





# SDSoC Platform Creation Labs

---

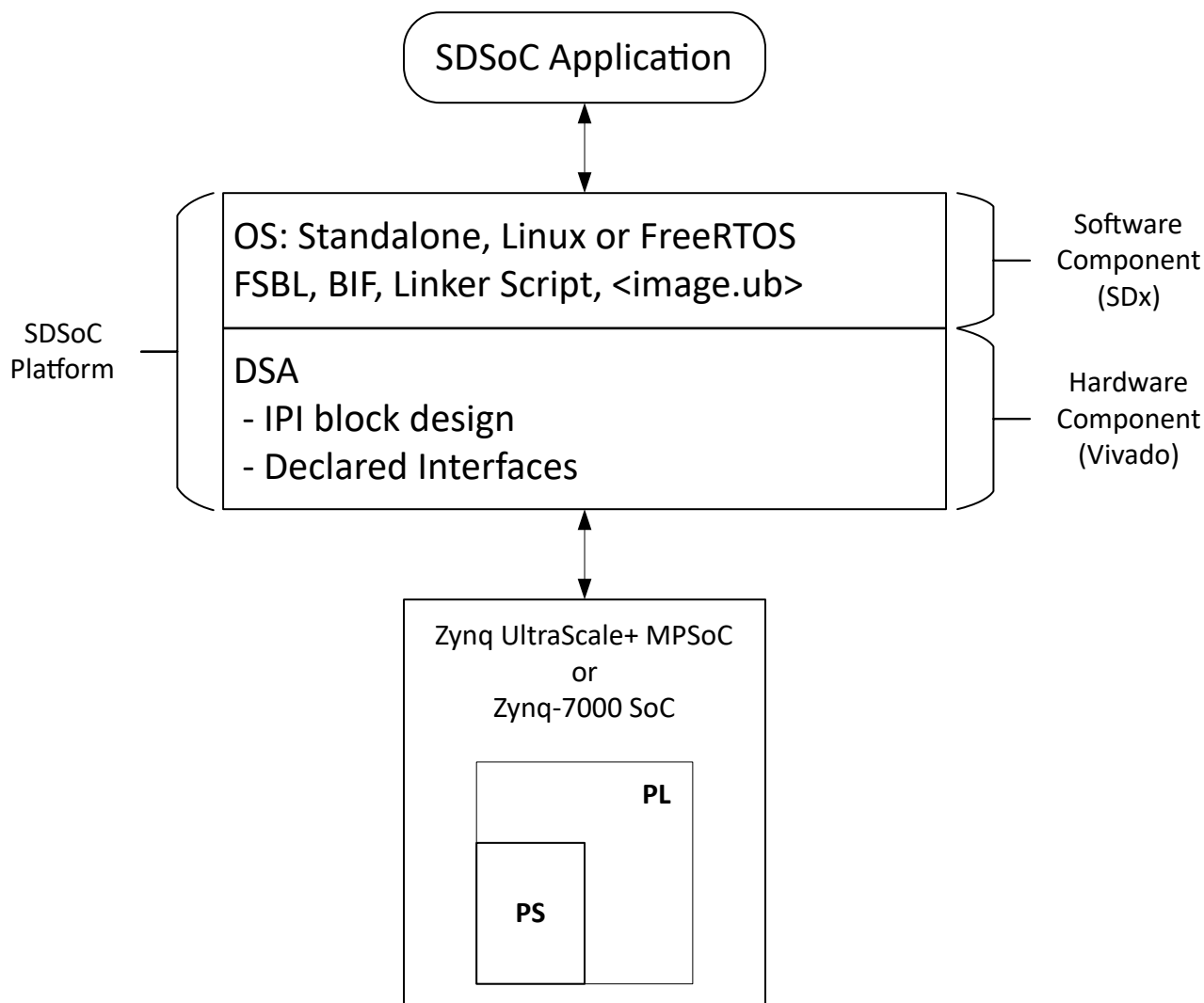
Introduction [Lab1: Creating the DSA for a Zynq UltraScale+ MPSoC Processor Design](#) [Lab 2: Creating the SDSoC Platform](#) [Lab 3: Using Your Custom Platform](#)

## Introduction

The SDSoC™ development environment provides the tools necessary for implementing heterogeneous embedded systems using the Zynq®-7000 SoC or the Zynq UltraScale+™ MPSoC devices.

This tutorial shows how to create an SDSoC platform on which an example SDSoC application is created and run. The Vivado® Design Suite is used to create the hardware platform and the SDx™ Integrated Design Environment (IDE) is used to build the software platform including an SDSoC application.

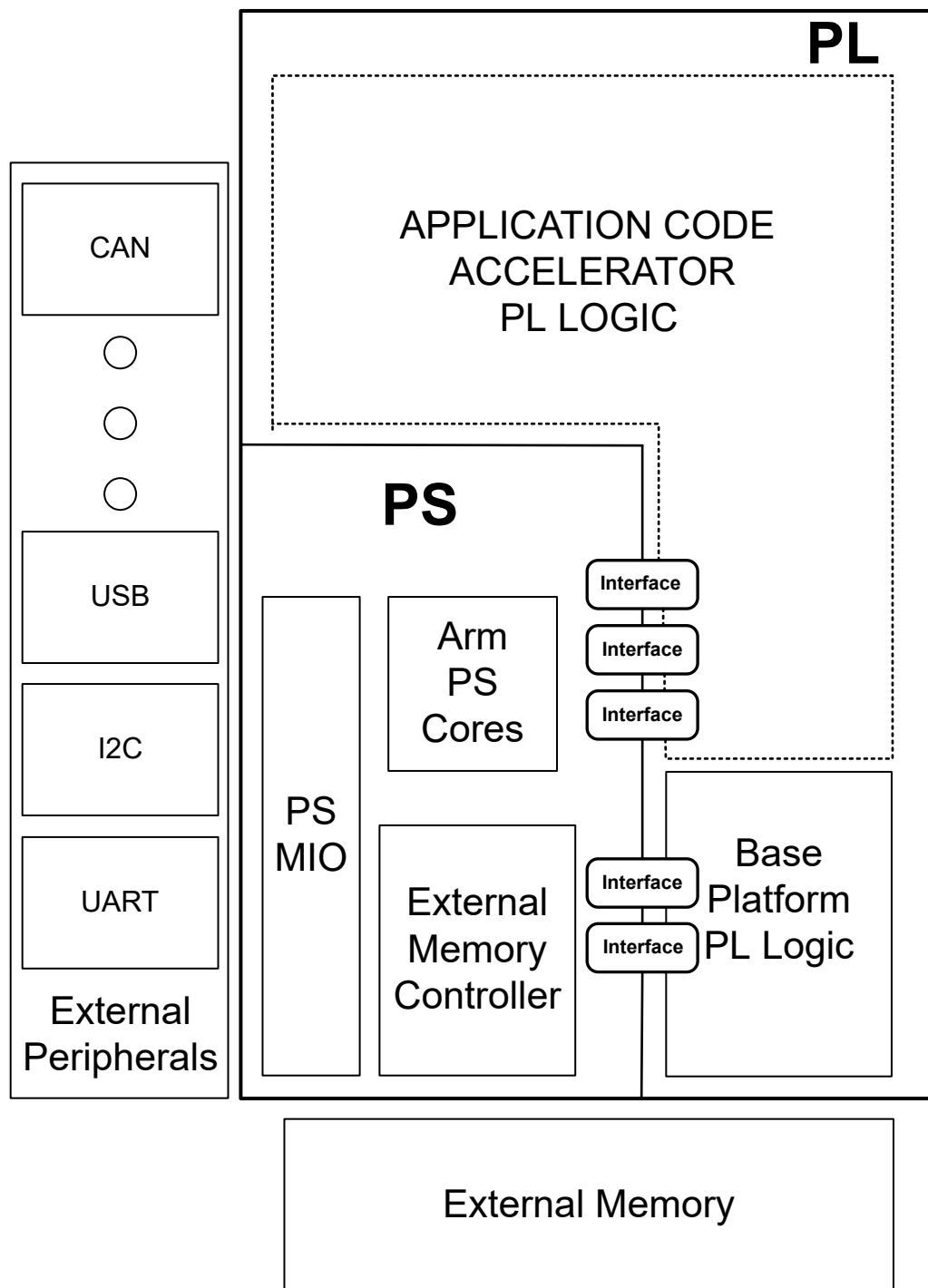
The concept of a platform is integral to the SDSoC environment and it defines the hardware, software, and meta-data components that SDSoC applications build upon. An SDSoC platform defines a base hardware and software architecture along with an application context, including the processing system, external memory interfaces, custom input/output, and software runtime. The software runtime may include an operating system (possibly “bare metal”), boot loaders, drivers for platform peripherals and a root file system. Every project you create within the SDx environment targets a specific hardware platform and utilizes tools such as the sds++/sdsc (referred to as sds++) system compiler to customize the platform’s application-specific hardware accelerators and data motion networks.



In this tutorial, you will be guided through three labs that target a Zynq UltraScale+ MPSoC-based ZCU102 board operating in a standalone or bare metal software runtime environment. Step-by-step instructions are provided on how to build the hardware and software components that constitute a platform:

1. Create the hardware component known as the Device Support Archive (DSA) for a platform.
  1. The DSA contains the IP integrator hardware design and the design meta-data properties.
2. Create the SDSoC platform by defining the software components, and matching them to the hardware DSA.
  1. Use the SDx IDE and its four (4) step process to create a custom platform.
3. Create and run an SDSoC hardware accelerated application on the new platform.

Platforms supported by the Zynq family of devices combine an Arm-based processor system (PS) with high performance programmable logic (PL) to develop products that can be optimized by partitioning the system design across hardware and software. The figure below shows the high-level Zynq device features on which platforms are created.



Zynq-based platforms can utilize the PS Arm processor cores, the integrated PS I/O (MIO) peripheral blocks, the PS external DDR memory controller, the multitude of configurable PS to PL and PL to PS interfaces, as well as logic blocks designed for the PL.

For more information on the SDSoC environment, see the *SDSoC Environment User Guide* ([UG1027](#)) and the *SDSoC Environment Platform Development Guide* ([UG1146](#)).

## ► Hardware and Software Requirements

This tutorial requires that you have the 2018.3 SDx tools installed. See [UG1294](#) for installation instructions, release notes, and licensing. All SDx environments include the Vivado Design Suite for programming the target devices and for developing custom hardware platforms. Although all the build steps can be accomplished without a target board, a ZCU102 board is required for testing on hardware.

Following the instructions of this platform creation tutorial generates a ZCU102 SDSoC platform for standalone target applications. For information on creating applications that run within a Linux operating system target environment, see the SDSoC Environment Platform Development Guide ([UG1146](#)). Linux target applications must be built on a Linux host machine, whereas standalone target applications can be built on either a Windows or a Linux host machine.

The following is a list of minimum software and hardware requirements for this tutorial:

- [SDSoC 2018.3 Development Environment](#)
  - [UG1294](#): Release Notes, Installation, and Licensing Guide
- [ZCU102 Evaluation board](#)
  - FAT32 formatted SD card
  - USB Host Type-A to mini-USB cable for ZCU102 USB to UART interface
  - Host PC USB to UART driver for Silicon Labs CP210x
    - See [XTP426](#), the Silicon Labs CP210x USB-to-UART Installation Guide.

## Related information

- [Lab1: Creating the DSA for a Zynq UltraScale+ MPSoC Processor Design](#)
- [Lab 2: Creating the SDSoC Platform](#)
- [Lab 3: Using Your Custom Platform](#)

## Lab1: Creating the DSA for a Zynq UltraScale+ MPSoC Processor Design

**NOTE:** This lab requires familiarity with the Vivado Design Suite and IP Integrator feature of the tool. If you are not familiar with the Vivado Design Suite, see the *Vivado Design Suite User Guide: Getting Started* ([UG910](#)).

In this lab you will use the Vivado® Design Suite to create a Zynq® UltraScale+ MPSoC processor design that uses the Arm® dual-core Cortex™-A53 and a dual-core Cortex-R5 real-time processing system (PS) as well as IP blocks in the programmable logic (PL) region. The Clocking Wizard IP is used to generate several clocks for possible use by the accelerated hardware functions. Processor System Reset IP blocks will be added to synchronize resets of the various clock sources. A Concat IP that combines the source of all interrupts from the IP blocks in the programming logic to the PS is also included. Select hardware interfaces will be declared for use by the sds++ system compiler for attaching hardware accelerators. The hardware design and its interface meta-data will then be encapsulated in a Device Support Archive (DSA) file for later use in creating a custom SDSoC platform (see [Lab 2: Creating the SDSoC Platform](#)).

### ► Step 1: Start the Vivado IDE and Create a Project

#### On a Linux host machine:

At the shell prompt, type the following commands:

1. ``source <Xilinx_Install_Directory>/SDx/<Version>/settings64.{sh,csh}``
2. ``vivado``

The first command sets the environment variables before launching Vivado and the second command launches the Vivado IDE.

### On a Windows host machine:

For a Windows host machine, use one of the following methods to launch Vivado:

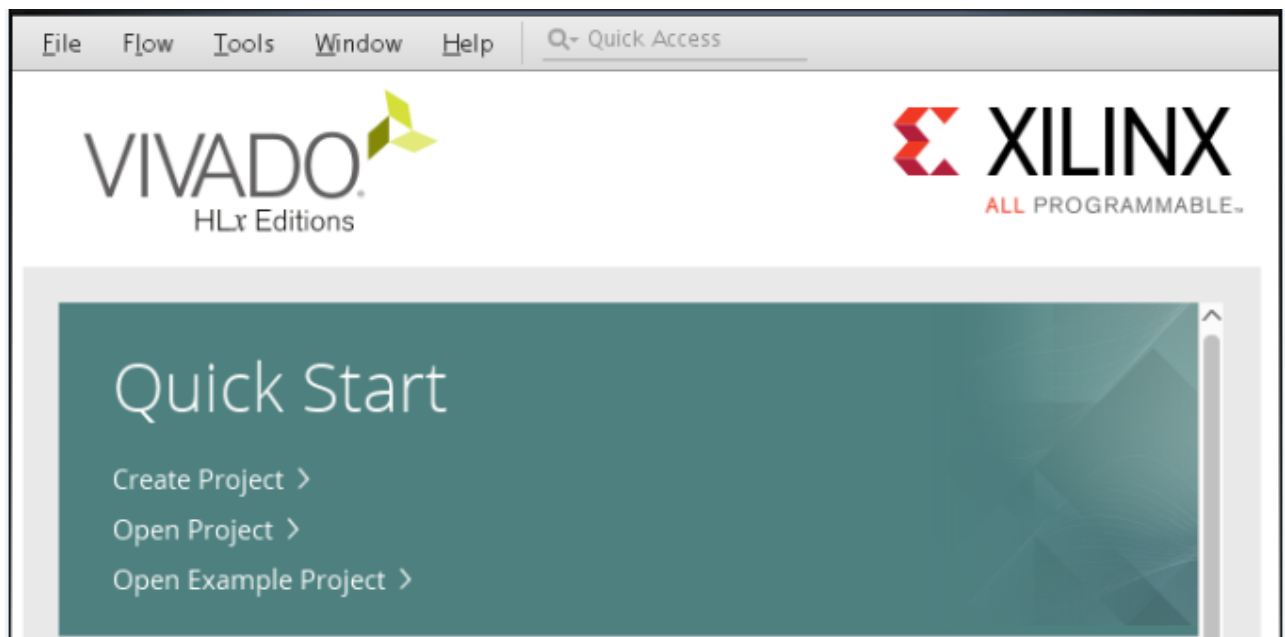
- Click the Vivado desktop icon.
- From the Start menu, select Xilinx Design Tools \> Vivado 2018.3 \> Vivado 2018.3.
- From a Command prompt window, type the following commands:
  1. ``<Xilinx_Install_Directory>/SDx/<Version>/settings64.bat``
  2. ``vivado``

The first command sets the environment variables before launching Vivado and the second command launches the Vivado IDE.

