325tffg900-2
opt_design
place_design
route_design
write_bitstream top
file mkdir c:/temp/export_hw_np_mode/sdk
write_sysdef -hwdef mb_ex_1_wrapper.hwdef -bitfile top.bit -file mb_ex_1_wrapper.hdf
Chapter 9

Updating Designs for a New Release

Overview

As you upgrade your Vivado® Design Suite to the latest release, you must upgrade the block designs created in the Vivado IP integrator as well.

- The IP version numbers change from one release to another.
- When IP integrator detects that the IP contained within a block design are older versions of the IP, it locks those IP in the block design.

If the intention is to keep older version of the block design (and the IP contained within it), then you do not need to do any operations such as modifying the block design on the canvas, validating it and/or resetting output products, and re-generating output products. In this case, the expectation is that you have all the design data from the previous release intact. You can use the block design from the previous release “as is” by synthesizing and implementing the design.

The recommended practice is to upgrade the block design with the latest IP versions, make any necessary design changes, validate design and generate target.

You can update projects in two ways:

- Upgrading a Block Design in Project Mode
- Upgrading a Block Design in Non-Project Mode

This chapter describes both methods.

Upgrading a Block Design in Project Mode

To upgrade a block design in Project Mode:

1. Launch the latest version of the Vivado Design Suite.
2. From the Vivado IDE, click Open Project and navigate to the design that was created from a previous version of Vivado tools.
Chapter 9: Updating Designs for a New Release

The **Older Project Version** pop-up opens. **Automatically upgrade for the current version** is selected by default.

Although you can upgrade the design from a previous version by selecting the **Automatically upgrade for the current version**, it is highly recommended that you save your project with a different name before upgrading. To do so:

3. Select **Open project in read-only mode**, as shown in the following figure, and click **OK**.

![Figure 9-1: Open Project in Read-Only Mode](image)

4. Select **Save Project As...** as shown in the following figure.

![Figure 9-2: Save Project As](image)
5. When the Save Project As dialog box opens, as shown in the following figure, type the name of the project, and click **OK**.

![Save Project As Dialog Box](image1)

*Figure 9-3: Specify Project Name*

The Project Upgraded dialog box opens, as shown in the following figure, informing you that the IP used in the design may have changed and therefore need to be updated.

![Project Upgraded Dialog Box](image2)

*Figure 9-4: Project Upgraded Dialog Box*

6. Click **Report IP Status**.

Alternatively, from the menu select **Tools > Report > Report IP Status**.
7. If any of the IP in the design has gone through a major version change, then the following message opens. Click **OK**.

![Report IP Status Results](image)

**Figure 9-5: Report IP Status Results**

8. In the IP Status window, look at the different columns and familiarize yourself with the IP Status report. Expand the block design by clicking on the + sign and look at the changes that the IP cores in the block design.

![IP Status Report](image)

**Figure 9-6: IP Status Report**

The top of the IP Status window shows the summary of the design. It reports how many changes are needed to upgrade the design to the current version. The changes reported are Major Changes, Minor Changes, Revision Changes and Other Changes. These changes are reported in the IP Status column as well.

- **Major Changes**: The IP has gone through a major version change; for example, from Version 2.0 to 3.0. This type of change is not automatically selected for upgrade. To select this for upgrade, uncheck the Upgrade column for the block design and then re-check it.
- **Minor Changes**: The IP has undergone a minor version change; for example, from version 3.0 to 3.1.
- **Revision Changes**: A revision change has been made to the IP; for example, the current version of the IP is 5.0, and the upgraded version is 5.0 (Rev. 1)
9. Click the **More info...** link in the Change Log column to see a description of the change.

![Change Log for AXI BRAM Controller](image)

**Figure 9-7:** Inspect the Change Log by in More Info link

The Recommendation column also suggests that you need to understand what the changes are before selecting the IP for upgrade.

10. When you understand the changes and the potential impact on your design, you can click **Upgrade Selected**.

The Upgrade IP dialog box opens to confirm that you want to proceed with upgrade.

**TIP:** You cannot select individual IP of a block design to upgrade, and let the others remain in their current versions. You must upgrade all IP in the block design at the same time.

11. Click **OK**.

When the upgrade process is complete, a Critical Messages dialog box opens, such as the one in the following figure, informing you about any critical issues to which you need to pay attention.

