Table of Contents

[Designing with IP Integrator using Block Design Containers 2](#_Toc156301631)

[Introduction 2](#_Toc156301632)

[Tutorial Design Description 2](#_Toc156301633)

[Design Creation Steps : Top – Bottom BDC flow 2](#_Toc156301634)

[Step 1: Creating a Project 2](#_Toc156301635)

[Step 2: Creating an IP Integrator Design with MicroBlaze Master 5](#_Toc156301636)

[Step 3: Adding New IP’s, Interfaces and IP Customization 13](#_Toc156301637)

[Step 4: Creating Connections to all IP’s and interfaces 18](#_Toc156301638)

[Step 5: Connecting all IP’s and Interfaces using both manual and automated connections 20](#_Toc156301639)

[Step 6: Using the Address Editor 23](#_Toc156301640)

[Step 7: Creating Hierarchies 24](#_Toc156301641)

[Step 8: Block Design Containers 27](#_Toc156301642)

[Design Creation Steps: Bottom - Up BDC flow 36](#_Toc156301643)

# Designing with IP Integrator using Block Design Containers

Introduction

The Xilinx® Vivado® Design Suite IP Integrator lets you create complex system designs by

instantiating and interconnecting IP cores from the Vivado IP catalog onto a design canvas. You can create designs interactively through the IP Integrator design canvas GUI, or programmatically using a Tcl programming interface.

This tutorial walks you through the steps for building a basic IP subsystem design using the IP

Integrator. You will instantiate a few IPs in the IP Integrator and then stitch them up to create an

IP sub-system design. While working through this tutorial, you will be introduced to the IP Integrator GUI, run design rule checks (DRC) on your design, and then integrate the design into a top-level design in the Vivado Design Suite. Finally, you will run synthesis and implementation and generate a bitstream on the design.

Tutorial Design Description

This tutorial is based on a simple processor-based IP Integrator design. It contains peripheral IP cores, AXI Interconnect core, SmartConnect core and DDR4 SDRAM. The major focus of this tutorial is to get familiar with Vivado IP integrator and to explore enhanced features like Block Design Containers (BDC).

For the purpose of learning the different IP Integrator (IPI) capabilities, we will manually do some of the steps described in this tutorial, instead of using an automated option all the time. This tutorial will use Graphical User Interface (GUI) flow of the IPI to demonstrate the Top - Bottom BDC creation steps. The last section would use TCL flow to demonstrate Bottom – Up BDC flow. For more information on IPI BDC flow please refer to UG994 Vivado Documentation.

The design targets a Kintex UltraScale KCU105 Evaluation Platform.

# Design Creation Steps : Top – Bottom BDC flow

## Step 1: Creating a Project

1. Open the Vivado Integrated Design Environment (IDE).

* On Linux, change to the directory where the Vivado tutorial design file is stored: cd

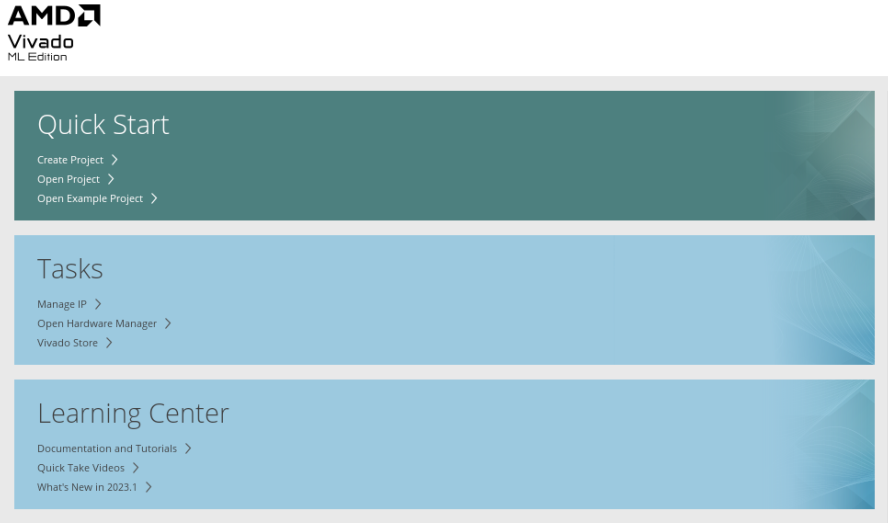
<Extract\_Dir>/Vivado\_Tutorial. Then launch the Vivado Design Suite: Vivado.