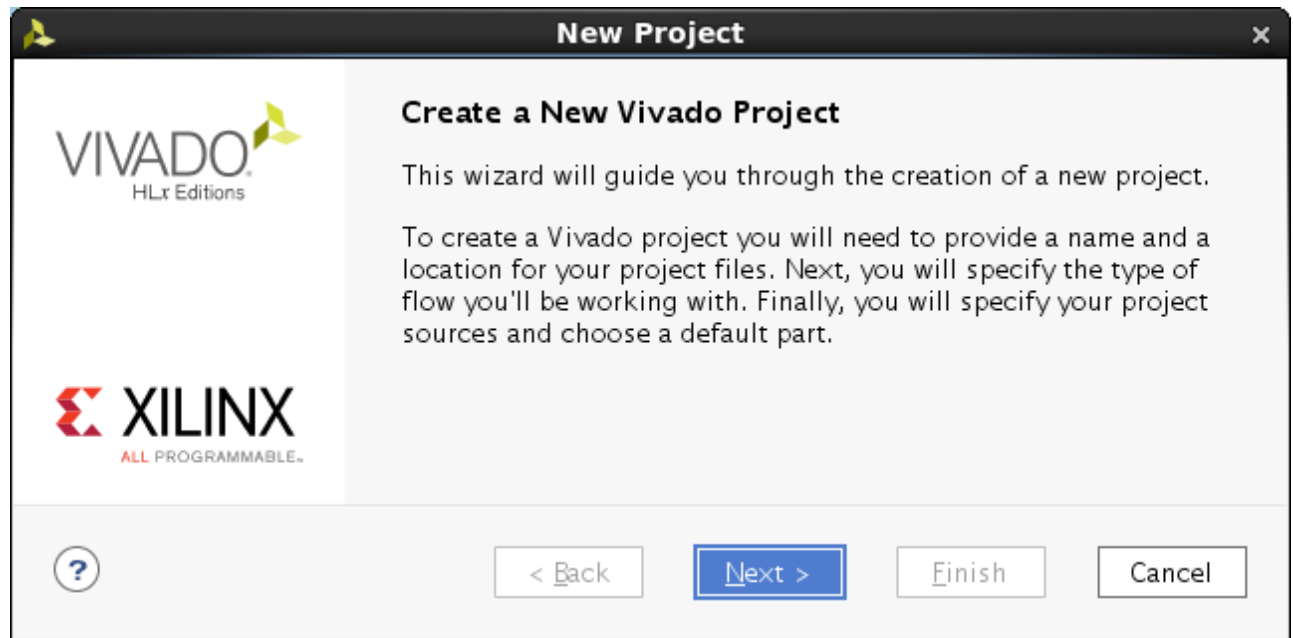
### Creating a Vivado project

Follow these steps to create a Vivado project:

1. From the **Quick Start** section, click **Create Project**, as shown in the following figure.

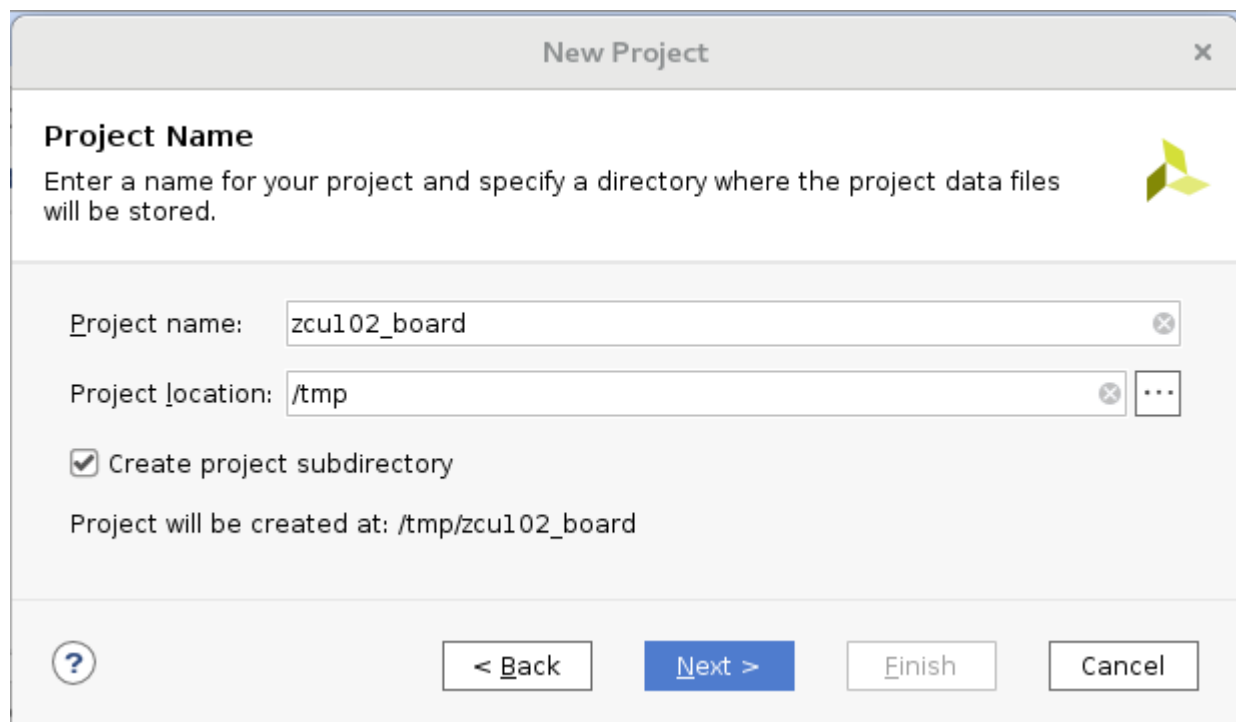


2. The New Project Wizard opens. Click **Next**.

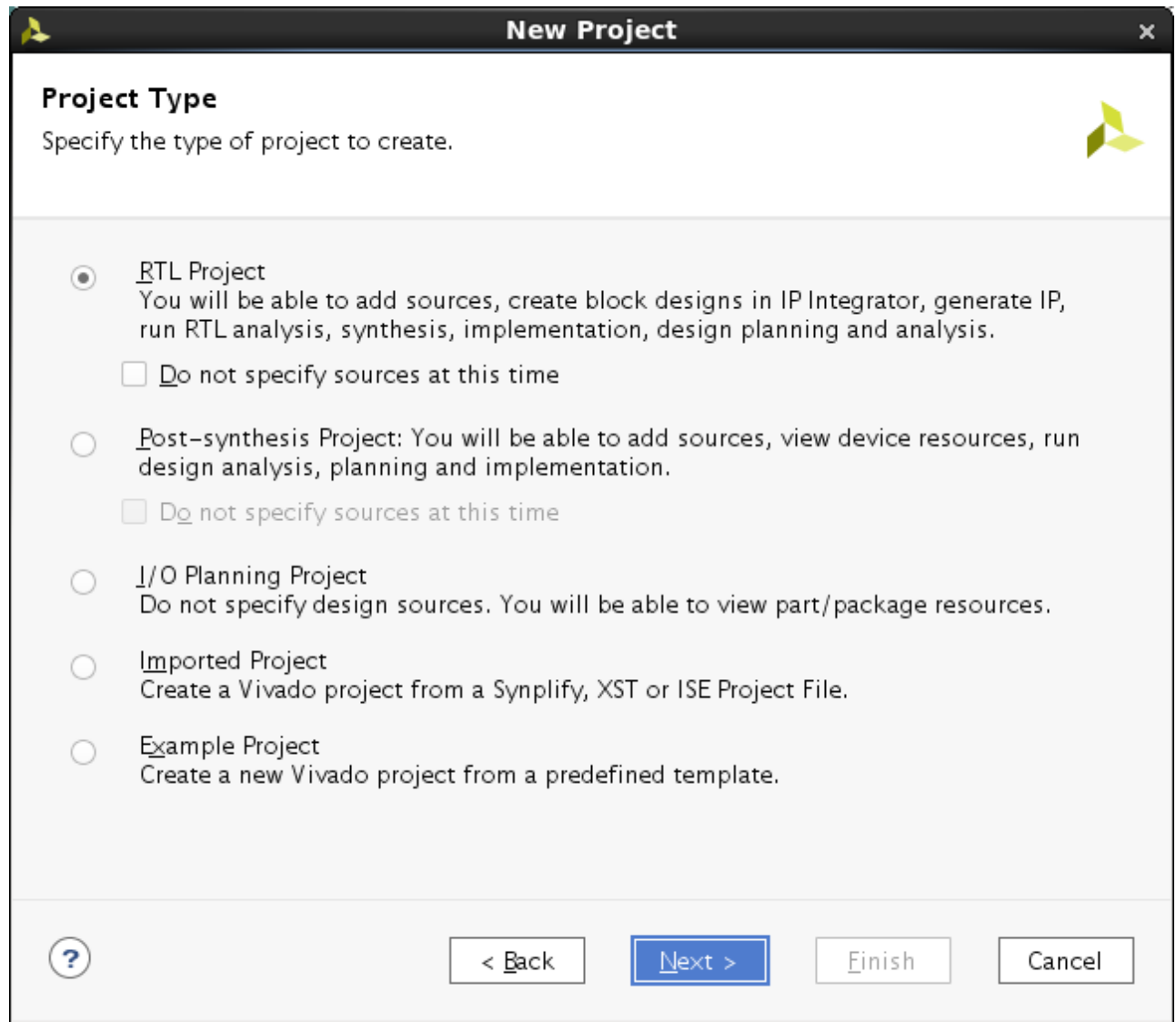


The **Project Name** dialog box opens.

3. Enter **zcu102\_board** in the Project name text box.
4. Enter **/tmp** in the Project Location text box.
5. Select the **Create project subdirectory** check box.

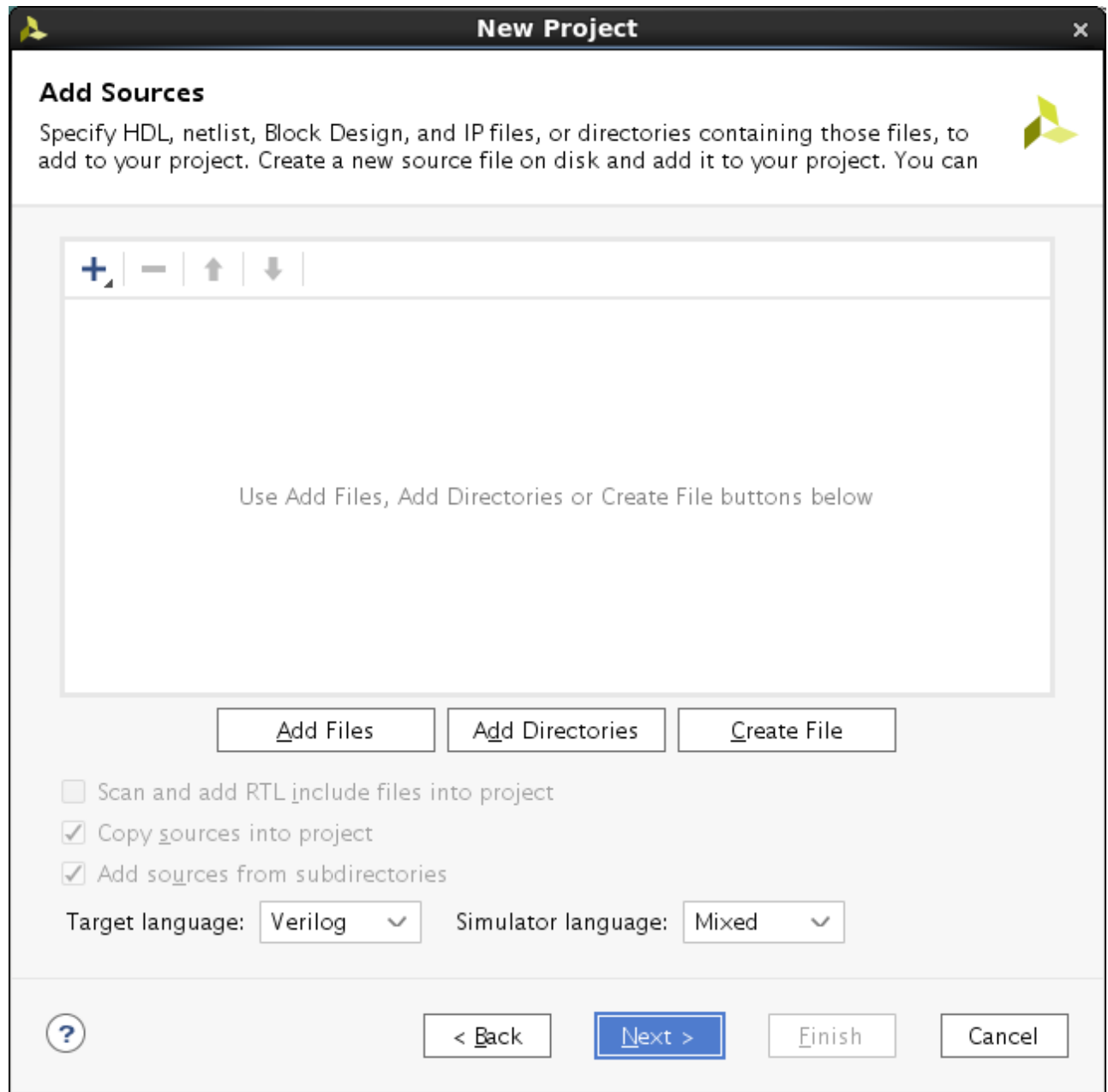


6. Click **Next**. The **Project Type** dialog box appears.
7. Select **RTL Project**.

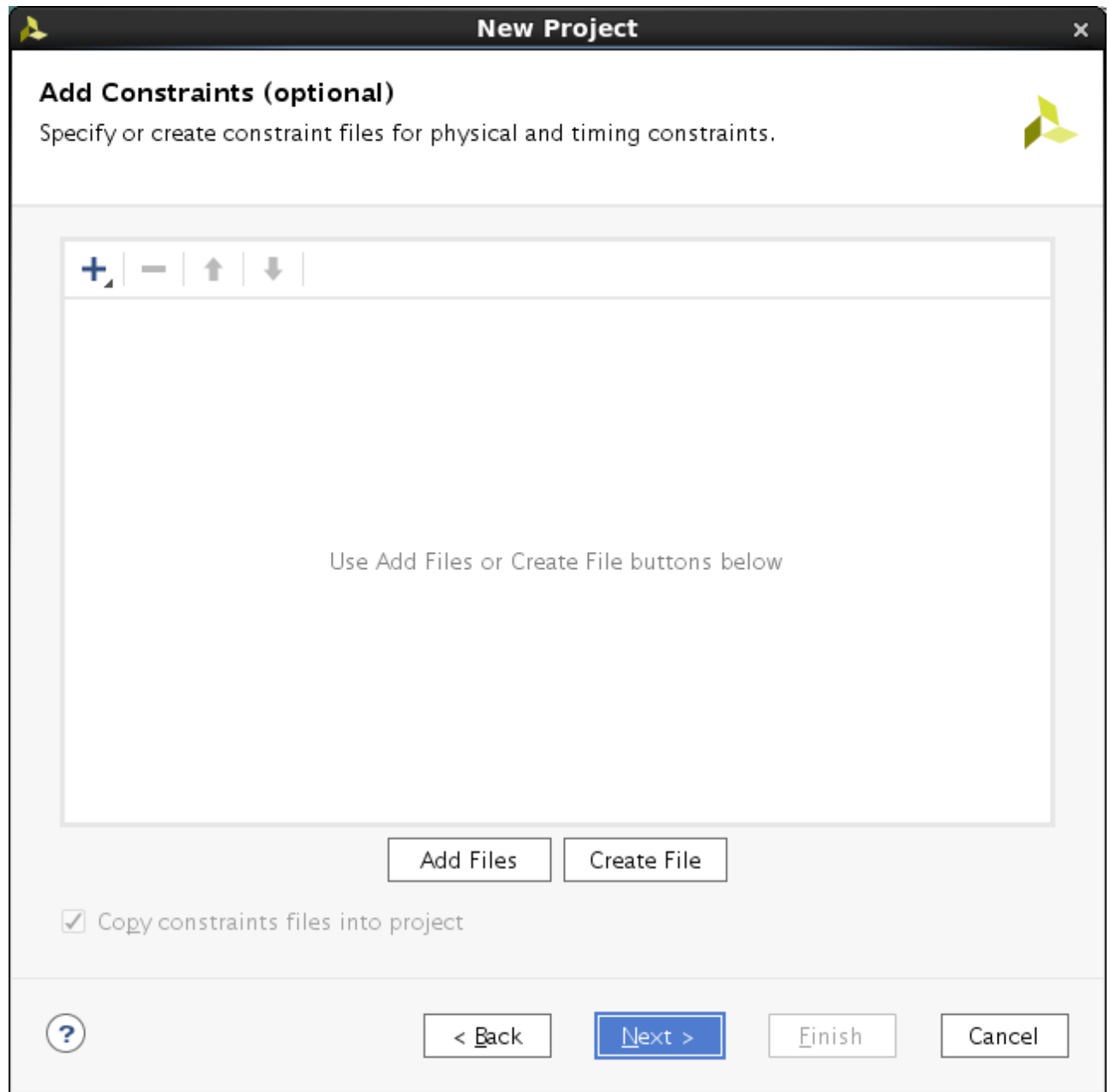


- Click **Next**. The **Add Sources** dialog box appears.
- Select **Verilog** from the Target language dropdown list.
- Select **Mixed** from the Simulator language dropdown list.





11. Click **Next**. The **Add Constraints** dialog box appears.



12. Click **Next**. The **Default Part** dialog box appears.

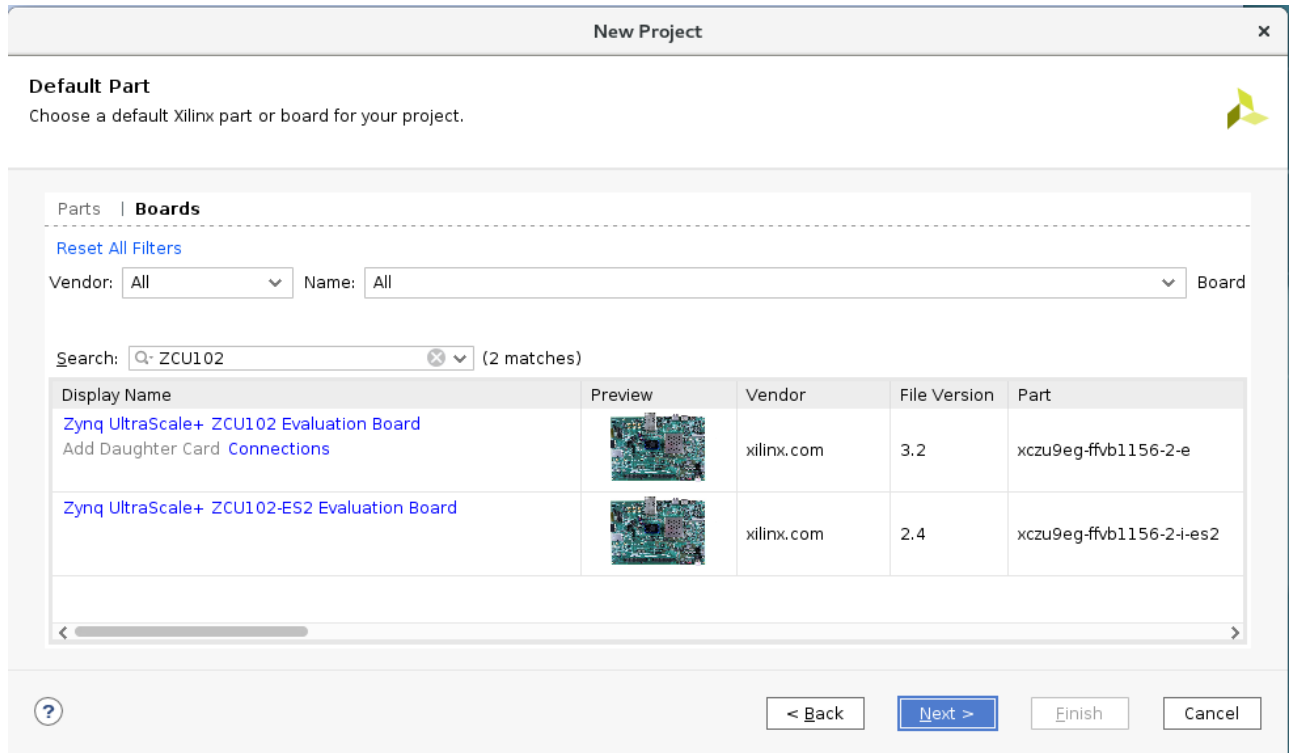
13. Click the **Boards** tab at the top of the dialog box.