![Critical Messages Dialog Box](image)

**Figure 9-8:** Critical Messages Dialog Box

12. Review any critical warnings and other messages that may be flagged as a part of the upgrade. Click **Ok**.
Chapter 9: Updating Designs for a New Release

If there are multiple diagrams in the design, the IP Status window shows the status of IP in all the diagrams as shown in Figure 9-9, page 120.

Running Design Rule Checks

From the toolbar, click Validate Design, as shown in the following figure.

You can also do this by clicking the Validate Design button in the block design toolbar.

Figure 9-9: IP Status Window for Multiple Diagrams

Figure 9-10: Validate Design
Chapter 9: Updating Designs for a New Release

Regenerating Output Products

In the Sources window, right-click the block diagram, and select **Generate Output Products**.

Alternately, in the **Flow Navigator > IP integrator**, click the **Generate Block Design**.

13. In the Generate Output Products dialog box, click **Generate**.
Chapter 9: Updating Designs for a New Release

Creating or Changing the HDL Wrapper

If you previously created an HDL wrapper in the previous version of the design, you may want to re-create it to reconcile any design changes. If you had chosen the option to let the Vivado tools create and manage the top-level wrapper for you, then the wrapper file will be updated as a part of generating the block design or generating output products as defined in the previous section. If you modified the HDL wrapper manually, then you will need to manually make any updates that may be necessary in the HDL wrapper.

1. In the Sources window, shown in the following figure, right-click the block diagram, and select Create HDL Wrapper.

![Create HDL Wrapper Option](image)

*Figure 9-13: Create HDL Wrapper Option*

The Create HDL Wrapper dialog box opens. You have two choices to make at this point. You can create a wrapper file that can be edited, or you can let the Vivado tools manage the wrapper file for you.

2. Make your selection from the dialog box shown in the following figure, and click OK.

![Create HDL Wrapper Dialog Box](image)

*Figure 9-14: Create HDL Wrapper Dialog Box*

You can now synthesize, implement, and generate the bitstream for the design.
Upgrading a Block Design in Non-Project Mode

You can open an existing project from a previous release using the Non-Project Mode flow and upgrade the block design to the current release of Vivado. You can use the following script as a guideline to upgrade the IP in the block diagram.

```
# Open an existing project from a previous Vivado release
open_project <path_to_project>/project_name.xpr
update_compile_order –fileset sim_1
# Open the block diagram
read_bd {<path_to_bd>/bd_name.bd}
# Make the block diagram current
current_bd_design bd_name.bd
# Upgrade IP
upgrade_bd_cells [get_bd_cells –hierarchical * ]
# Reset output products
reset_target {synthesis simulation implementation} [get_files
<path_to_project>/project_name.srcs/sources_1/bd/bd_name/bd_name.bd]

# Generate output products
generate_target {synthesis simulation implementation} [get_files
<path_to_project>/project_name/project_name.srcs/sources_1/bd/bd_name/bd_name.bd]
# Create HDL Wrapper (if needed)
make_wrapper -files [get_files
<path_to_project>/project_name/project_name.srcs/sources_1/bd/bd_name/bd_name.bd]
-‐top
# Overwrite any existing HDL wrapper from before
import_files -force -norecurse
<path_to_project>/project_name/project_name.srcs/sources_1/bd/bd_name/hdl/bd_name_wrapper.v
update_compile_order -fileset sources_1
# Continue through implementation
```
Chapter 10

Using the Platform Board Flow in IP Integrator

Overview

The Vivado Design Suite is board aware. The tools know the various interfaces present on the target boards and can customize and configure an IP to be connected to a particular board component. Several 7 series and an UltraScale board is currently supported.

The IP integrator shows all the interfaces to the board in a separate tab called the Board tab. When you use this tab to select the desired components and the Designer Assistance offered by IP integrator, you can easily connect your design to the board component of your choosing. All the I/O constraints are automatically generated as a part of using this flow.

Select a Target Board

When a new project is created in the Vivado environment, you have the option to select a target board from the Default Part page of the New Project dialog box.
Chapter 10: Using the Platform Board Flow in IP Integrator

You can filter the list of available boards based on Vendor, Display Name, and Board Revision, as shown in the following figure.

- **Board Rev**: Allows filtering based on the revision of the board.
  - Setting the **Board Rev** to **All** shows revisions of all the boards that are supported in Vivado.
  - Setting **Board Rev** to **Latest** shows only the latest revision of a target board. Various information such as resources available and operating conditions are also listed in the table.