* On Windows, launch the Vivado Design Suite: **Start → All Programs → Xilinx Design Tools→ Vivado 2023.1**.

As an alternative, click the **Vivado 2023.1** Desktop icon to start the Vivado IDE.

The Vivado IDE Getting Started page contains links to open or create projects and to view

documentation, as shown in the following figure:



***Note*:** Your Vivado Design Suite installation may be called something different from Xilinx Design Tools

on the Start menu.

1. Under the Quick Start section, select **Create Project**.
2. The New Project wizard opens. Click **Next** to confirm the project creation.
3. In the Project Name page, shown in the following figure, set the following options:

a. In the Project name field, enter Lab-2.

b. In the Project location field, enter <IPI-Basics>, or choose any location as per your working folder structure.

|  |
| --- |
|  |

1. Ensure that Create project subdirectory is checked and click **Next.**
2. In the Project Type page, select **RTL Project**, and select **Do not specify sources at this time**, then click **Next**, as shown in the following figure:

Graphical user interface, text, application, email

Description automatically generated

1. Click next, and then you will land on the Default Part page, as shown in the following figure. Select the Kintex Ultrascale KCU105 Evaluation Platform under **Boards** tab.

|  |
| --- |
|  |

1. Review the project summary in the New Project Summary page.

|  |
| --- |
|  |

1. Click Finish to create the project.
2. The new project opens in the Vivado IDE.

## Step 2: Creating an IP Integrator Design with MicroBlaze Master

1. This step will show how Vivado is board aware of the connections and the interfaces it supports on the selected board. IPI features like block automation, connection automation and IP customization shall also be demonstrated.
2. Using the Flow Navigator, select Create Block Design.

Notice how you can set Design Name, Directory, and source set in the **Create Block Design** dialog box. You can change or keep the default values and proceed. For this lab, choose design name as ‘top’.

|  |
| --- |
|  |

The Vivado IP Integrator displays a design canvas to let you quickly create complex

subsystem designs by integrating IP cores.

1. There are a few different ways to add IPs in the block design-

* By clicking the **Add IP** button in the block design canvas. 

Graphical user interface, table

Description automatically generated

* You can also right-click on the design canvas to open the context menu and select **Add IP**.
* You can also add an IP by dragging and dropping the IP from the IP catalog to the block design canvas. In this case, you can search for the IP, select it and drag-and-drop it on the block design canvas.

Let’s add **MicroBlaze** processor IP into our block design now.

1. In the search field of the IP catalog, type MICROBLAZE to find the IP.

|  |
| --- |
|  |

1. Select **MicroBlaze** and press **Enter** on the keyboard or double-click the core in the IP catalog. The MicroBlaze core is instantiated onto the IP Integrator design canvas.

Double click the MicroBlaze IP to re-customize it. In general settings, make sure to check the following options:

* Enable MicroBlaze Debug Module Interface
* Use Instruction and Data Caches

Leave all the other options to default settings and press OK.

|  |
| --- |
|  |

1. Click **Run Block Automation** in the banner at the top of the design canvas

|  |
| --- |
|  |

1. Select all the default options in the Run Block Automation dialog box and click OK. Make sure that clock connection is set to New Clocking Wizard, for this lab.

|  |
| --- |
| Graphical user interface, text, application, Word  Description automatically generated |

1. The IP Integrator adds local memory and debug to the processor block and connects a Clocking Wizard and Processor System Reset to the subsystem.
2. Click the Regenerate Layout button  to redraw the subsystem design. The optimized layout of the design should now look like the following figure:

|  |
| --- |
|  |

1. In the IP Integrator Flow of the Flow Navigator there are multiple tabs available like Sources, Design, Signals and Board. If you click on the ‘Board’ interface tab, it shows all the available interfaces on the KCU105 evaluation platform.