For this tutorial, you will be using the ZCU102 evaluation board as a template for the platform that you are creating. Selecting an existing board provides the different parts and interfaces you will use in creating your custom platform.

14. Select **Zynq UltraScale+ ZCU102 Evaluation Board** from the available boards list.



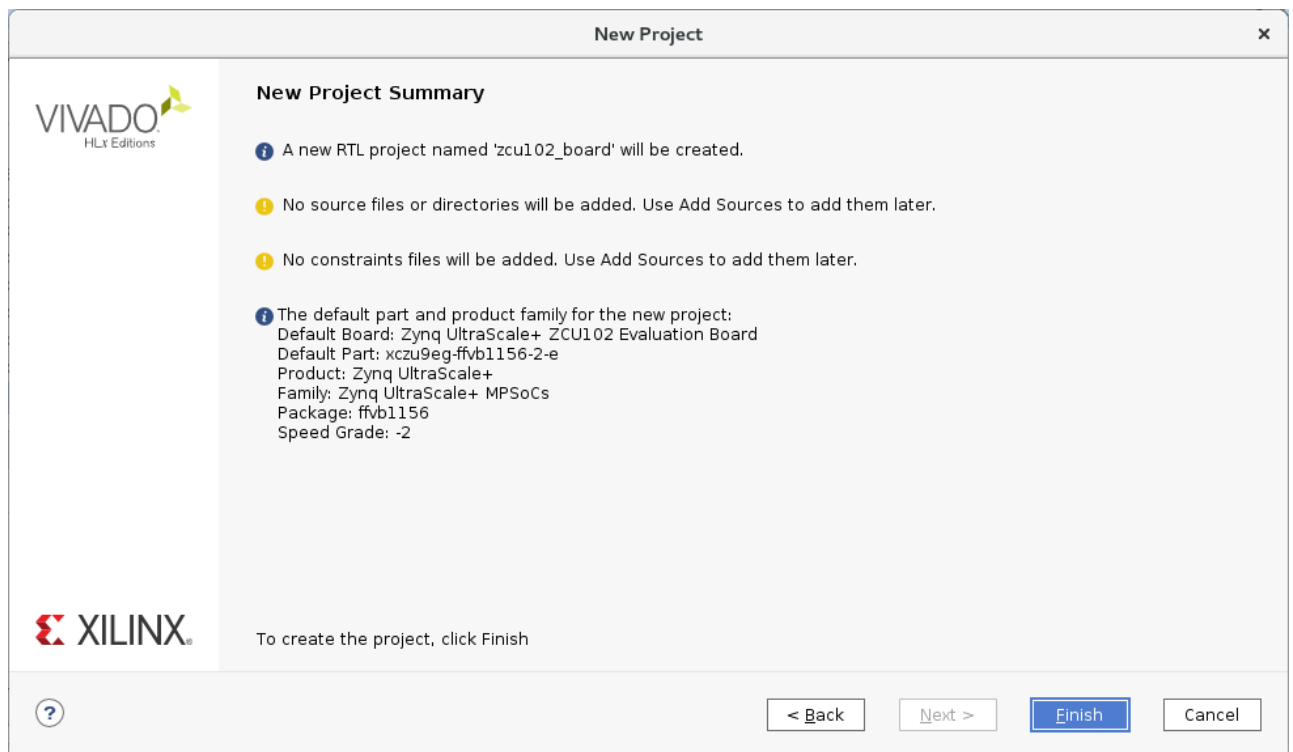
**TIP** You can use the Search feature to filter for ZCU102.



15. Click **Next**.

**WARNING** Multiple versions of each board are supported in Vivado. Ensure that you are targeting the design to the right hardware.

16. Review the project summary on the **New Project Summary** page, and then click **Finish** to create the project.




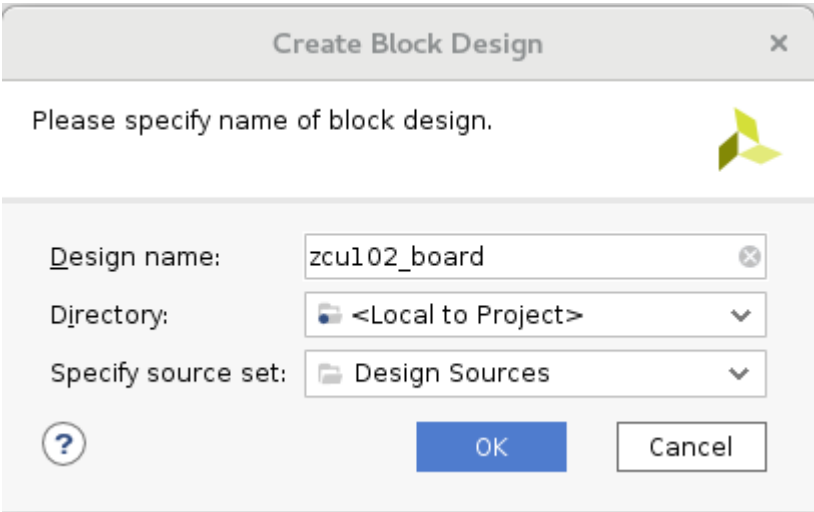
## ► Step 2: Create an IP Integrator Design


1. In the **Flow Navigator** view, expand **IP INTEGRATOR** and select **Create Block Design**. The **Create Block Design** dialog box appears.



2. Specify a name for your IP subsystem design. For this example, use **zcu102\_board**.

 **NOTE:** Do not change the default values in the **Directory** and the **Specify source set** dropdown lists.



 **TIP** If the Vivado project contains multiple block designs, the IP integrator block design containing the SDSoC platform must have the same name as the SDSoC platform.

3. Click **OK**.

**Adding IP to the Block Design**

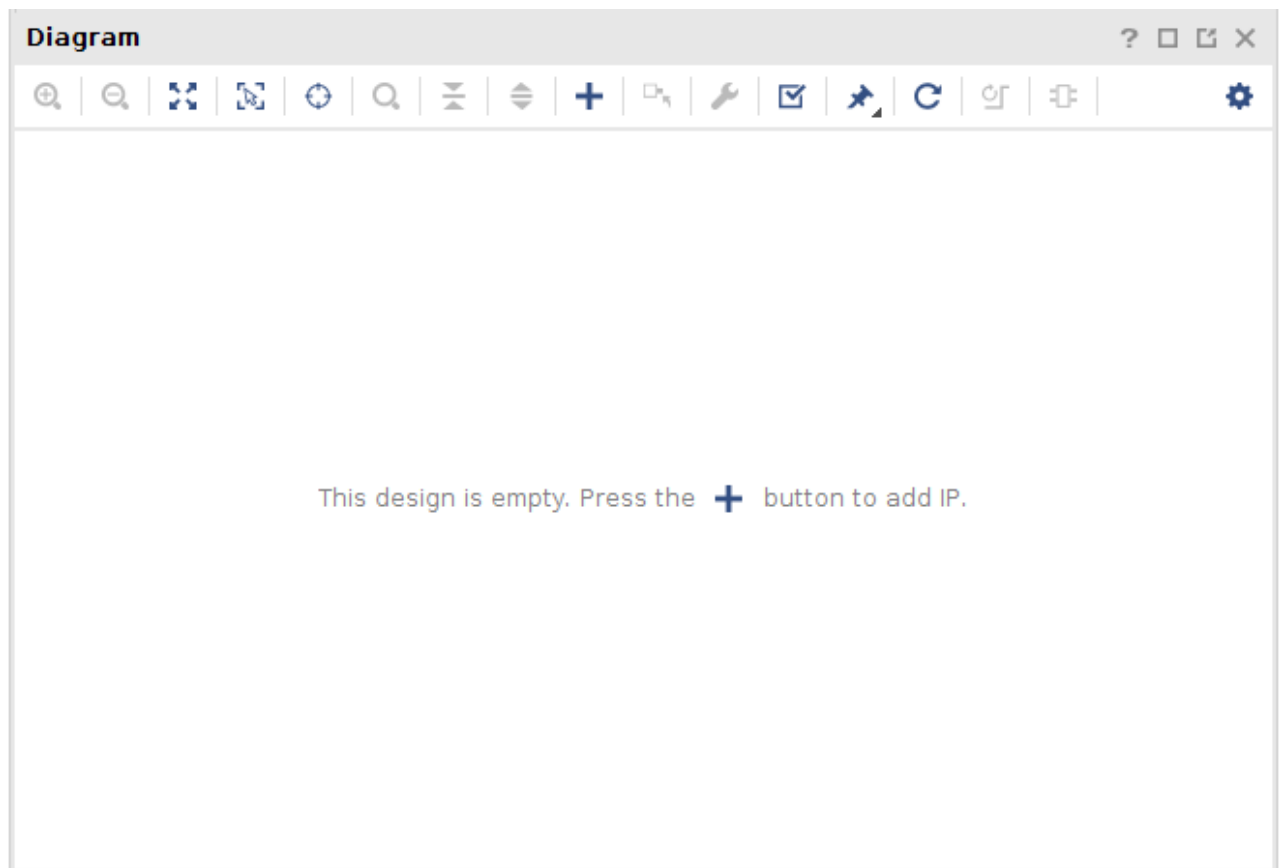
You will now add several IP blocks to the IP integrator design. The following table list of the added IP blocks and a summary of their usage in an SDSoC platform:

--	--

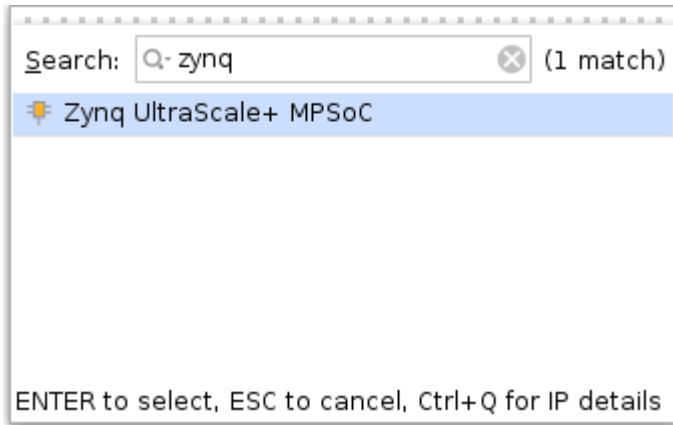
IP Block	Usage Summary
Zynq UltraScale+ MPSoC Processor System (PS)	<ul style="list-style-type: none"> <li>• Dual-core Arm processor with cache hierarchy</li> <li>• Integrated I/O peripherals</li> <li>• DDR memory controller with external memory interface</li> <li>• PS to Programmable Logic (PL) interconnects</li> <li>• PL to PS interconnects</li> </ul>
Processor System Reset Block (PL)	<ul style="list-style-type: none"> <li>• Reset sequencing and synchronization block for PL logic</li> </ul>
Clocking Wizard (PL)	<ul style="list-style-type: none"> <li>• Multiple output clock generator to drive PL logic</li> </ul>
Concat Block (PL)	<ul style="list-style-type: none"> <li>• PL interrupt structure that feeds Zynq UltraScale+ MPSoC PS interrupt request input</li> </ul>

1. On the block design canvas, right-click and select **Add IP**.

Alternatively, you can click the Add IP button (+) on the IP integrator canvas.

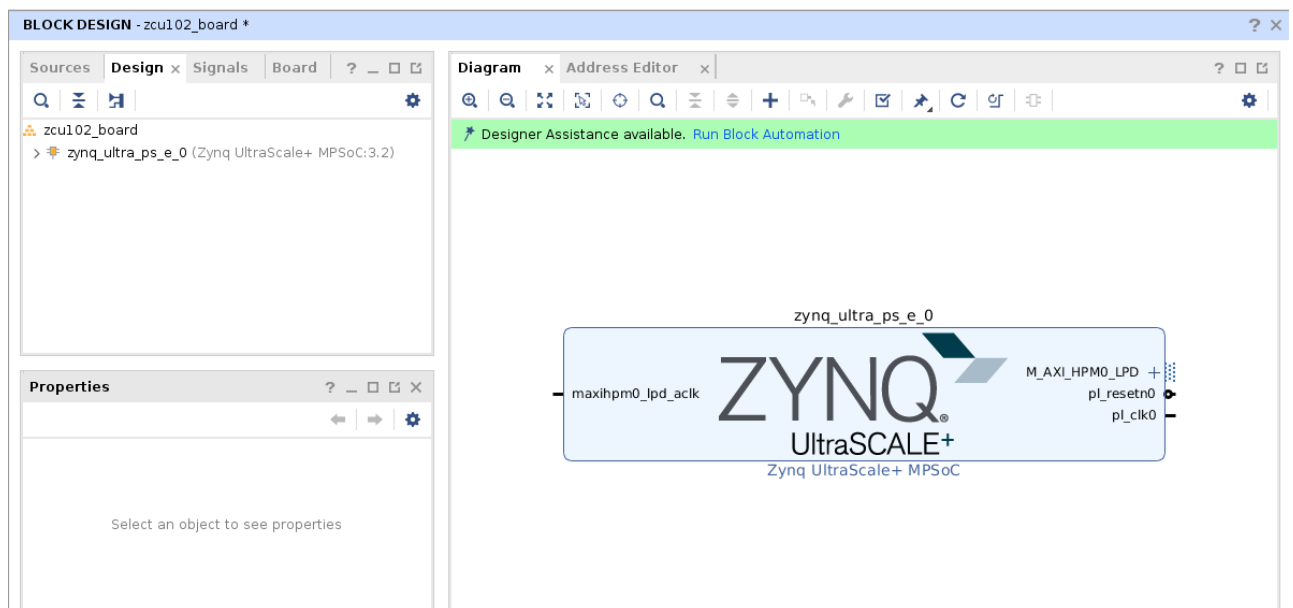


2. The IP catalog **Search** dialog box appears. In the Search field, type **zynq** to find the Zynq UltraScale+ MPSoC IP.



3. Select the **Zynq UltraScale+ MPSoC** and press the `Enter` key to add the IP to your design.

You can also double-click the IP block to add it to the IP integrator canvas.

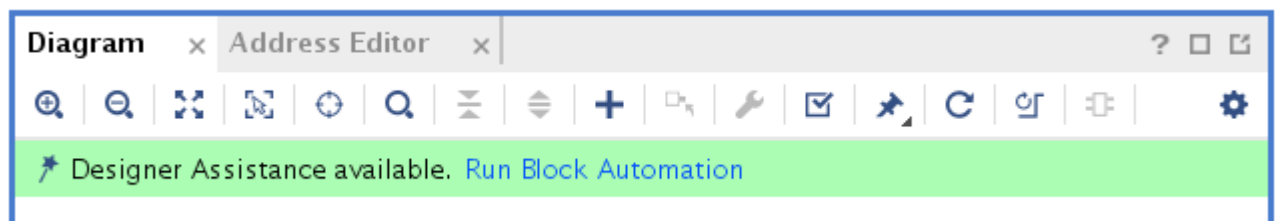


The Zynq UltraScale+ MPSoC is added to the IP integrator canvas and the Tcl Console window also shows the command used to add the IP.

```
create_bd_cell -type ip -vlnv xilinx.com:ip:zynq_ultra_ps_e:3.2 zynq_ultra_ps_e_0
```

**NOTE** There is a corresponding Tcl command for most actions performed in the block design. Tcl commands are documented in the *Vivado Design Suite: Tcl Command Reference Guide* ([UG835](#)).

4. In the IP integrator window, click the **Run Block Automation** link.

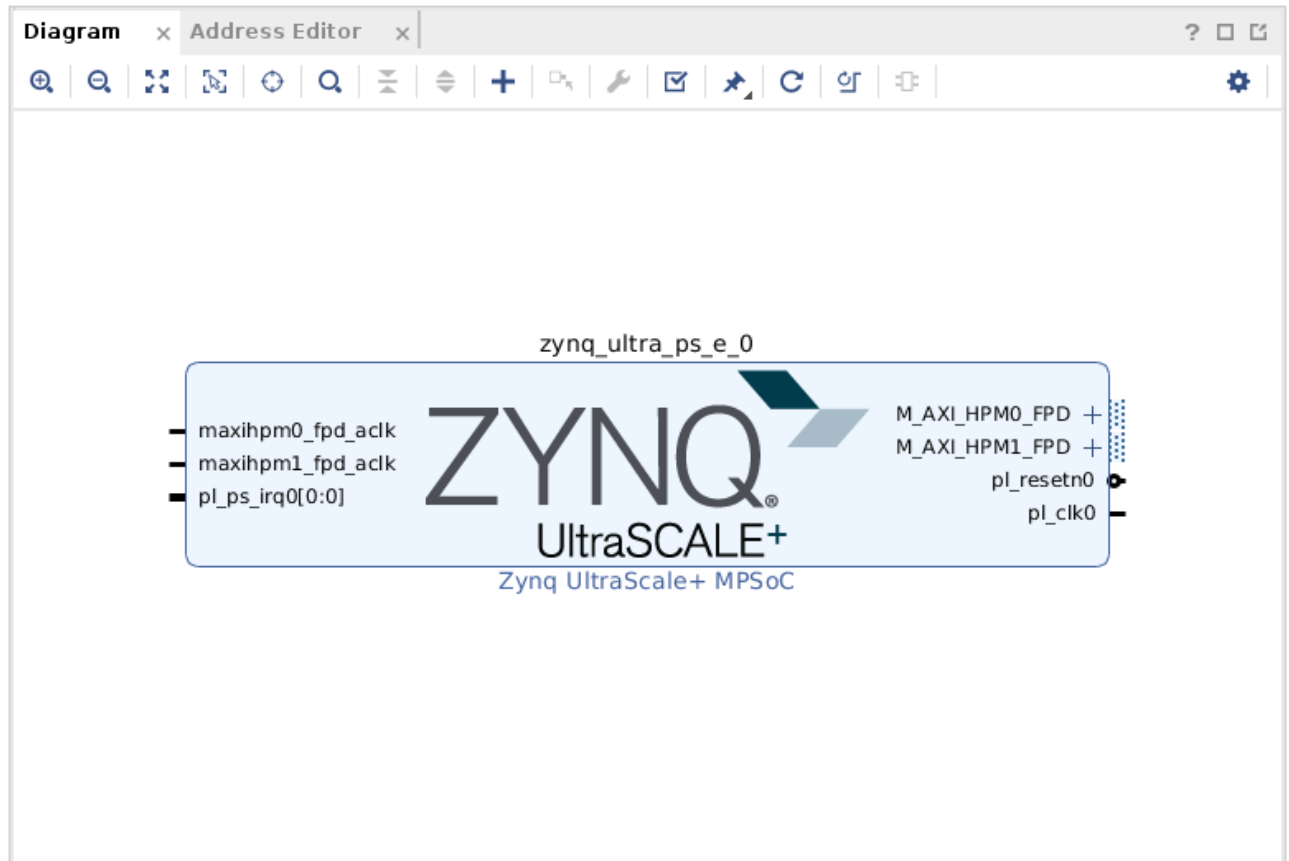


The **Run Block Automation** dialog box opens, as shown below. The default check in the **Apply Board Preset** field allows the tool to configure the PS to take advantage of the predefined board.