When you select a board, the project is configured using the pre-defined interface for that board.
Create a Block Design to use the Board Flow

The real power of the board flow can be seen in the IP integrator tool.

From Flow Navigator > IP integrator, start a new block design by clicking Create Block Design.

As the design canvas opens, you see a Board window, as shown in the following figure.

![Board Window](image)

Figure 10-3: Board Window

This Board window lists all the possible components for an evaluation board (in the preceding figure the KC705 board is displayed). By selecting one of these components, an IP can be quickly instantiated on the block design canvas.

The first way of using the Board window, is to select a component from the Board window and drag it on the block design canvas. This instantiates an IP that can connect to that component and configures it appropriately for the interface in question. It then also connects the interface pin of the IP to an I/O port. As an example, when you drag an drop the Linear Flash component under the External Memory folder, on the IPI canvas, the AXI EMC IP is instantiated and the interface called EMC_INTF is connected. See Figure 10-4, page 127.
The second way to use an interface on the target board, is to double-click the unconnected component in question from the Board tab. As an example, when you double click the DDR3 SDRAM component in the Board tab, the Connect Board Component dialog box opens, as shown in the following figure.

Figure 10-4: Dragging and Dropping an Interface on the Block Design Canvas

Figure 10-5: Connect Board Part Interface Dialog Box
mig_ddr_interface is selected by default. If there are multiple interfaces listed under the IP then select the interface desired.

Select the mig_ddr_interface as shown in the following figure, and click **OK**.

Notice that the IP is placed on the Diagram canvas and connections are made to the interface using the I/O ports. As shown in the following figure, the IP is all configured accordingly to connect to that interface.

![Diagram Canvas](image)

**Figure 10-6:** IP instantiated, Configured, and Connected to Interfaces on the Diagram Canvas

As an interface is connected, that particular interface now shows up as a shaded circle in the Board window.

![Board Window](image)

**Figure 10-7:** Board Window after Connecting to an Interface

A component can also be connected using the **Auto Connect** option.

To do this, select and right-click the component and from the menu, select **Auto Connect**.
You will notice that the GPIO IP has been instantiated and the GPIO interface is connected to an I/O port, as shown in Figure 10-9, page 129.

If another component such as DIP switches is selected, the board flow is aware enough to know that a GPIO already is instantiated in the design and it re-uses the second channel of the GPIO.
Chapter 10: Using the Platform Board Flow in IP Integrator

The already instantiated GPIO is re-configured to use the second channel of the GPIO as shown in the following figure.

![GPIO Auto Connection](image)

Figure 10-10: GPIO Auto Connection

If an external memory component such as the Linear Flash or the SPI Flash is chosen, then as one of them is used the other component becomes unusable as only one of these interfaces can be used on the target board. In this case, the following message will pop-up when the user tries to drag the other interface such as the SPI Flash on the bd canvas.

![Auto Connect Warning](image)

Figure 10-12: Auto Connect Warning
Complete Connections in the Block Design

After the desired interfaces are used in the design, the next step is to instantiate a processor (in case of an processor-based design) or an AXI interconnect if this happens to be a non-embedded design to complete the design.

To do this, right-click on the canvas and select add IP. From the IP catalog choose the processor, such as MicroBlaze™ processor, as shown in the following example.

As the processor is instantiated, Designer Assistance becomes available, as shown in the following figure.

Click **Run Block Automation** to configure a basic processor sub-system. The processor sub-system is created which includes commonly used IP in a sub-system such as block memory controllers, block memory generator and a debug module.

Then you can use the Connection Automation feature to connect the rest of the IP in your design to the MicroBlaze processor by selecting **Run Connection Automation**.
Figure 10-15: Run Connection Automation to Complete Connections

The rest of the process is the same as needed for designing in IP integrator as described in of this document.
Using Third-Party Synthesis Tools in IP Integrator

Overview

Sometimes it is necessary to use a third-party synthesis tool as a part of the design flow. In this case, you will need to incorporate the IP integrator block design as a black-box in the top-level design. You can synthesize the top-level of the design in a third-party synthesis tool, write out an HDL or EDIF netlist, and implement the post-synthesis project in the Vivado environment.

This chapter describes the steps that are required to synthesize the black-box of a block design in a third-party synthesis tool. Although the flow is applicable to any third-party synthesis tool, this chapter describes the Synplify® Pro synthesis tool.

Setting the Block Design as Out-of-Context Module

You can create a design checkpoint (DCP) file for a block design by setting the block design as an Out-of-Context (OOC) module.