An interface is a grouping of signals that share a common function, containing both individual signals and multiple buses. By grouping these signals and buses into an interface, the Vivado IP Integrator can identify common interfaces and automatically make multiple connections in a single step. See the *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* ([UG994)](https://www.xilinx.com/cgi-bin/docs/rdoc?v=2020.2;d=ug994-vivado-ip-subsystems.pdf) for more information on interface pins and ports.

|  |
| --- |
|  |

1. You can drag and drop the interfaces required for the design from here directly to the design canvas. For this lab choose DDR4 SDRAM interface. You can choose different auto connect options for Interfaces depending on the design requirement. For this design choose ddr4\_sdram\_062.

|  |
| --- |
|  |

NOTE: Component mode can be changed later as well by using re-customization feature of the IP. You can re-customize any IP by double clicking on it in the design canvas, and doing modifications as needed, for the project. Additionally, we can also add the interface IP’s to the design canvas by **add ip** feature, described above for MicroBlaze IP.

1. The design canvas should now look like shown below:

|  |
| --- |
|  |

1. You can now see that there is an orange circle next to DDR4 SDRAM in the boards tab. This shows that Vivado is board aware and knows what interfaces (external memory in this case) in the design has already been used.

|  |
| --- |
|  |

1. As you can see that Microblaze and DDR4 are not connected to each other yet. IP Integrator offers the Designer Assistance feature to automate certain kinds of connections. For the current subsystem design, you can connect DDR4 SDRAM IP with the MicroBlaze using connection automation. Click **Run Connection Automation** in the banner at the top of the design canvas.

|  |
| --- |
|  |

The Run connection Automation dialog box opens.

1. Before you run connection automation, make sure that the IP’s are customized as per the design requirements. For example, in the current design we want the MicroBlaze clock frequency at 125 MHz and DDR4 frequency at 300 MHz. To do this, double click the CLOCKING WIZARD IP auto generated with the MICROBLAZE board automation and re-customize the CLK\_IN1 to sysclk125. Leave all the settings to default and click **OK.**

|  |
| --- |
|  |

1. Now click on Run Connection Automation, and the Automation window opens up. Select all the blocks you need for connection automation.

For this lab, make the following changes:

* reset of clk\_wiz\_1 is set to custom
* sys\_reset for ddr4\_0 is set to FPGA reset
* ext\_reset\_in of rst\_clk\_wiz\_1\_100M is set to FPGA reset

Leave all other connections to default settings, select all interfaces and click OK

The MicroBlaze and DDR4 IP’s with their respective interface, clock and reset connections is now ready on the design canvas.

|  |
| --- |
|  |

1. You will see that after running connection automation, the ‘Run Connection Automation’ banner is still active for M\_AXI\_DP port of Microblaze. This port has not been connected to any slave port so the Vivado design assistance will give an option to connect it automatically, if you run it then you will observe that a new slave port will be automatically created in the SmartConnect and will be connected to Microblaze master. But that is not the current design requirement, and that port has not been connected intentionally. You will see later that an AXI Interconnect IP will be instantiated in the design and that will become a slave for the M\_AXI\_DP port.

You must be mindful of the design connections you are making as per the requirements of your design. This will help you debug the design easily if the end design does not behave as expected.

1. Click the Regenerate Layout button if you need to better placement of the blocks on the canvas.  
     
   
2. The design canvas should look like the design shown below:

|  |
| --- |
|  |

1. It can be observed that after running connection automation, an AXI SmartConnect is automatically instantiated along with a processor system reset IP for DDR4 SDRAM. Since the design uses two different frequencies therefore Vivado is aware of it to minimize Clock Domain Crossing (CDC) issues, it auto instantiates the necessary IP’s as required by the design.

**IMPORTANT!** *IP Integrator treats an external reset coming into the block design as asynchronous to the clocks. You should always synchronize the external resets with a clock domain in the IP subsystem to help the design meet timing.*

1. You can use a Processor System Reset block (proc\_sys\_reset) to synchronize the reset. The Processing System Reset is a soft IP that handles numerous reset conditions at its input and generates appropriate system reset signals at its output; however, if a clock and a reset are external inputs to the block design, and the reset signal synchronizes externally to the clock, then you need to associate the related clock with the reset. This does not require the Processor System Reset block.