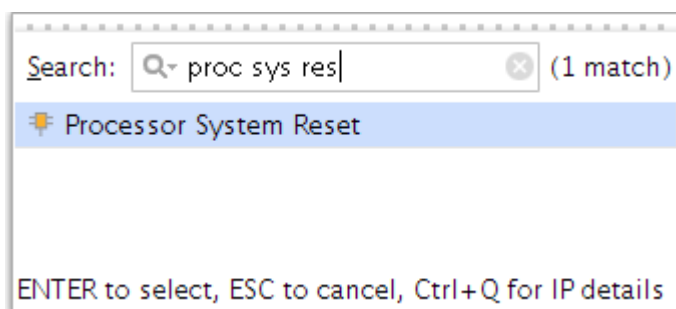


- Click **OK** to accept the default settings for the ZCU102 board.

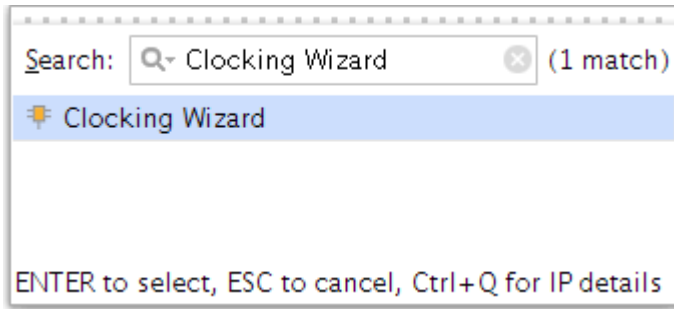
The IP integrator diagram is updated after running block automation on the Zynq UltraScale+ MPSoC IP block.



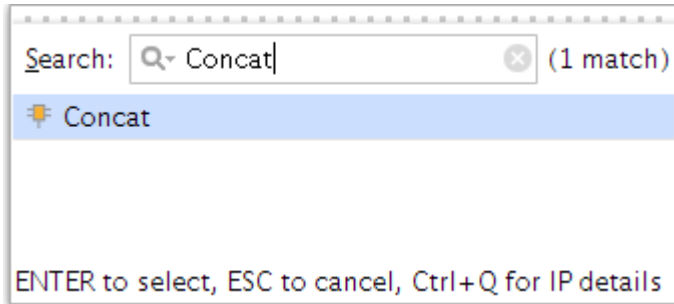
- Right-click the IP integrator diagram and select **Add IP** to add peripherals to the PL.
- In the **Search** field, type **proc sys res** to find the Processor System Reset, and then press **Enter** to add it to the design. A synchronized reset signal for each declared platform clock is created using a Processor System Reset IP block. Each reset will be associated with a clock generated by the Clocking Wizard.



- Add a **Clocking Wizard** IP block to provide PL clock(s) for the platform. You will customize the clock settings in a subsequent step.

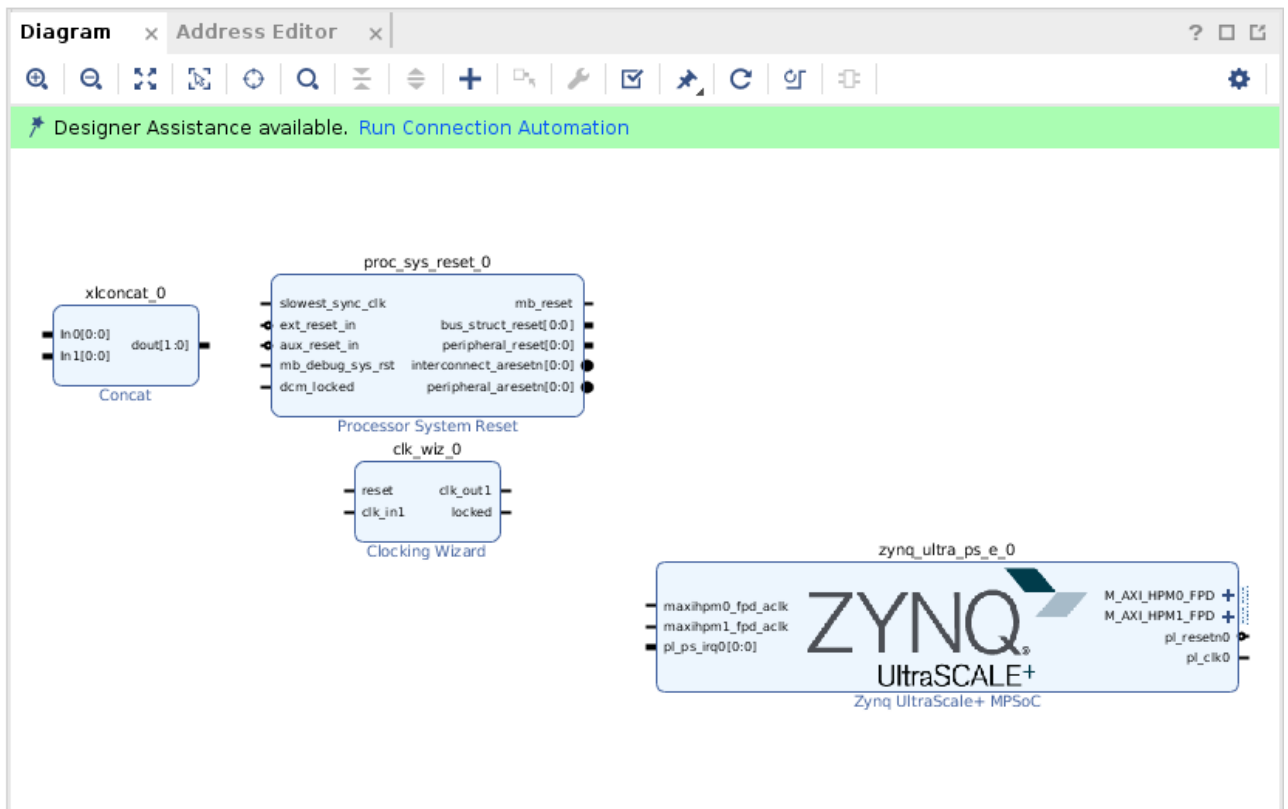


9. Add the **Concat** IP to connect PL generated interrupts to the PS block. This provides a path for the hardware accelerator to interrupt the processor if needed.



Your block design window should look similar to the figure below. The relative positions of the IP might vary.

**TIP** You can zoom in and out of the IP integrator diagram using the Zoom In and Zoom Out buttons or their equivalent keyboard shortcuts (**Ctrl+Equals** and **Ctrl+Minus**, respectively).

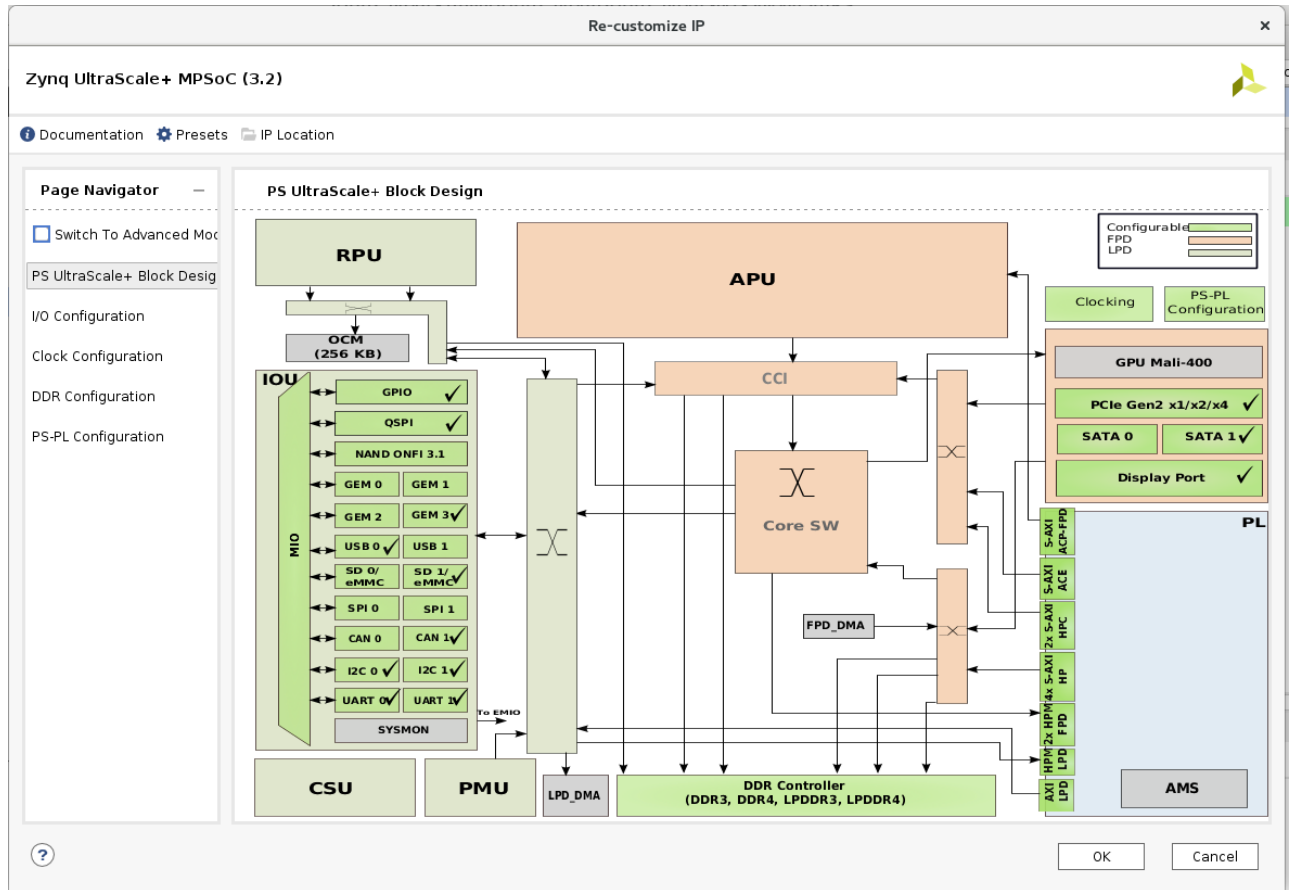


## Configuring the ZYNQ MPSoC

1. Double-click the ZYNQ UltraScale+ MPSoC IP block to open the Re-customize IP dialog box.

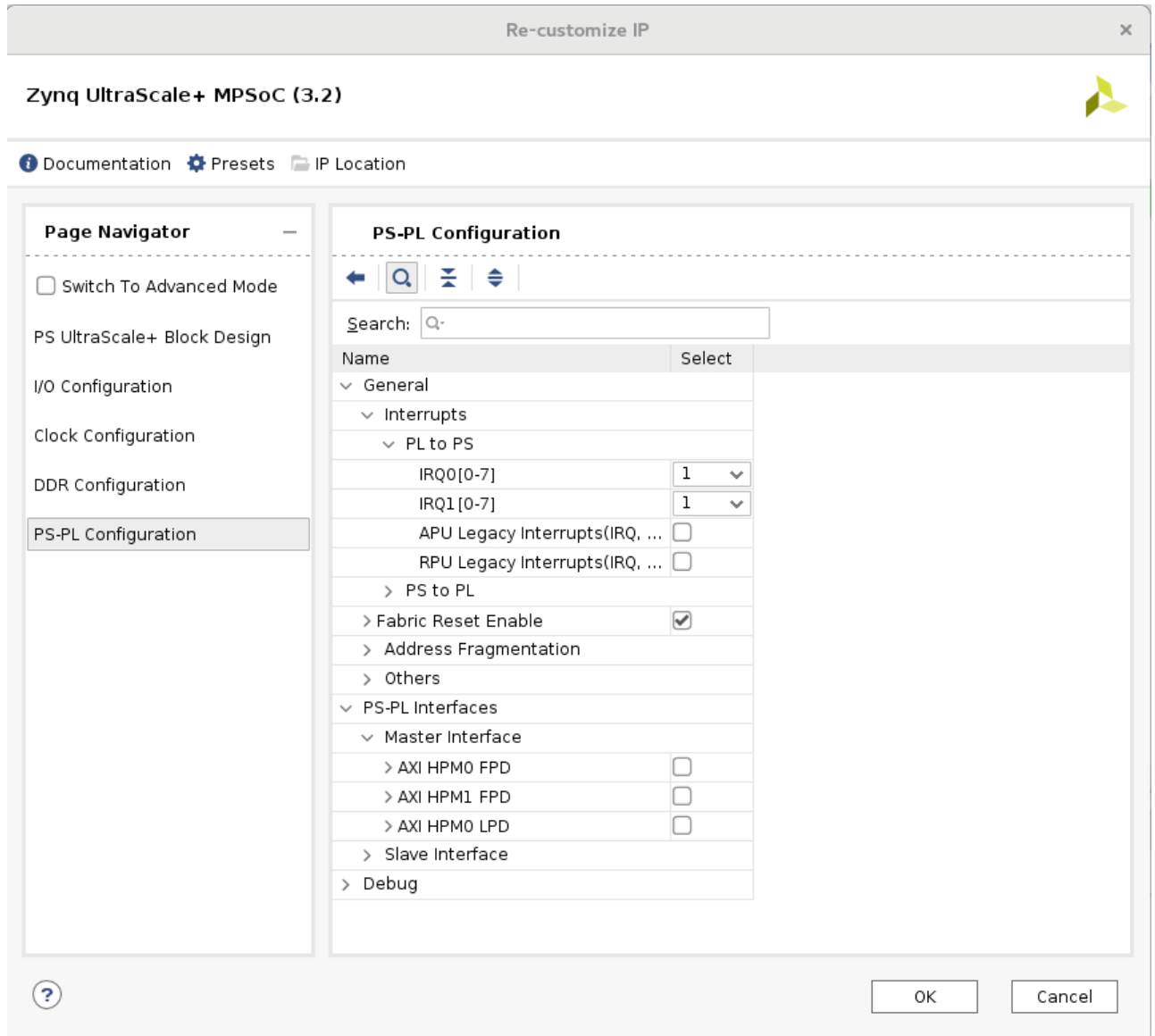


## 2. Recustomize the ARM configuration to enable a path for PL to PS interrupts exists.



3. In the Re-customize IP dialog box, select the PS-PL Configuration page in the Page Navigator (on the left side of the dialog box), and expand the General Settings, Interrupts, and PL to PS sections. Make sure the **IRQ0[0-7]** and **IRQ1[0-7]** dropdown menu displays "1", as shown in the figure below. This allows up to 8 PL interrupts to be handled by each input port of the PS interrupt.

This enables the PS-PL interrupt interface that will be connected to the PL Concat IP block. The input side of the Concat IP block is left open for the sds++ system compiler to route PL interrupts from the hardware accelerator into the PS as needed.



4. On the same PS-PL Configuration page, expand the PS-PL Interfaces, and Master Interface sections. Uncheck the **AXI HPM0 FPD** and **AXI HPM1 FPD** checkboxes, as shown in the previous figure.

Unchecking these boxes keeps the AXI HPM interfaces available for accelerator attachment by the sds++ system compiler. The default configuration reserves the M\_AXI\_HPM interfaces for use by the PS, so by un-checking these boxes you are freeing up the interface for use by the SDx tools.

5. Click **OK**.

## Configuring the Clocking Wizard IP

1. Re-customize the Clocking Wizard by double-clicking on its IP block.
2. In the **Clocking Options** tab of the Re-customize IP dialog box, examine the **Input Frequency** check box for the **Primary Input Clock**. The ZCU102 pre-set for the PS block provides a 100 MHz clock (pl\_clk0) which will be connected as an input to the Clocking Wizard IP. When the Input Frequency is set to AUTO, the input clock frequency is automatically determined from the connected input clock source.
3. Click the **Output Clocks** tab and check that the output frequency of clk\_out1 is set to 100 MHz.

4. With the **Output Clocks** tab still selected, scroll down to the bottom of the page, and set the **Reset Type** to **Active Low**, as shown in the following figure.

Component Name: clk\_wiz\_0

Board | Clocking Options | **Output Clocks** | MMCM Settings | Summary

The phase is calculated relative to clk\_out1.