1. Select the block design in the Sources window, right-click to open the menu, and select the **Generate Output Products** command.

![Generate Output Products](image)
2. In the Generate Output Products dialog box, enable the **Out-of-Context per Block Design** option, as shown below. See *Generating Output Products, page 58* for more information.

![Out-of-Context Settings Dialog Box](image1.png)

**Figure 11-2: Out-of-Context Settings Dialog Box**

A square is placed next to the block design in the Sources window to indicate that the block design has been defined as an out-of-context module. The Design Runs window also shows an Out-of-Context Module Run for the block design.

![Sources and Design Runs](image2.png)

**Figure 11-3: Out-of-Context Settings Dialog Box**

3. When out-of-context synthesis run for the block design is complete, a design checkpoint file (DCP) is created for the block design. The DCP file also shows up in the Sources window, under the block design tree in the IP Sources tab. The DCP file is written to the following directory:

```
<project_name>/<project_name>.srcs/<sources_1>/bd/<block_design_name>
```

DCPs let you take a snapshot of your design in its current state. The current netlist, constraints, and implementation results are stored in the DCP. Using DCPs, you can:
- Restore your design if needed
- Perform design analysis
- Define constraints
- Proceed with the design flow

When the out-of-context run for the block design is created, two stub files are also created; one each for Verilog and VHDL. The stub file includes the instantiation template which can be copied into the top-level design to instantiate the black box of the block design. An example stub file is shown in the following figure.

![Example Stub File](image)

Figure 11-4: Example Stub File

Creating an HDL or EDIF Netlist in Synplify

Create a Synplify project and instantiate the black-box stub file (created in Vivado) along with the top-level HDL wrapper for the block design in the Synplify project. The block design is treated as a black-box in Synplify.

After the project is synthesized, an HDL or EDIF netlist for the project can be written out for use in a post-synthesis project.
Chapter 11: Using Third-Party Synthesis Tools in IP Integrator

Creating a Post-Synthesis Project in Vivado

The next step is to create a post-synthesis project in the Vivado IDE. Refer to this link in Vivado Design Suite User Guide: System-Level Design Entry (UG895) for more information.

1. Create a new Vivado project, and select the Post-synthesis Project option in the New Project wizard.

![Creating a Post-Synthesis Project](image1)

*Figure 11-5: Creating a Post-Synthesis Project*

**Note:** If the Do not specify sources at this time option is enabled, you can add design sources after project creation.

2. Click Next.

![Adding Files to the Post-Synthesis Project](image2)

*Figure 11-6: Adding Files to the Post-Synthesis Project*
3. In the Add Netlist Sources Page click on the ‘+’ sign to Add Files, as seen in Figure 11-6, page 136.

Add the EDIF netlist for the top-level design, and the design checkpoint file (DCP) for your block design. The DCP file for the block design is created when you generate output products with the Out-of-context per Block Design option. See The Generate Output Products dialog box, page 59 for more information.

4. Click Next.

![New Project window](image)

*Figure 11-7: Add the EDIF and DCP files to the Project*

5. On the Add Constraints page, add any constraints files (XDC) that are needed for the project, and click Next.

6. Specify the target part or target platform board as required by the project, and click Next.

7. Verify all the information for the project as presented on the New Project Summary page, and click Finish.
If you did not add the EDIF netlist file, DCP, or design constraints at the time you created the project, you can add those design source files in an open project by right-clicking in the Design Sources window and selecting **Add Sources** to add files as needed.

![Add HDL Netlist from Synplify and the DCP File to the Project](image)

**Figure 11-8:** Add HDL Netlist from Synplify and the DCP File to the Project

---

## Adding Top-Level Constraints

Prior to implementing the design, you must add any necessary design constraints to your project. The constraints file for the block design are added to the project when you add the block design to the netlist project. However, if you have changed the hierarchy of the block design, then you must modify the constraints in the XDC file to ensure that hierarchical paths used in the constraints have the proper design scope. For more information, refer to this [link](#) in the *Vivado Design Suite User Guide: Using Constraints* (UG903).

A constraints file can be added to the project at the time it is created, as discussed previously, or by right-clicking in the Sources window and choosing **Add Sources**.

---

## Adding an ELF File

If the block design has an executable and linkable format (ELF) file associated with it, then you will need to add the ELF file to the Vivado project. An ELF file can be added to the project at the time it is created, or by right-clicking in the Sources window and choosing **Add Sources**.