## Step 3: Adding New IP’s, Interfaces and IP Customization

1. You can add multiple IP’s from the Vivado IP catalogue as per the specific design requirements. For continuity of this lab, add the following IP’s from the IP catalogue, by using add IP feature on the design canvas. For a quick reference, check the above-mentioned steps (Step-2), used for adding Microblaze IP.

* AXI UARTLITE
* AXI GPIO (2 instances)
* AXI INTERCONNECT
* AXI BRAM CONTROLLER
* CONCAT
* CONSTANT (2 instances)
* AXI INTERRUPT CONTROLLER
* AXI IIC

1. After adding all the IPs, the IP Integrator should look like the snapshot below. The relative positions of the blocks placed on the canvas might be slightly different.

|  |
| --- |
|  |

1. Double-click the AXI Interconnect core to open the Re-Customize IP dialog box, as shown in the following figure:

|  |
| --- |
|  |

1. In the Top Level Settings, change Number of Master Interfaces field to **5** from the drop-down menu.
2. Leave all the remaining options as is and click **OK**.

The IP integrator re-customizes the AXI Interconnect, changing the number of master interfaces to five, as shown in the following figure:

|  |
| --- |
|  |

Now you can connect the five slave IP cores to the AXI Interconnect.

1. Double-click Concat IP to open the Re-Customize IP dialog box. Change number of ports to 4 from 2. Leave all the remaining options as is and click **OK.**

|  |
| --- |
|  |

1. Double-click axi\_gpio\_0 to open the Re-Customize IP dialog box. In the Board tab of Re-customize IP window choose the following:

GPIO (IP interface) -> rotary switch (Board Interface)

GPIO2(IP interface) -> push buttons 5bits (Board Interface)

Leave all the remaining options as is and click **OK.**

|  |
| --- |
|  |

1. Double-click axi\_gpio\_1 to open the Re-Customize IP dialog box. In the Board tab of Re-customize IP window choose the following:

GPIO (IP interface) -> led 8bits (Board Interface)

GPIO2(IP interface) -> dip switches 4bits (Board Interface)

Leave all the remaining options as is and click **OK.**

|  |
| --- |
|  |

1. Double-click AXI SMARTCONNECT IP which was auto-generated earlier to open the Re-Customize IP dialog box and change number of Master Interfaces to ‘2’. Leave all the remaining options as is and click **OK.**

|  |
| --- |
|  |

## Step 4: Creating Connections to all IP’s and interfaces

At this point, you have instantiated several AXI slaves that you can access through a master such as a processor. To connect to a master controlling these slaves, first let’s create connectivity between the AXI Interconnect and the instantiated IPs

1. Place the cursor on top of the M00\_AXI interface pin of the AXI Interconnect. Click and drag the cursor from the M00\_AXI interface pin to the S\_AXI interface port of AXI GPIO block.

***Note*:** The cursor changes into a pencil indicating that a connection can be made from that interface pin. Clicking the mouse button here starts a connection on the M00\_AXI interface pin. You must press and hold down the mouse button while dragging the connection from the M00\_AXI pin to the S\_AXI interface port.

|  |
| --- |
|  |

As you drag the connection wire, a green checkmark appears on the S\_AXI interface pin indicating that you can make a valid connection between these points. The Vivado IP Integrator highlights all possible connection points in the subsystem design as you interactively wire the pins and ports.

1. Release the mouse button and Vivado IP Integrator makes a connection between the M00\_AXI interface pin and the S\_AXI port, as shown in the following figure:

|  |
| --- |
|  |

1. Repeating the steps outlined above, connect the M01\_AXI and the M02\_AXI, M03\_AXI, M04\_AXI to the S\_AXI interface ports of AXI Interrupt Controller, AXI Uartlite, AXI IIC and AXI GPIO(axi\_gpio\_1).

Note that the order of connection between M\_AXI interfaces of the AXI Interconnect and S\_AXI interfaces of the slave IPs does not matter. The connections to the AXI SmartConnect should now appear as shown in the following figure:

|  |
| --- |
|  |

## Step 5: Connecting all IP’s and Interfaces using both manual and automated connections

1. On the design canvas you can see that there are few IP’s and Interfaces which are not connected, such as, UART interface of the AXI Uartlite, GPIO interface of the AXI GPIO’s, Concat, AXI BRAM Controller, and so on.