Output Clock	Port Name	Output Freq (MHz)		Phase (degrees)		Duty Cycle (%)		Drives
		Requested	Actual	Requested	Actual	Requested	Actual	
<input checked="" type="checkbox"/> clk_out1	clk_out1	100.000	100.000	0.000	0.000	50.000	50.0	Buffer
<input type="checkbox"/> clk_out2	clk_out2	100.000	N/A	0.000	N/A	50.000	N/A	Buffer
<input type="checkbox"/> clk_out3	clk_out3	100.000	N/A	0.000	N/A	50.000	N/A	Buffer
<input type="checkbox"/> clk_out4	clk_out4	100.000	N/A	0.000	N/A	50.000	N/A	Buffer
<input type="checkbox"/> clk_out5	clk_out5	100.000	N/A	0.000	N/A	50.000	N/A	Buffer
<input type="checkbox"/> clk_out6	clk_out6	100.000	N/A	0.000	N/A	50.000	N/A	Buffer
<input type="checkbox"/> clk_out7	clk_out7	100.000	N/A	0.000	N/A	50.000	N/A	Buffer

☐ USE CLOCK SEQUENCING

Output Clock	Sequence Number
clk_out1	1
clk_out2	1
clk_out3	1
clk_out4	1
clk_out5	1
clk_out6	1
clk_out7	1

Enable Optional Inputs / Outputs for MMCM/PLL: ☒ reset ☐ power\_down ☐ input\_clk\_stopped ☒ locked ☐ clkfbstopped

Reset Type: ☐ Active High ☒ Active Low

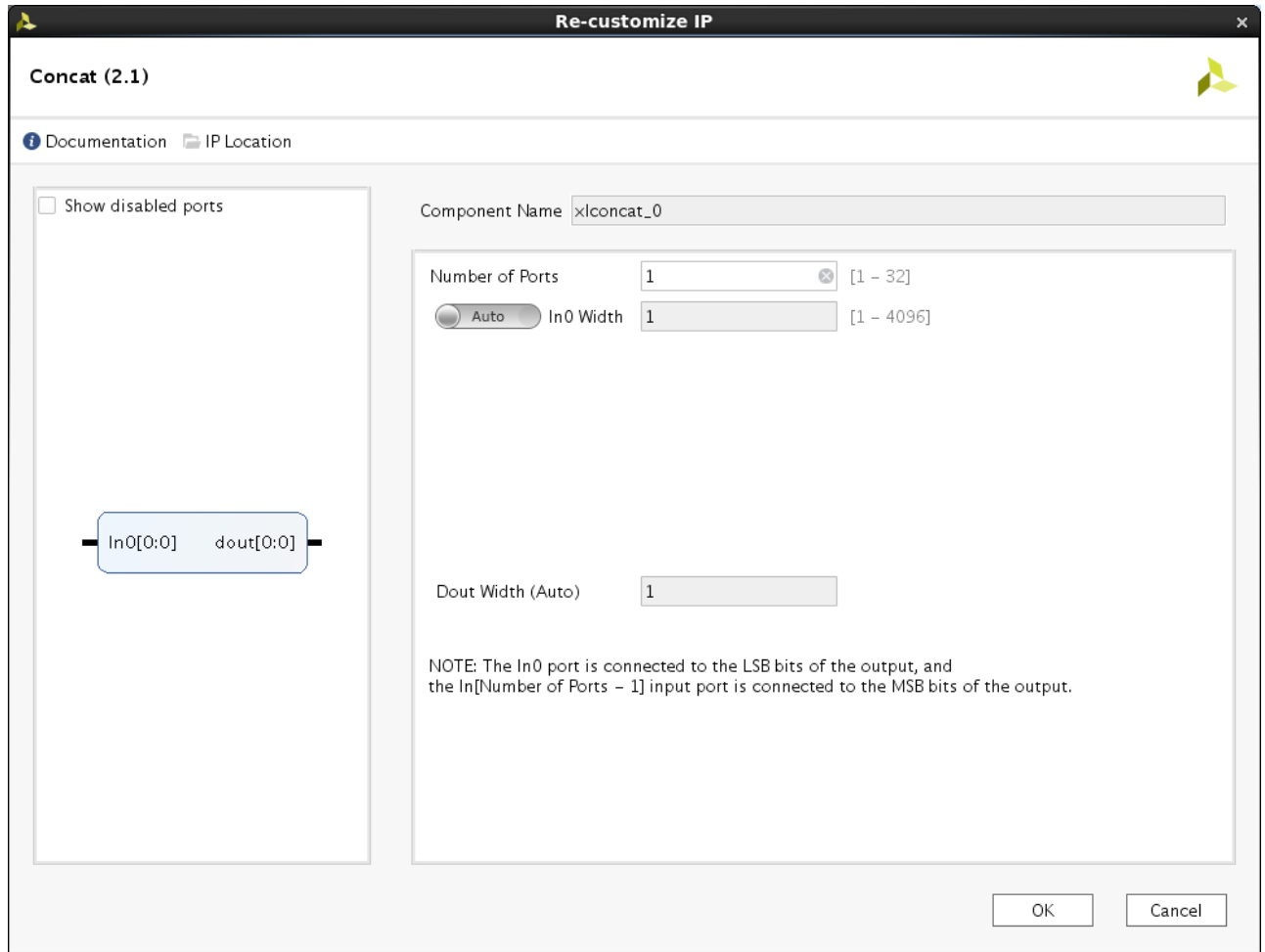
Phase Shift Mode: ☐ WAVEFORM ☒ LATENCY

5. Click **OK** to configure the Clocking Wizard IP.

## Configuring the Concat Block

1. Double-click the **Concat** IP block to open the Re-Customize IP dialog box. PL interrupts will be routed into the PS by the sds++ system compiler through the Concat IP block.
2. Change the **Number of Ports** field to 1 and press **Enter**.

This enables interrupts from the PL side that the sds++ system compiler may generate for the PS.

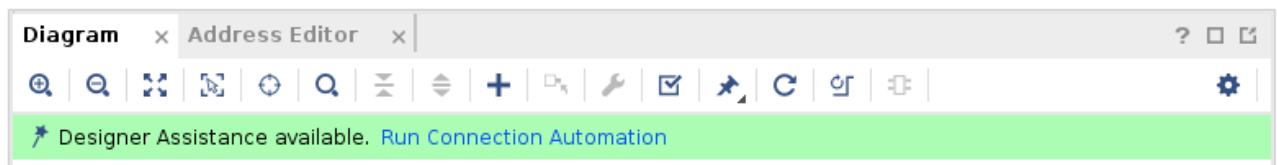


3. Click **OK** to configure the Concat IP.

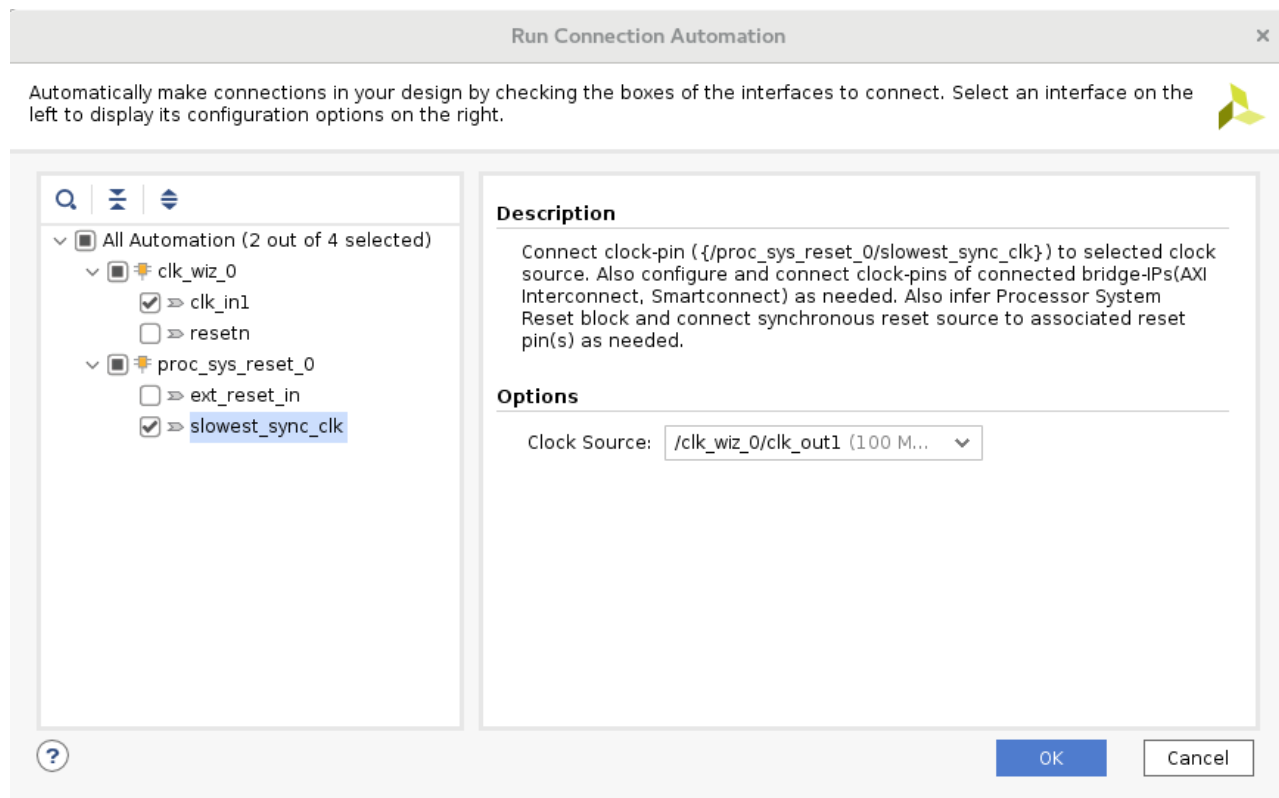
## Using Designer Assistance

Designer Assistance helps connect the Clocking Wizard and Processor System Reset blocks to the Zynq UltraScale+ MPSoC processing system.

1. Click **Run Connection Automation**. The **Run Connection Automation** dialog box appears.



2. Select the clock options for the IP as shown in the following dialog box. As you select each interface on which connection automation is run, the Description and Options available for the selected interface are shown on the right-hand side of the dialog box.



Change the **slowest\_sync\_clk** setting to the **/clk\_wiz\_0/clk\_out1** as shown in the figure.

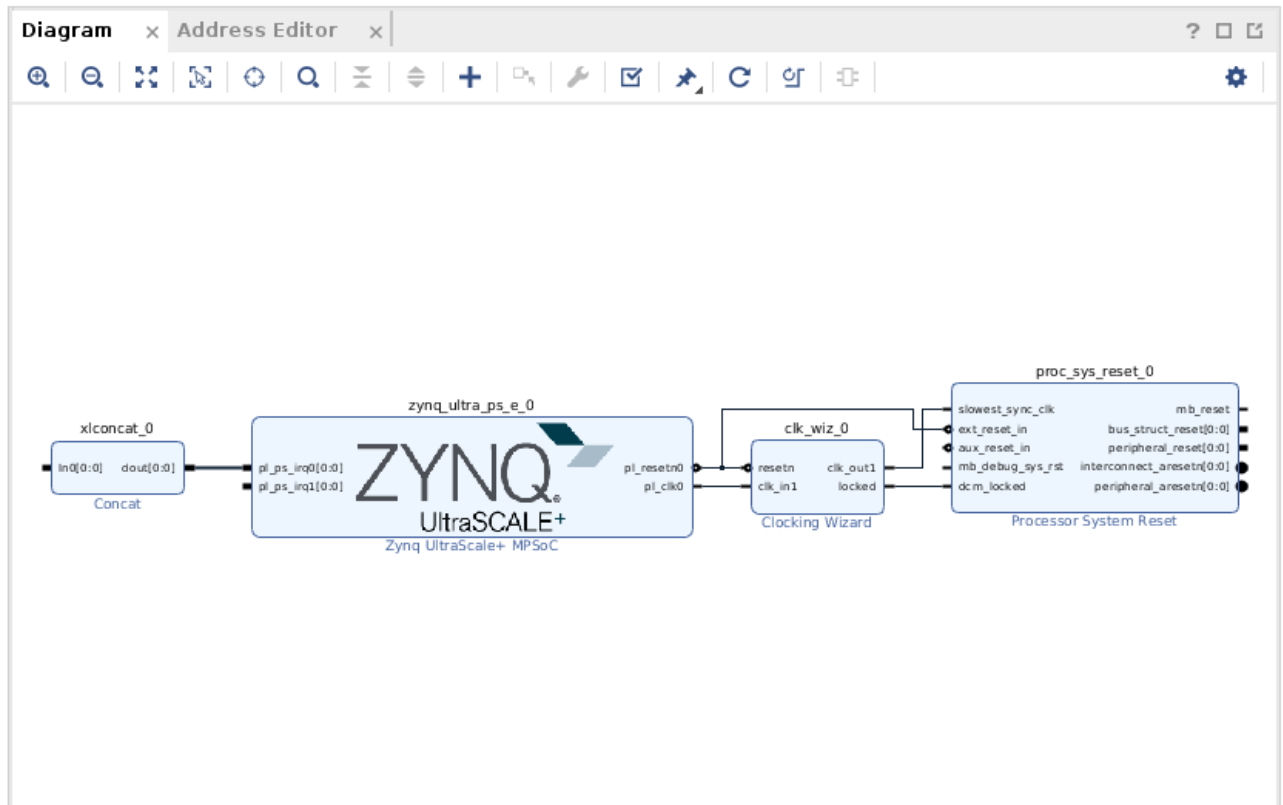
3. Click **OK** to save the settings and close the dialog box.

The Vivado IP Integrator feature adds connections to the blocks based on the settings you provided for the **Run Connection Automation** command.

## Manually Connecting the Rest of the Design

You will complete the design by manually wiring connections that could have multiple design dependent sources. In this example platform, you use a reset output generated by the PS block to control the reset inputs of logic on the PL-side. Specifically, the Processor System Reset IP block and the Clocking wizard's clock sources are connected in this manner. You also keep the PL Processor System Reset blocks in their reset state until the clock sources have locked to their requested frequencies. Additionally, any PL generated interrupts are wired to the interrupt request inputs of the PS block.

1. Connect the **pl\_resetn0** output pin of the Zynq UltraScale+ MPSoC IP to the **resetn** input pin of the Clocking Wizard.
2. Connect the **pl\_resetn0** output pin of the Zynq UltraScale+ MPSoC IP to the **ext\_reset\_in** pin on the Processor System Reset IP block.
3. Connect the **locked** output pin of the Clocking Wizard IP to the **dcm\_locked** input pin on the Processor System Reset IP block.
4. Connect the **dout[0:0]** output pin of the xlcconcat\_0 block to the **pl\_ps\_irq0[0:0]** input pin of the Zynq UltraScale+ MPSoC IP block.
5. Click the **Regenerate Layout** command to re-draw the block design with a more optimal layout. At this point, the block diagram should look similar to the view in the figure below.



6. Click the **Validate** button to validate the design.



7. Click **OK** in the Validate Design dialog box.

8. Save the block design by clicking on the floppy disk icon in the toolbar or by pressing the **Ctrl+S** keys.

### Adding Clocks, Resets, and Interrupts

At this point, the block design represents the framework of your platform: the Zynq UltraScale+ MPSoC, clocking wizard and processor system reset, and interrupts managed through the concat block. The platform only has one clock and reset, and only has space available for 8 interrupts. However, this may be sufficient for your platform requirements, and you can move on to the next steps in this Lab.

For the purposes of this tutorial though, you will add some additional clocks, resets, and interrupts. Providing a variety of clock frequencies in the PL region adds design flexibility to the base platform. An SDx hardware accelerator can change input clock sources without having to re-define and re-build the base platform if multiple clock sources are available in the platform.