1. In the Add Sources dialog box, select **Add design sources**, as shown in Figure 11-9, page 139.
2. Click **Next**.
Chapter 11: Using Third-Party Synthesis Tools in IP Integrator

3. In the Add Design Sources page click on the ‘+’ sign to **Add Files**.
   
   When the Add Source Files dialog box opens make sure that **Files of type:** is set to All Files.

4. Browse to the folder containing the ELF file, select the ELF file and click OK.
5. Click **Finish** to add the file to the project.

   **Note:** You can also add the ELF file using the following Tcl command:
   
   ```tcl
   add_files <path_to_elf_file>/<file_name>.elf
   ```

   The added ELF file can be seen in the Sources window, as shown in the following figure.

   ![ELF File in Project](image)

   *Figure 11-11: ELF File in Project*

   After the ELF file has been added to the project, you must associate the ELF file with the embedded processor design object by setting the SCOPED_TO_REF and SCOPED_TO_CELL properties.

6. Select the ELF file in the Sources window.

   In the Source File Properties window click in the text field of the SCOPED_TO_CELLS and SCOPED_TO_REF properties to edit them.

7. Set the **SCOPED_TO_REF** property to the name of the block design.

8. Set the **SCOPED_TO_CELL** property to the instance name of the embedded processor cell in the block design.

   In **Figure 11-12, page 141** for example, SCOPED_TO_REF is `block_design_1`, and SCOPED_TO_CELL is `microblaze_0`.
You can also set these properties using the following Tcl commands:

```tcl
set_property SCOPE PROPERTIES { SCOPE_TO_REF <block_design_name> [get_files \<file_path>/file_name.elf]
set_property SCOPE PROPERTIES { SCOPE_TO_CELLS { <processor_instance> } [get_files \<file_path>/file_name.elf]
```

### Implementing the Design

Next the design can be implemented and a bitstream generated for the design.

1. In the Flow Navigator, under Program and Debug, click on **Generate Bitstream**.

![Figure 11-13: Open Implemented Design](image-url)
Chapter 11: Using Third-Party Synthesis Tools in IP Integrator

After the bitstream is generated you can open the implemented design to ensure that all timing constraints were met.

2. In the Bitstream Generation Completed dialog box, click on **Open Implemented Design**.

Verify timing by looking at the Timing Summary report. You can also ensure that BRAM INIT_STRINGS were populated with the ELF data.

3. From the main menu, select **Edit > Find**.
4. In the Find window, set the **PRIMITIVE TYPE** to **BMEM.BRAM** as shown below and click **OK**.

![Find BRAM](image)

5. In the Find Results window, select an instance of the **BRAM** and verify that the **INIT** properties have been populated in the Cell Properties window.
Chapter 11: Using Third-Party Synthesis Tools in IP Integrator

Figure 11-15: Verify BRAM INIT Properties
Appendix A

Additional Resources

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Xilinx Support.

See the Xilinx Solution Centers for support on devices, software tools, and intellectual property at all stages of the design cycle. Topics include design assistance, advisories, and troubleshooting tips.

References

8. ISE to Vivado Design Suite Migration Guide (UG911)
12. 7 Series FPGAs Memory Interface Solutions User Guide (UG586)
13. LogiCORE IP AXI Interrupt Controller (PG099)
14. UltraScale Architecture-Based FPGAs Memory Interface Solutions (PG150)
15. Vivado Design Suite Documentation
Training Resources

Xilinx provides a variety of training courses and QuickTake videos to help you learn more about the concepts presented in this document. Use these links to explore related training resources:

1. [Vivado Design Suite QuickTake Video: Designing with Vivado IP Integrator](#)
2. [Vivado Design Suite QuickTake Video: Targeting Zynq Devices Using Vivado IP Integrator](#)
3. [Essentials of FPGA Design Training Course](#)
4. [Vivado Design Suite Embedded Systems Design](#)
5. [Vivado Design Suite Advanced Embedded Systems Design](#)
Please Read: Important Legal Notices

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 Xilinx's limited warranty, please refer to Xilinx's Terms of Sale which can be viewed at http://www.xilinx.com/legal.htm#tos; 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 such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at http://www.xilinx.com/legal.htm#tos.

© Copyright 2013-2015 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Spartan, Virtex, Vivado, Zynq, UltraScale and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.

AMBA, AMBA Designer, ARM, ARM1176JZ-S, CoreSight, Cortex, and PrimeCell are trademarks of ARM in the EU and other countries.