To connect the remaining interfaces, you can once again use the connection automation feature of IPI as used earlier to connect DDR4 SDRAM IP. The remaining unconnected IP’s can be connected manually as per the design requirement.

1. Run Connection Automation from the designer assistance by selecting all interfaces.

|  |
| --- |
|  |

1. After running the connection Automation, and regenerating layout, the design canvas should look like the figure below:

|  |
| --- |
|  |

1. All the external interfaces connect to I/O ports, and the BRAM Controller connects to the Block Memory Generator. You can right-click on the external ports (dip\_switches\_4bits and rs232\_uart etc.) and select the External Interface Properties command.

In the External Interface Properties window, you can change the name of the port if needed. The IP Integrator automatically assigns the name of the port when connection automation is run.

1. You can see on the design canvas that some IP’s are still not connected but apart from that all the interfaces have been connected according to their customizations.

Now we can make the following connections manually

*dout[0:0] port of CONSTANT(xlconstant\_0) to In3[0:0] port of CONCAT  
 dout[0:0] port of CONSTANT(xlconstant\_1) to In2[0:0] port of CONCAT*

*interrupt port of AXI UARTLITE to In1[0:0] port of CONCAT  
 iic2intc\_irpt port of AXI IIC to In0[0:0] port of CONCAT*

*dout[3:0] port of CONCAT block to intr[0:0] port of AXI INTERRUPT CONTROLLER*

*interrupt port of INTERRUPT CONTROLLER to Interrupt pin of MICROBLAZE*

To automate these manual connections, you can run the following TCL commands on the TCL console to make the connections:

|  |
| --- |
| connect\_bd\_net [get\_bd\_pins xlconstant\_0/dout] [get\_bd\_pins xlconcat\_0/In3]  connect\_bd\_net [get\_bd\_pins xlconstant\_1/dout] [get\_bd\_pins xlconcat\_0/In2]  connect\_bd\_net [get\_bd\_pins axi\_uartlite\_0/interrupt] [get\_bd\_pins xlconcat\_0/In1]  connect\_bd\_net [get\_bd\_pins axi\_iic\_0/iic2intc\_irpt] [get\_bd\_pins xlconcat\_0/In0]  connect\_bd\_net [get\_bd\_pins xlconcat\_0/dout] [get\_bd\_pins axi\_intc\_0/intr]  connect\_bd\_intf\_net [get\_bd\_intf\_pins axi\_intc\_0/interrupt] [get\_bd\_intf\_pins microblaze\_0/INTERRUPT] |

1. After making all the connections the final design should look something like the figure shown below. You can use regenerate layout for a cleaner design canvas.

|  |
| --- |
|  |

1. Click the **File → Save Block Design** command from the main menu.
2. From the menu at the top of the IPI design canvas, run the IP subsystem design rule checks

|  |
| --- |
|  |

(DRCs) by clicking the **Validate Design** button

1. The Validate Design dialog box opens, and validation should be successful. Click **OK**.

|  |
| --- |
|  |

## Step 6: Using the Address Editor

For various memory mapped master and slave interfaces, IP Integrator follows the industry

standard IP-XACT data format for capturing memory requirements and capabilities of endpoint

masters and slaves. This section provides an overview of how IP Integrator models address

information on a memory-mapped slave.

Master interfaces have address spaces, or address\_space objects. Slave interfaces have an

address\_space container called a memory map to map the slave to the address space of the

associated master. Typically, these memory maps are named after the slave interface pins, for example S\_AXI, though that is not required.

The memory map for each slave interface pin contains address segments, or address\_segment objects. These address segments correspond to the address decode window for that slave. A typical AXI4-Lite slave will have only one address segment, representing a range of addresses. However, some slaves, like a bridge, will have multiple address segments or a range of addresses for each address decode window.

When you map a slave to the master address space, a master address\_segment 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. In the Address Editor, you see the address segments of the slaves, and can map them to address spaces in the masters.

1. Click the **Address Editor** tab to show the memory map of all the slaves in the design.

***Note*:** If the Address Editor tab is not visible then select **Window →  Address Editor** from the main menu.

The IP Integrator has automatically assigned the addresses.