1. Re-customize the Clocking Wizard by double-clicking on it.

2. Click the **Output Clocks** tab and enable all output clocks, **clk\_out1** to **clk\_out7**, by clicking on their respective check-boxes. Set the output frequencies under the **Output Freq (MHz) Requested** column as follows:

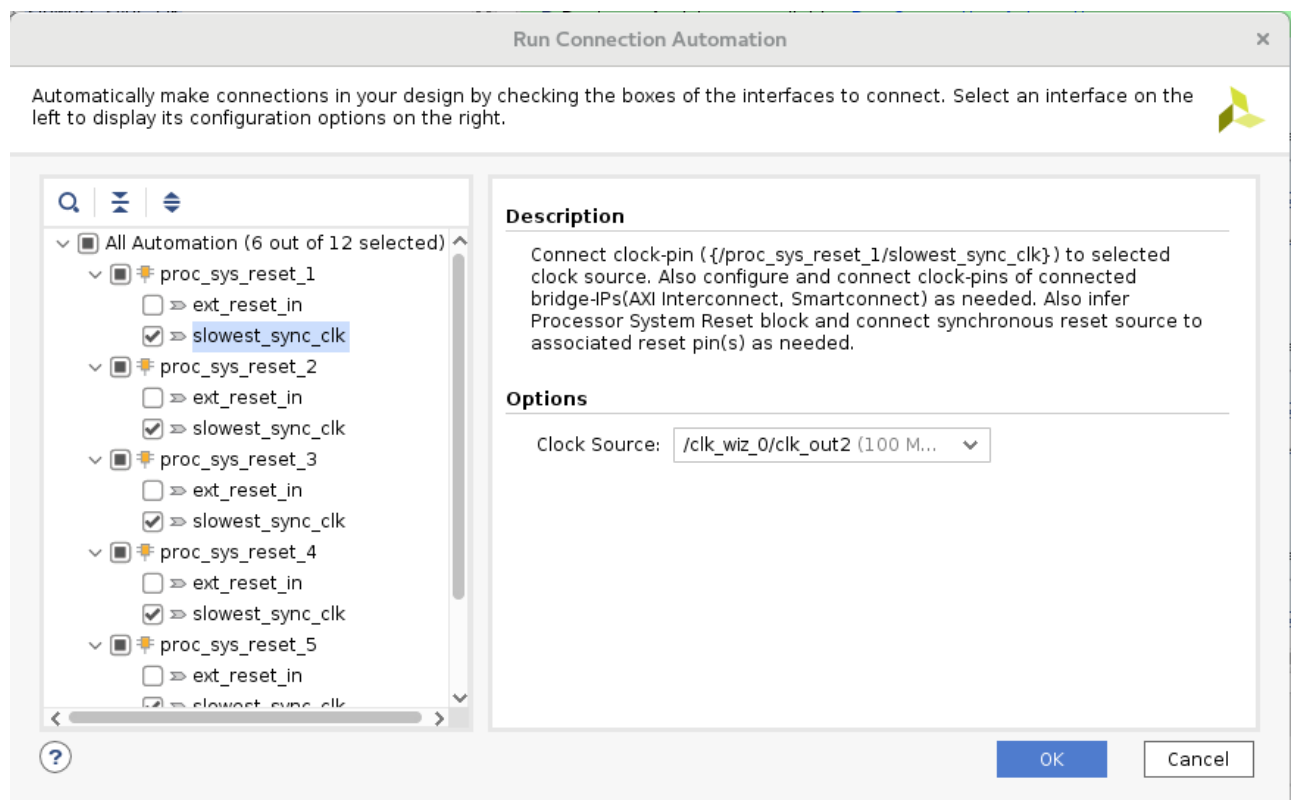
- clk\_out1 -> 75
- clk\_out2 -> 100
- clk\_out3 -> 150
- clk\_out4 -> 200.000
- clk\_out5 -> 300.000
- clk\_out6 -> 400.000
- clk\_out7 -> 600.000

3. Select the **proc\_sys\_reset\_0** block, and copy and paste it multiple times, for a total of seven reset IP blocks.

**Note:** You can use keyboard shortcuts Ctrl-C and Ctrl-V to copy and paste on the design canvas.

4. Select the **xlconcat\_0** block, and copy and paste it to create a second instance.

5. Select **Run Connection Automation**. The dialog box should appear as follows.



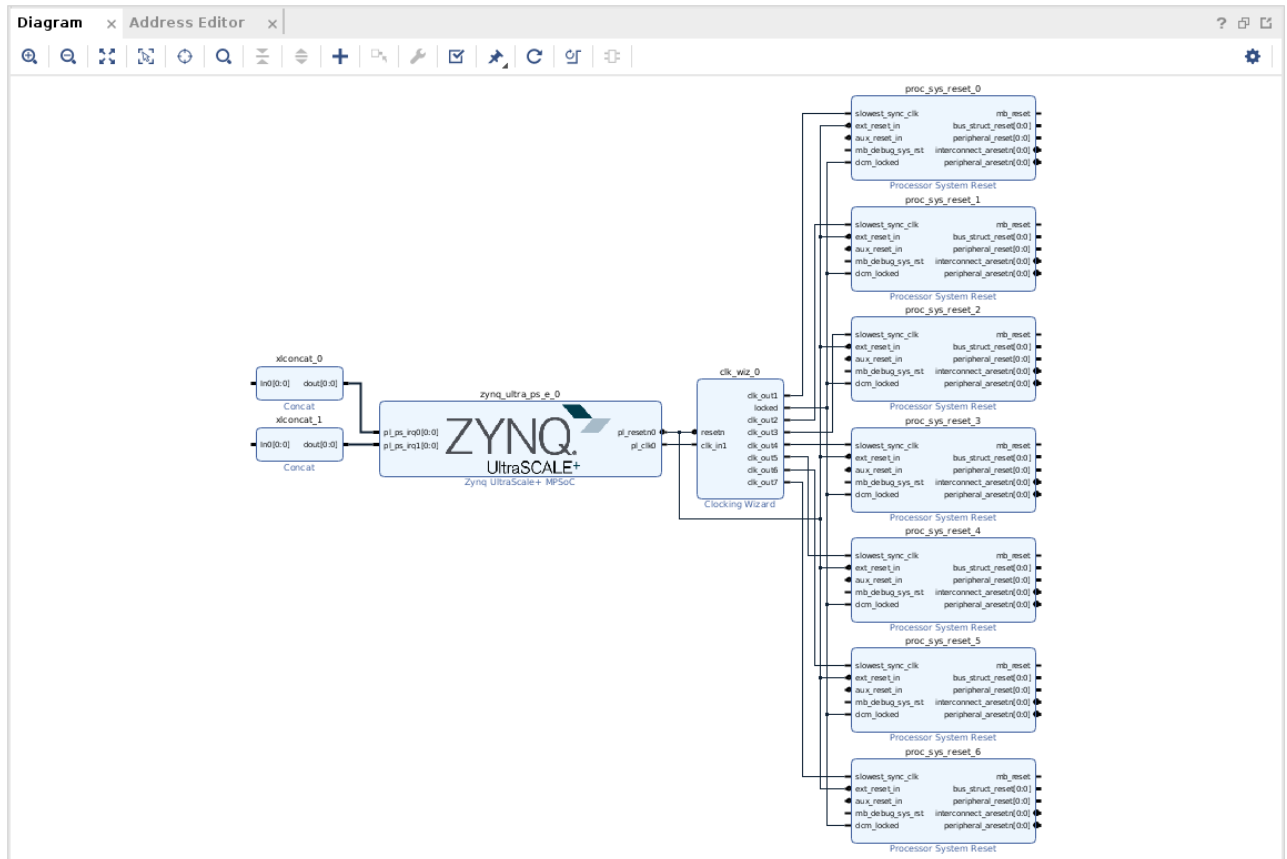
Ensure that the automation options are set as in the table below.

**Note:** Some of the elements in the table may have been previously connected, and will not appear in the table. The complete table is provided for your reference.

Connection	Description	Setting
<b>clk_wiz_0</b> - clk_in1	The input clock to the clocking wizard	<i>/zynq_ultra_ps_e_0/pl_clk0 (99 MHz)</i> is selected by default as the Clock Source option. Leave set to the default value.
<b>proc_sys_reset_0</b> - slowest_sync_clk	Clock source to which this reset is synchronized	Select <b>/clk_wiz_0/clk_out1 (75 MHz)</b> from the Clock Source options drop-down menu.
<b>proc_sys_reset_1</b> - slowest_sync_clk	Clock source to which this reset is synchronized	Select <b>/clk_wiz_0/clk_out2 (100 MHz)</b> from Clock Source options drop-down menu.
<b>proc_sys_reset_2</b> - slowest_sync_clk	Clock source to which this reset is synchronized	Select <b>/clk_wiz_0/clk_out3 (150 MHz)</b> from the Clock Source options drop-down menu.
<b>proc_sys_reset_3</b> - slowest_sync_clk	Clock source to which this reset is synchronized	Select <b>/clk_wiz_0/clk_out4 (200 MHz)</b> from the Clock Source options drop-down menu.
<b>proc_sys_reset_4</b> - slowest_sync_clk	Clock source to which this reset is synchronized	Select <b>/clk_wiz_0/clk_out5 (300 MHz)</b> from the Clock Source options drop-down menu.
<b>proc_sys_reset_5</b> - slowest_sync_clk	Clock source to which this reset is synchronized	Select <b>/clk_wiz_0/clk_out6 (400 MHz)</b> from the Clock Source options drop-down menu.
<b>proc_sys_reset_6</b> - slowest_sync_clk	Clock source to which this reset is synchronized	Select <b>/clk_wiz_0/clk_out7 (600 MHz)</b> from the Clock Source options drop-down menu.

1. Manually connect the **pl\_reseth0** output pin of the Zynq UltraScale+ MPSoC IP to the **ext\_reset\_in** pin on all of the Processor System Reset IP blocks.
2. Connect the **locked** output pin of the Clocking Wizard IP to the **dcm\_locked** input pin on all of the Processor System Reset IP blocks.
3. Manually connect the **dout[0:0]** output pin of the xclconcat\_1 block to the **pl\_ps\_irq1[0:0]** input pin of the Zynq UltraScale+ MPSoC IP block.
4. Validate and save the block design again. The block design you have created now matches the released ZCU102 platform, and should look similar to the figure below.





### ► Step 3: Declaring the Platform Hardware Interfaces

After you complete the IP integrator hardware platform design, you must declare the hardware interfaces that will be available as attachment points for SDSoc accelerators and the data movers that will communicate with them. These declarations are added to the design by setting platform (PFM) properties that define the platform name and indicate which specific clocks, interrupts, and bus interfaces are available for the sds+ + system compiler as it generates the hardware accelerators. These properties are stored in the project.

As the PFM properties persist within the block design (BD), if you are starting a new platform from an existing platform project, there may be existing properties that are not intended for the new platform and can lead to invalid hardware platforms. In these cases, you must unset any conflicting PFM properties.

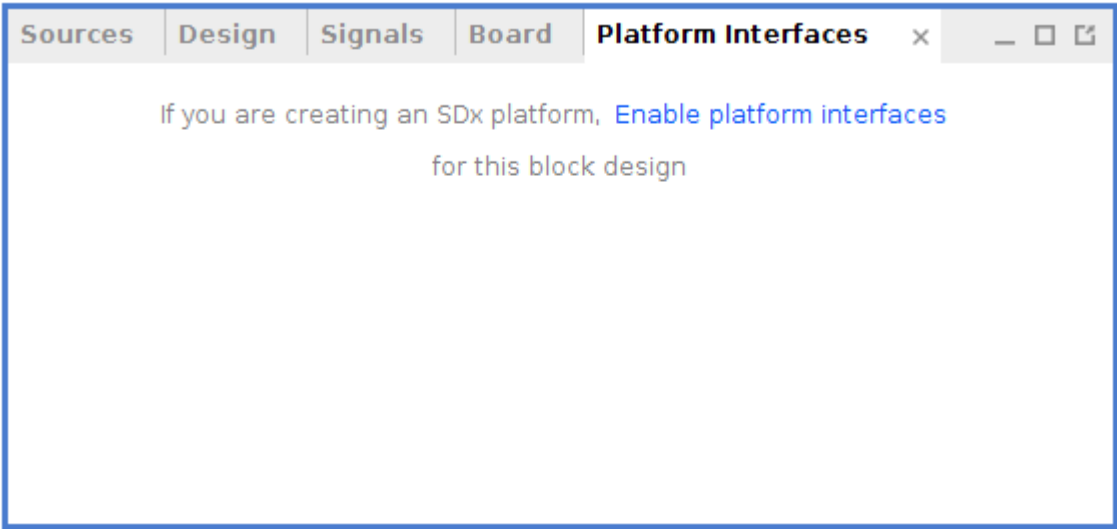
In this lab, you use the Platform Interfaces window of the Vivado IP Integrator feature to declare the hardware interfaces. These PFM properties will be added to the block design. You can also use the Vivado Block Properties tab or the TCL console to set the PFM properties that declare the hardware interfaces. At the end of this section the TCL commands are provided that you can use to define the PFM properties.

Your block design contains seven different clocks generated through the Clocking Wizard. While only one of the clocks is enabled in this lab, the others are available for use with the SDx IDE. Through the SDx IDE you will be able to select which functions to accelerate in hardware, and specify clock sources for the hardware accelerators. Likewise, you will declare the AXI ports that are available for moving data between the processor (PS) the hardware functions (PL). Although, these AXI ports may or may not be directly visible on your block design, the PFM property settings define them for use in the SDx environment.

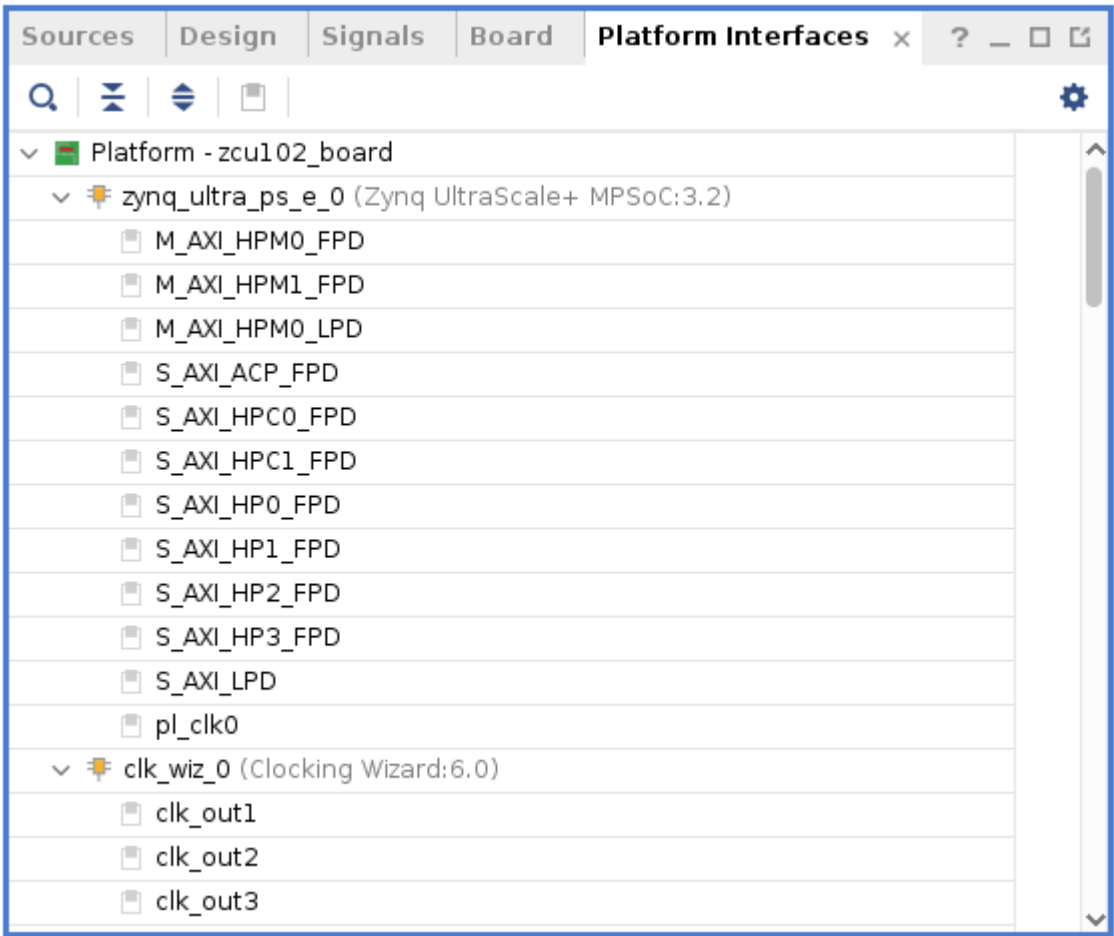
### Enabling the Platform Interfaces Tab

1. On the Vivado main menu select **Window -> Platform Interfaces**.

2. Click the **Enable platform interfaces** link to enable the Platform Interfaces tab.



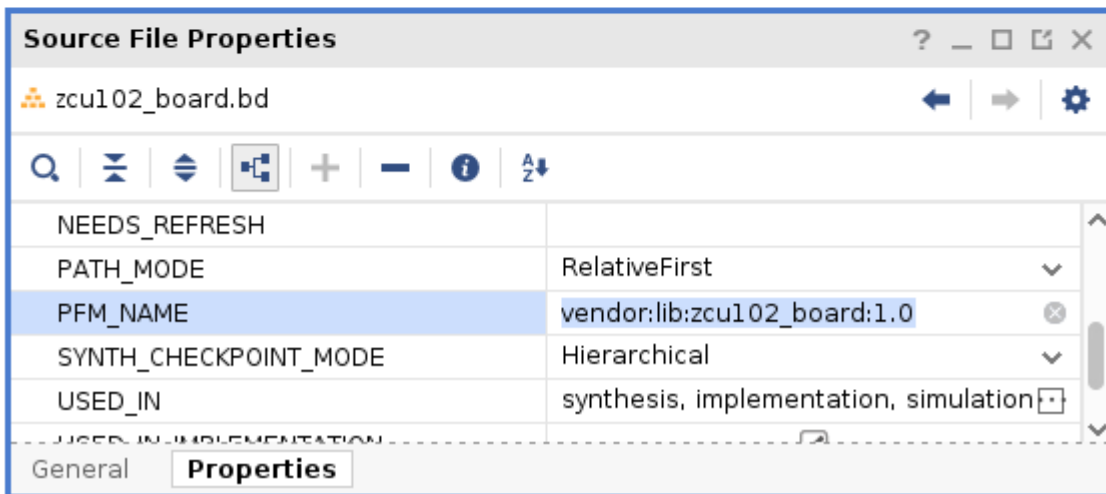
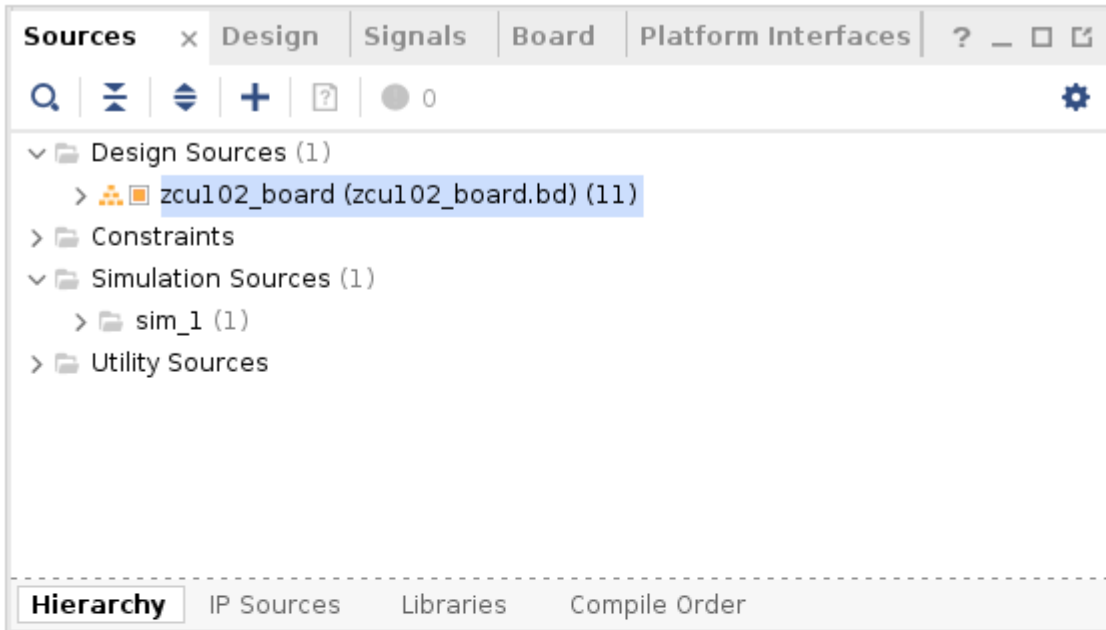
The Platform Interfaces tab shows all the interfaces available in the block design that can be enabled/disabled for use by the SDx environment in creating hardware accelerators. Right-clicking on an interface or a group of selected interfaces and then selecting Enable, changes the greyed-out icon in front of the interface name to a solid colored icon. An enabled interface means it is available for use by the SDx tools.



**Platform Name**

Enabling the Platform Interfaces tab automatically sets the platform name property on the block design, as seen in the Sources view in the following figure. The Vendor, Library, Name, and Version of the platform can

be edited from the properties tab of the block design in the Source File Properties window through the **PFM\_NAME** property. Clicking on the pencil icon at the right-hand edge of the property text box allows you to edit the property. For the purpose of this tutorial, use the defaults that have already been set.



## Platform Clocks

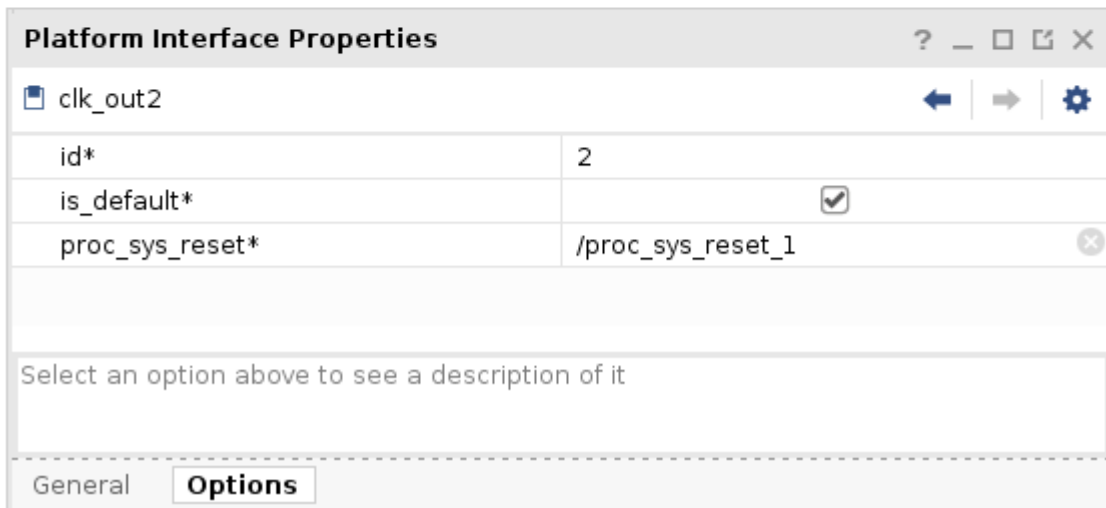
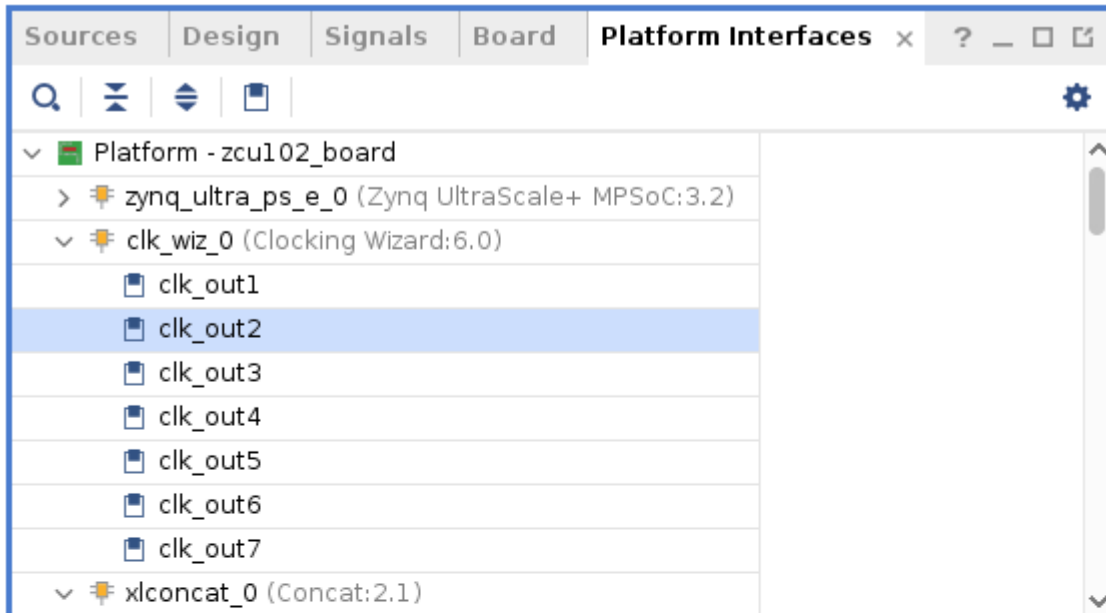
The seven output clocks generated by the Clocking Wizard are declared as available for the sds++ system compiler to use when each of the **clk\_wiz\_0** clocks are enabled in the Platform Interfaces tab, as shown in the figure below.

Each declared clock must have an associated synchronized reset signal using the Processor System Reset IP block. The **proc\_sys\_reset** property in the Platform Interface Properties dialog box is used to make this association for each declared clock. Providing multiple platform clock frequencies allows you to select an accelerator clock source that has a high-probability of being routed and meeting timing constraints when the Vivado implementation tools are invoked. If a particular clock frequency selection does not meet timing, selecting a lower frequency clock source may remedy the issue.

```
>**:pushpin: NOTE**
>To select a range, click a line, then hold the <kbd>Shift</kbd> key down and
```

click on another line to select the lines in between.

1. Right-click on each clock of the **clk\_wiz\_0**, and select **Enable** for **clk\_out1** through **clk\_out7**.
2. Select **clk\_out2** in the Platform Interfaces window.
3. In the Platform Interfaces Properties window, select the **Options** tab and click the **is\_default** option box to enable the checkbox, marking this as the default clock for accelerators.



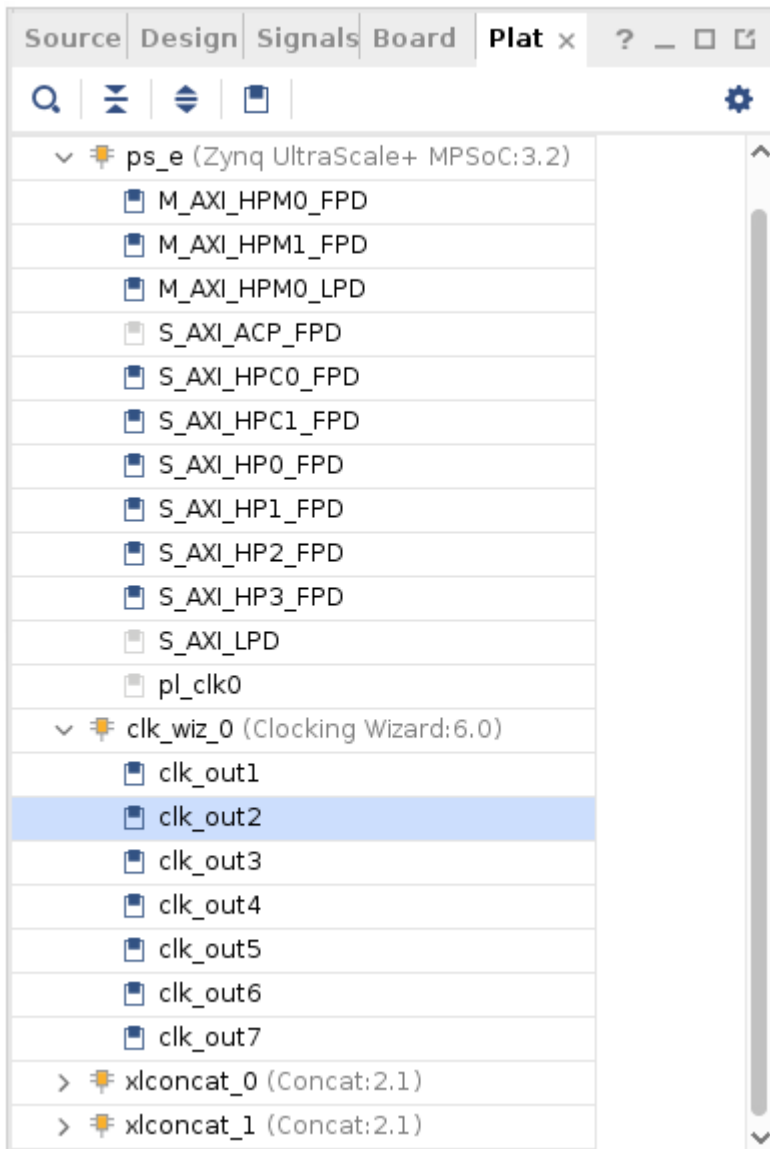
## Platform AXI Ports

In this custom platform, all the Zynq UltraScale+ MPSoC PS-side AXI ports (master and slave) are declared as available for the sds++ system compiler to use, as shown by the enabled **zynq\_ultra\_ps\_e\_0** interfaces below. The pl\_clk0 interface is not enabled for SDx accelerator use, since it is already in use by the hardware design to provide the 100.000 MHz input clock to the Clocking Wizard IP.

1. Select all of **zynq\_ultra\_ps\_e\_0** interfaces, except pl\_clk0, and select **Enable**.

**NOTE:** Remember, you can hold down **Shift** to select multiple interfaces.

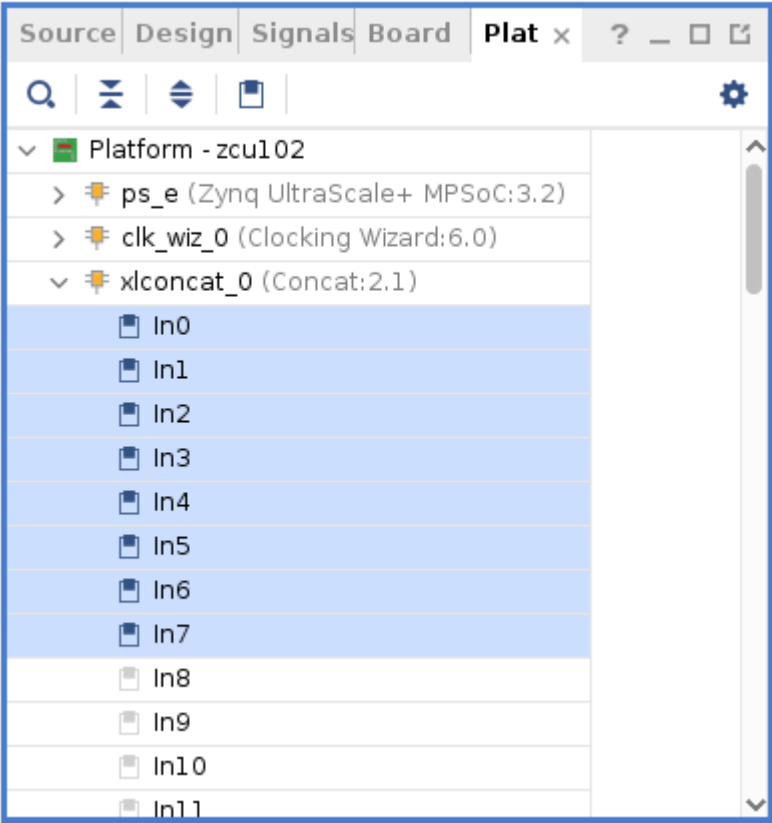
2. Right-click **S\_AXI\_ACP\_FPD**, and **Disable** the interface.
3. Right-click **S\_AXI\_LPD**, and **Disable** the interface.



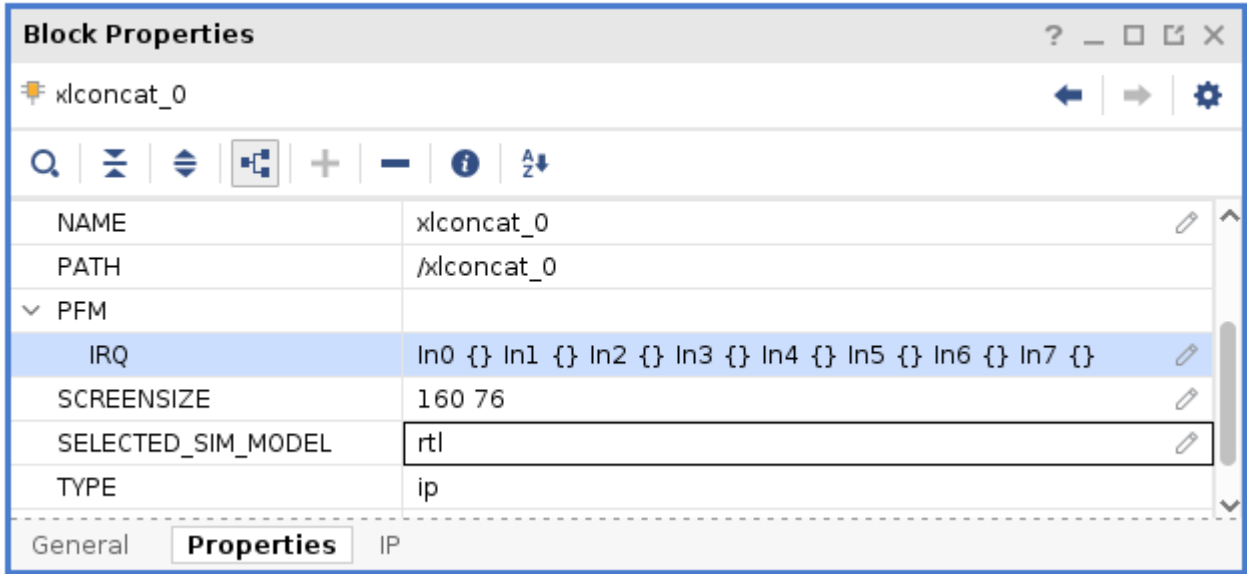
## Platform Interrupts

Interrupt sources from the hardware accelerator in the PL logic will be connected through the **Concat** IP block to the Zynq UltraScale+ MPSoC IRQ input ports (**pl\_ps\_irq0** and **pl\_ps\_irq1**). The figure below shows that 8 interrupt sources are enabled on each concat block for the sds++ system compiler to use.

1. Right-click on **xlconcat\_0** inputs **In0** to **In7** and select **Enable**.
2. Right-click on **xlconcat\_1** inputs **In0** to **In7** and select **Enable**.



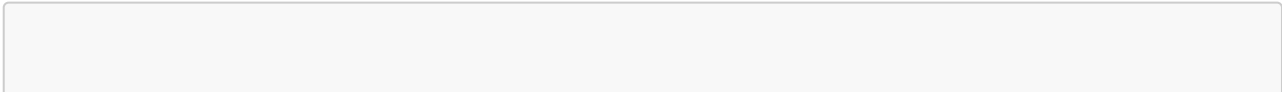
Selecting an object in the Platform Interfaces window, such as the **xlconcat\_0** object, lets you view the PFM property of the object in the Block Properties window, as shown in the figure below.



**Tcl Console Commands (For Reference)**

These commands are provided for reference and do not need to be executed if the platform properties are set through enabling and disabling selections in the **Platform Interfaces** tab, as presented in the preceding section. The Vivado journal or log files can be examined to view a history of the Tcl commands issued by actions performed in the **Platform Interfaces** tab.