Note that there are multiple address networks shared between processors accessing the peripherals (AXI BRAM, GPIO etc.), and networks for local memory belonging to each processor subsystem.

You can change the automatic address assignments by clicking in the corresponding column and changing the values.

|  |
| --- |
|  |

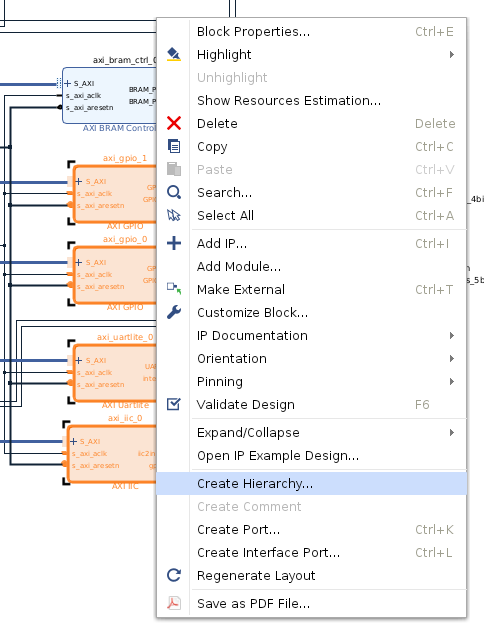
1. Select the **Diagram** tab, to return to the IP Integrator design canvas.

## Step 7: Creating Hierarchies

1. There are different ways of changing the view on the canvas and better organizing the blocks. One of these capabilities is creating hierarchy levels to include one or more blocks.

For this lab you will create 3 hierarchies called peripherals, memory\_ss and microblaze\_ss.

1. To create the ***peripherals*** hierarchy select the following 4 blocks- axi\_iic\_0 (IIC),axi\_uartlite\_0(UARTLITE),axi\_gpio\_0(GPIO),axi\_gpio\_1(GPIO) by holding down the ctrl- button clicking on them one after the other. They should be highlighted in orange.
2. Now, right-click and select Create Hierarchy.



1. You can assign a new name to the hierarchy as “peripherals”.
2. You can expand the hierarchy to see the content inside by clicking on the + button on the top left of the block.

|  |  |
| --- | --- |
|  |  |

1. Similarly, to create the ***memory\_ss*** hierarchy select the following 4 blocks- axi\_smc (SmartConnect), ddr4\_0(DDR4 SDRAM), axi\_bram\_ctrl\_0 (BRAM Controller), axi\_bram\_ctrl\_0\_bram (Block Memory Generator) by holding down the ctrl- button clicking on them one after the other.
2. And lastly, to create the ***microblaze\_ss*** hierarchy select the following 7 blocks – microblaze\_0(Microblaze), axi\_intc\_0(Interrupt Controller), microblaze\_0\_local\_memory,mdm\_1(MDM),xlconstant\_0(Constant), xlconstant\_1(Constant), xlconcat\_0(Concat)

You might get a critical warning when trying to create a MicroBlaze subsystem as shown in figure below, but it can be safely ignored for this design. Elf file is the executable. It's generated after compilation & build process in Software Development Kit (SDK). Since this design won’t be using the SDK therefore it is safe to ignore the warning.

|  |
| --- |
|  |

1. The design canvas should look something like the snapshot shown below.

|  |
| --- |
|  |

1. You can also switch to interface view and try other options for better clarity on the design canvas.

|  |
| --- |
|  |

1. Click on Validate Design and click OK once the design is Validated successfully.
2. Click on **File → Save Block Design** command from the main menu.

## Step 8: Block Design Containers

Block Design Containers (BDC) is a feature in IPI which allows you to instantiate a Block Design (BD) inside another BD. We call the BD that gets instantiated as the **childBD**and the BD in which the childBD is instantiated as the **topBD**. The following features are supported:

* **Full visibility into the contents of the childBD from the topBD**: The instantiated childBD looks like a regular hierarchy in the topBD, where clicking on the '+' icon on the block will expand the childBD in-place and allow you to view the contents.
* **Ability to change addresses from the topBD**: Slaves assigned to certain addresses in the childBD can be changed directly from the topBD, and can be different from the original assignments. This allows multiple instantiations of the same childBD to map to different Masters at different addresses.
* **Ability to propagate parameters inside childBD**: You can propagate parameters from the topBD inside the childBD. This also allows for different set of parameters to propagate to different instantiations of the same childBD.
* **Changes made to childBD and reflected into topBD:**Any changes made to the master copy of the childBD pops up a Stale banner in the topBD which will then update all block containers to the changes made.
* **Ability to specify multiple Synthesis sources**: This allows you to add multiple implementations (versions) for the same piece of logic to be specified, all in one block.
* **Create partition definitions and Reconfigurable Modules(RMs)**: A block container can be made into a Partially Reconfigurable (PR) module, which means that when the design is being generated, a Partition Definition is created for the block container and all "variants" are converted into RMs.

Block Design Containers support both Top – Down flow and Bottoms Up flow. In Top- Down flow a diagram for the top-level design is created first. The user/designer them creates proper hierarchies in the top-level block design to push a set of IP blocks into BDC’s as sub-blocks. In this step, you will implement a Top Down flow where you will convert a regular hierarchy in top BD to a BDC cell. In the next step you will implement the bottom-up flow.

1. To create a block design container from the ‘peripherals’ hierarchy Right-click on the 'peripherals' hierarchy and select "Create block container". Ensure that the design is validated before creating the BDC.

|  |
| --- |
|  |

1. You will notice that a new block design has been created for the selected hierarchy. Also, the hierarchy block in the topBD has been replaced with a block design container.

|  |
| --- |
|  |

1. If you go to sources and double click on ‘peripherals.bd’, it should look like screenshot below:

|  |
| --- |
|  |

1. If you switch back to the top.bd by double clicking on ‘top.bd’ in the sources tab and click on the '+' icon, you should see contents of 'peripherals.bd' (NOTE: This looks similar to the hierarchy block we had previously, but in fact it is now showing the contents of the new peripherals.bd). Click on the '-' icon to collapse the hierarchy and reduce clutter:

|  |
| --- |
|  |

1. **Adding a second variant:**We will now add a second variant to this container. To do this, switch back to 'peripherals.bd', click on File → Save Block Design As ... → Type in a name for the new variant. For example 'peripherals\_2' → Click OK.

|  |
| --- |
|  |

1. For this Lab, you can choose the name peripheral\_2

|  |
| --- |
|  |

**NOTE**: Users are allowed to use whatever BD they choose to as a potential Synthesis source. However, when a block is marked as a PR module, the tool requires that the boundaries of all the sources are matching EXACTLY.

1. Add this variant as a Synthesis source to the block container. To do this, switch back to the top.bd, and double-click on the block 'peripherals' (on design canvas).

|  |
| --- |
|  |

1. You will see the following options in the GUI:
   1. The '**Enable Dynamic Function eXchange on this module**' checkbox specifies if this container is a PR module or not.
   2. The '**Freeze the boundary of this container**' checkbox prevents any parameters from the topBD from entering inside this container.  
      This is useful when you have certain IPs configured at a particular frequency, for example, and want to maintain that.
   3. The two tables below this allow you to specify multiple Synthesis and Simulation sources for this container.
2. Enable the '**Enable Dynamic Function eXchange on this module**' checkbox. Then, click on the '+' icon near the 'Synthesis sources' table, and add the newly created 'peripherals\_2.bd':

|  |
| --- |
|  |

1. Validate and save the design
2. Click on 'Generate Block Design' on the left panel and select 'On local host' in 'Run Settings'. You can also choose to run it on a cluster if you have the configurations defined correctly. Click on **Generate**.

|  |
| --- |
|  |

1. If you switch to the 'Design Runs' tab, you should see something like this:

|  |
| --- |
|  |

**NOTE:**This window indicates that a Synthesis run has been created for the two variants inside the container, and Synthesis runs have been started for IPs inside the top BD. Since this container is in PR mode, the runs of the variants are deferred until Synthesis starts. At that point, the runs for the variants will be launched Out-of-Context and stitched at run-time.  
**NOTE:**The completion of this step might take some time.

1. Create a top-level wrapper for the top.bd: Right-click on 'top' in the Sources view and select "Create HDL wrapper". Click OK.

|  |
| --- |
|  |

In the ‘Create HDL Wrapper Window’ select ‘Let Vivado manage wrapper and auto-update’ and click OK.

1. Setup Dynamic Function Exchange (DFX) configurations: This step is essential because you want to associate your Partition Definitions to certain configurations. This allows your Reconfigurable Modules (RMs) to synthesize differently than other Block Designs (BDs).
2. Click on 'Dynamic Function eXchange Wizard' in the Flow Navigator. Click Next

|  |
| --- |
|  |

1. This window shows you a list of Partition Definitions in your design and RMs inside each of them. You are allowed to modify/add/delete RMs as you'd like. In this tutorial we will not concentrate on those, Click Next.

|  |
| --- |
|  |

1. This window allows you to attach custom configurations to you RMs. Click on 'automatically create configurations' → Next.

|  |
| --- |
|  |

1. The next window allows you to attach custom runs to your RMs. Click on 'automatically create configuration run’.

|  |
| --- |
|  |

1. Click on Next → Finish.
2. Now, you have 3 options to move forward:

* Use the Run Synthesis command to run only synthesis.
* Use the Run Implementation command, which will first run synthesis if it has not been run and then run implementation.
* Use the Generate Bitstream command, which will first run synthesis, then run implementation if they have not been run, and then write the bitstream for programming the Xilinx device.

These options can be selected from the Flow Navigator.

1. For this lab, we are going to generate the bitstream for the design.
2. Before doing this step, you should add the constraint file to the design. You can create a constraint file as required by your design. For this lab, we will use the constraint file “top.xdc” already provided in ‘BDC\_top\_bottom\_ref\_files’ folder. To add the constraint file to current working directory, select add sources from flow Navigator, choose ‘add or create constraints’ option in the add sources wizard. Add the constraint files to the current directory and click Finish. You will see ‘top.xdc’ file added in constraints folder, under design sources tab.

|  |
| --- |
|  |

1. From the Flow Navigator, click on **Generate Bitstream,** which will automatically synthesize, implement, and generate the bitstream for the design.
2. You now have a fully generated PR design using Block Design Containers using Top -   
   Down flow.

# Design Creation Steps: Bottom - Up BDC flow

1. In this lab you will get familiar with Bottom UP design flow of IP Integrator using TCL flow, instead of creating the design completely using the GUI mode from IPI.
2. In this flow, sub-block designs have been created separately and the user needs to instantiate these sub-blocks as BDC’s within the top-level block design.
3. Vivado offers the flexibility to populate and connect the IP’s and Interfaces on design canvas using either the GUI flow, TCL flow or a hybrid flow, where few steps can be completed using either flow. In step 8 of this tutorial, it could be observed that for every activity performed on the Design Canvas, be it adding IP, making connections, validation et al, there was a corresponding TCL command generated by Vivado which could be seen on the TCL console.

The entire design can be recreated using this TCL flow if you generate a corresponding TCL script of the design using either the “write\_bd\_tcl” command or by using the Vivado GUI as shown in snapshot below:

|  |
| --- |
|  |

Additionally, Vivado also saves journal files automatically, which can be used to regenerate few steps of the already created design.

1. You will create the bottoms up BDC in this step using the TCL flow. The TCL script for top block design, peripheral block design and the constraints files are already generated for this section of the tutorial to demonstrate the specific feature of the BDC using same process as mentioned in point 2. Put the top\_bd.tcl and peripherals.bd.tcl , provided with this tutorial in the same folder.
2. For bottoms up BDC flow, Create project and top level design by running ‘top\_bd.tcl’ in Vivado (source top\_bd.tcl), and you will see the following design populated on the design canvas with a top.bd source pre created.

|  |
| --- |
|  |

1. Create a sub design, source peripherals.bd in the project using tcl console (source peripherals.bd.tcl)

|  |
| --- |
|  |

1. Drag and Drop peripherals.bd from Sources/Hierarchy window onto the top.bd block design.

|  |
| --- |
| A picture containing graphical user interface  Description automatically generated |

1. Connect BDC cell peripherals in top design (Run Connection Automation or connect manually)
2. Validate and save the design.
3. Follow steps #5 to #21 from the above section Top Down Flow and you can take the design to bitstream generation

This brings us to the end of the tutorial.