1. PFM NAME



```
set_property PFM_NAME "vendor:lib:zcu102_board:1.0"\
[get_files [get_property FILE_NAME [get_bd_designs]]]
```

## 2. PFM CLOCK

```
set_property PFM.CLOCK {\
clk_out1 {id "1" is_default "false"\
proc_sys_reset "proc_sys_reset_0"}\
clk_out2 {id "2" is_default "true"\
proc_sys_reset "proc_sys_reset_1"}\
clk_out3 {id "3" is_default "false"\
proc_sys_reset "proc_sys_reset_2"}\
clk_out4 {id "4" is_default "false"\
proc_sys_reset "proc_sys_reset_3"}\
clk_out5 {id "5" is_default "false"\
proc_sys_reset "proc_sys_reset_4"}\
clk_out6 {id "6" is_default "false"\
proc_sys_reset "proc_sys_reset_5"}\
clk_out7 {id "7" is_default "false"\
proc_sys_reset "proc_sys_reset_6"}\
} [get_bd_cells /clk_wiz_0]
```

## 3. PFM AXI Ports

```
set_property PFM.AXI_PORT {\
M_AXI_HPM0_FPD {memport "M_AXI_GP" sptag "" memory ""}\
M_AXI_HPM1_FPD {memport "M_AXI_GP" sptag "" memory ""}\
M_AXI_HPM0_LPD {memport "M_AXI_GP" sptag "" memory ""}\
S_AXI_HPC0_FPD {memport "S_AXI_HPC" sptag "" memory ""}\
S_AXI_HPC1_FPD {memport "S_AXI_HPC" sptag "" memory ""}\
S_AXI_HP0_FPD {memport "S_AXI_HP" sptag "" memory ""}\
S_AXI_HP1_FPD {memport "S_AXI_HP" sptag "" memory ""}\
S_AXI_HP2_FPD {memport "S_AXI_HP" sptag "" memory ""}\
S_AXI_HP3_FPD {memport "S_AXI_HP" sptag "" memory ""}\
} [get_bd_cells /zynq_ultra_ps_e_0]
```

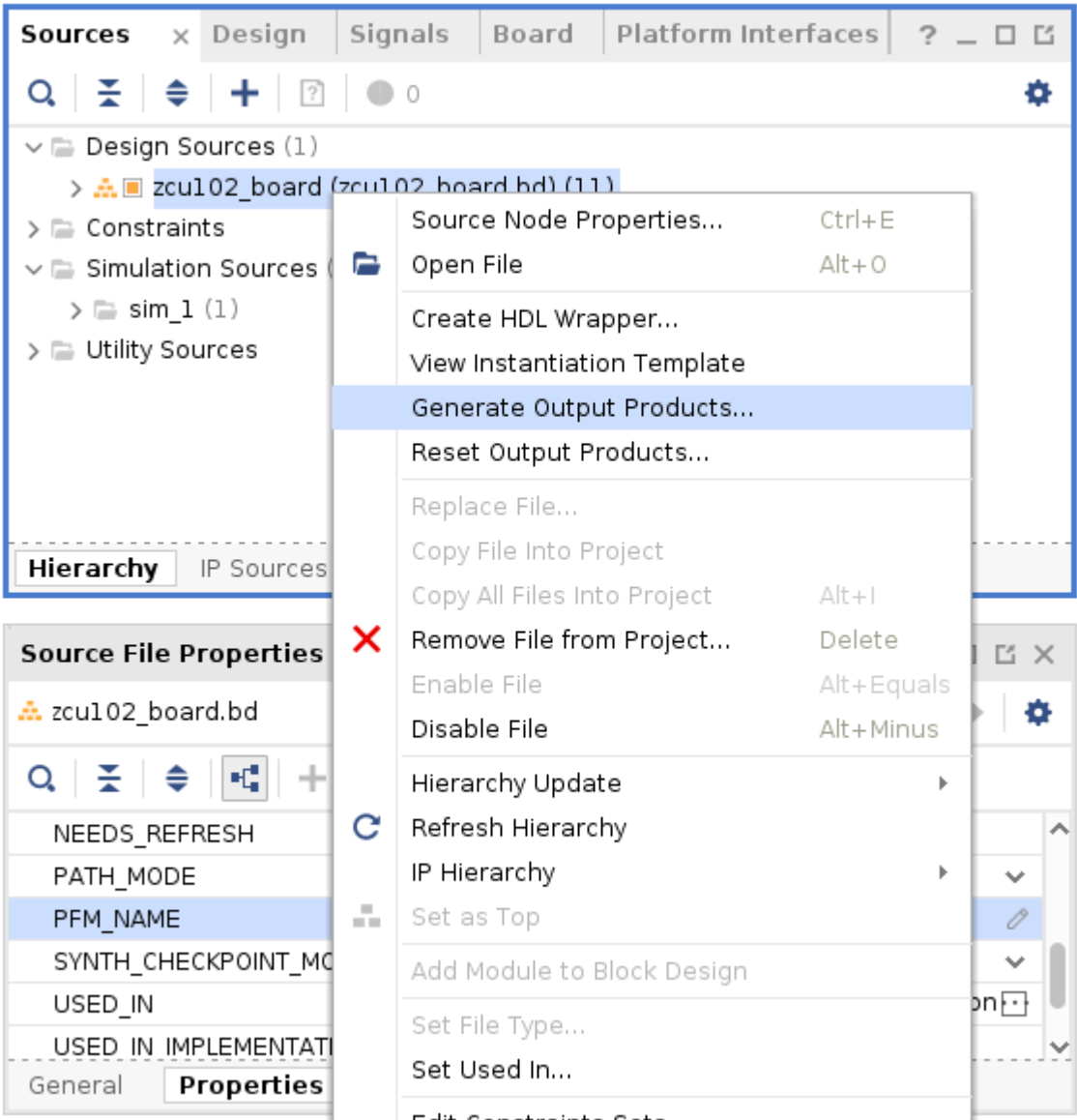
## 4. PFM Interrupts

```
set_property PFM.IRQ {\
In0 {} In1 {} In2 {} In3 {} In4 {} In5 {} In6 {} In7 {} \
} [get_bd_cells {/xlconcat_0 /xlconcat_1}]
```

## ► Step 4: Generating HDL Design Files

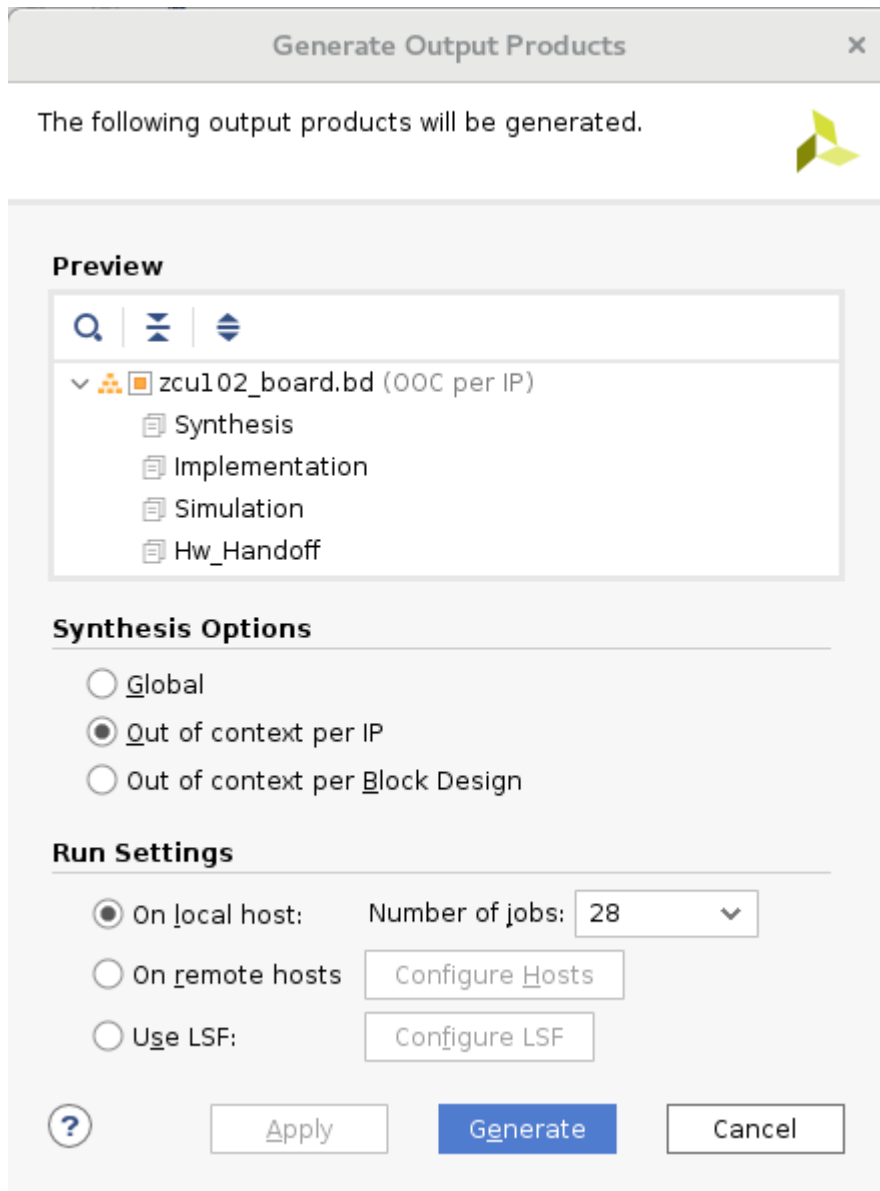
You can now generate the HDL files for the design.

- 1. In the Sources window, right-click the **zcu102\_board.bd** block design and select **Generate Output Products**.



- 2. Click **Generate**.

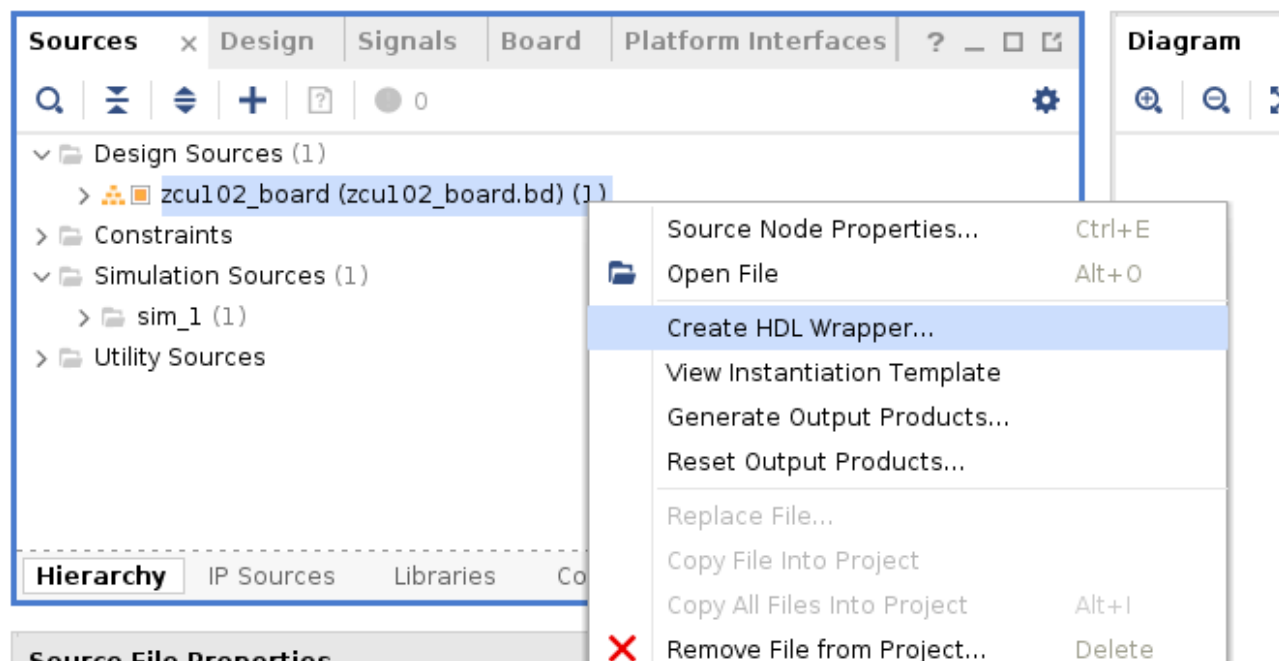




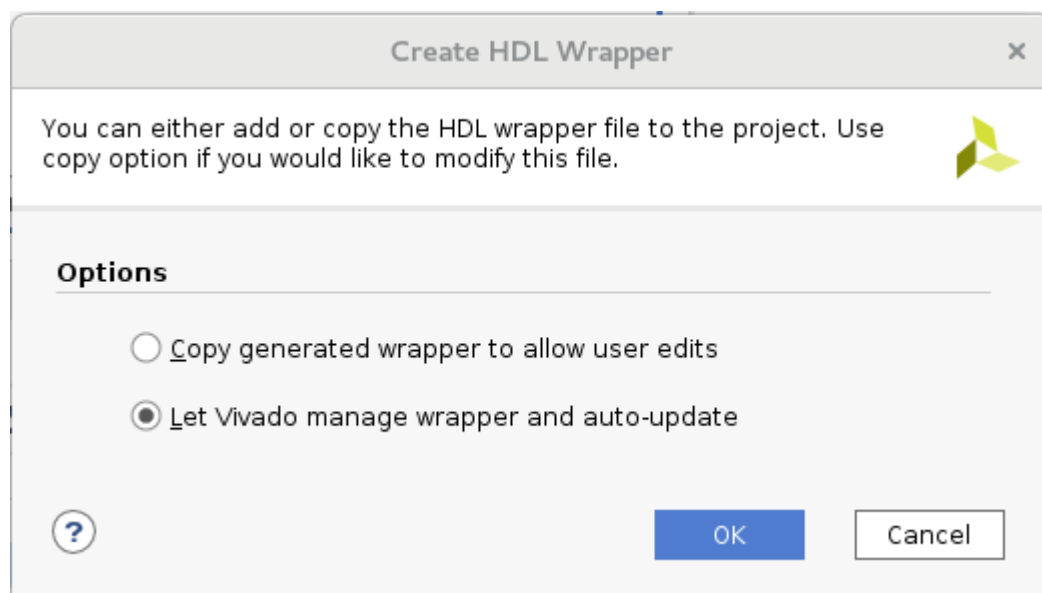
3. Observe zcu102\_board output generation completed as indicated by the status column: **Submodule Runs Complete** in the Design Runs view. The running output products generation can be set to run in the background to access the Design Runs view if necessary.

Tcl Console   Messages   Log   Reports   Design Runs x						
<input type="button" value="Q"/> <input type="button" value="Z"/> <input type="button" value="A"/> <input type="button" value="I"/> <input type="button" value="K"/> <input type="button" value="L"/> <input type="button" value="P"/> <input type="button" value="R"/> <input type="button" value="S"/> <input type="button" value="T"/>						
Name	Constraints	Status	WNS	TNS	WH	
▼ ▶ <b>synth_1</b> (active)	<b>constrs_1</b>	<b>Not started</b>				
▶ impl_1	constrs_1	Not started				
▼ Out-of-Context Module Runs						
> ✓ zcu102_board		Submodule Runs Complete				

4. Right-click on **zcu102\_board.bd** in the Sources window and click on **Create HDL Wrapper** to create a top-level HDL wrapper for the platform design.



5. Click **OK**.



6. In the Flow Navigator, click **Generate Bitstream**.

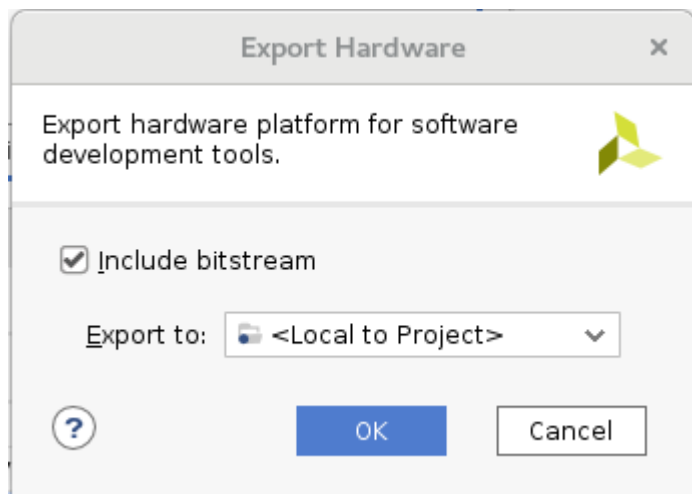
7. The Vivado tool reports that there are no implementation results available, and ask if you would like the tool to generate those as well. Click **Yes** to continue.

8. In the Launch Runs dialog box, leave the default settings and select **OK**.

Synthesis and Implementation runs are kicked off in sequence to synthesize the netlist for the platform design, place and route the logic in the design, and finally generate the bitstream you requested.

9. After the Bitstream Generated dialog box is displayed, click **Cancel** to close it, and select the **File > Export > Export Hardware** command from the main menu.

10. Select the **Include Bitstream** checkbox, as shown below.



You now have a Zynq UltraScale+ MPSoC bitstream that includes a hardware design without any SDSoC accelerators. This bitstream can be used as a check of hardware functionality before running the design through the SDx IDE to generate hardware accelerators.

## ► Step 5: Writing Out the DSA

At this point, you can encapsulate the IP integrator hardware design, along with the PFM properties design metadata, into a Device Support Archive (DSA) that becomes part of the SDSoC platform definition.

1. In the Tcl Console, type the following command and press the `Enter` key.

```
write_dsa -force -include_bit /tmp/zcu102_board/zcu102_board.dsa
```

**NOTE:** If your project is named differently, you will need to modify the command to match your design.

2. Validate the DSA by entering the following command in the Tcl console.

```
validate_dsa /tmp/zcu102_board/zcu102_board.dsa
```

3. Close the Vivado GUI after validating DSA.

## Conclusion

In completing Lab 1, you have used the Vivado Design Suite to create a Zynq UltraScale+ MPSoC hardware design using Vivado IP integrator to target a ZCU102 board. In addition to the PS Zynq UltraScale+ MPSoC IP, you added key PL IP blocks for clocks, resets, and interrupts to define the base hardware platform. After declaring select hardware interfaces for use by the `sds++` system compiler, you created the DSA to define the hardware component of an SDSoC platform.

## Lab 2: Creating the SDSoC Platform

In Lab 1 you created the hardware component of the SDSoC platform: the DSA file which contains the framework for the Zynq UltraScale+ MPSoC hardware design. In this lab, you will create an SDSoC platform project to define the **zcu102\_board** platform, while also generating the elements of the software for a standalone (or baremetal) operating system. The required software components are:

- **First Stage Boot Loader (FSBL):** The FSBL initializes the processor (PS) block, configures the programmable logic (PL) with a bitstream if needed, and loads either a second stage boot loader, or

loads an application as specified in the boot file.

- **Linker Script:** The linker script specifies the memory address space used for the code and data regions present within the executable software application file. The executable file is formatted in the executable and linkable format (ELF). The stack and heap sizes are also specified within the linker script.
- **Boot Image Format (BIF) File:** The BIF file specifies how the Zynq® UltraScale MPSoC boot file (BOOT.BIN) is constructed. The boot file contains the FSBL and the executable application ELF file. It can also contain a bitstream as well as additional program and data files.



**NOTE** For more information on creating a boot image, see Zynq UltraScale+ MPSoC Software Developer Guide ([UG1137](#)).

## ► Step 1: Launching the SDx™ IDE

### On a Linux host machine:

At the shell prompt, type the following commands:

```
1. `source <Xilinx_Install_Directory>/SDx/<Version>/settings64.{sh,csh}`  
2. `sdx`
```

The first command sets the environment variables before launching the SDx IDE, and the second command launches the tool.

### On a Windows host machine:

For a Windows host machine, use one of the following methods to launch Vivado®

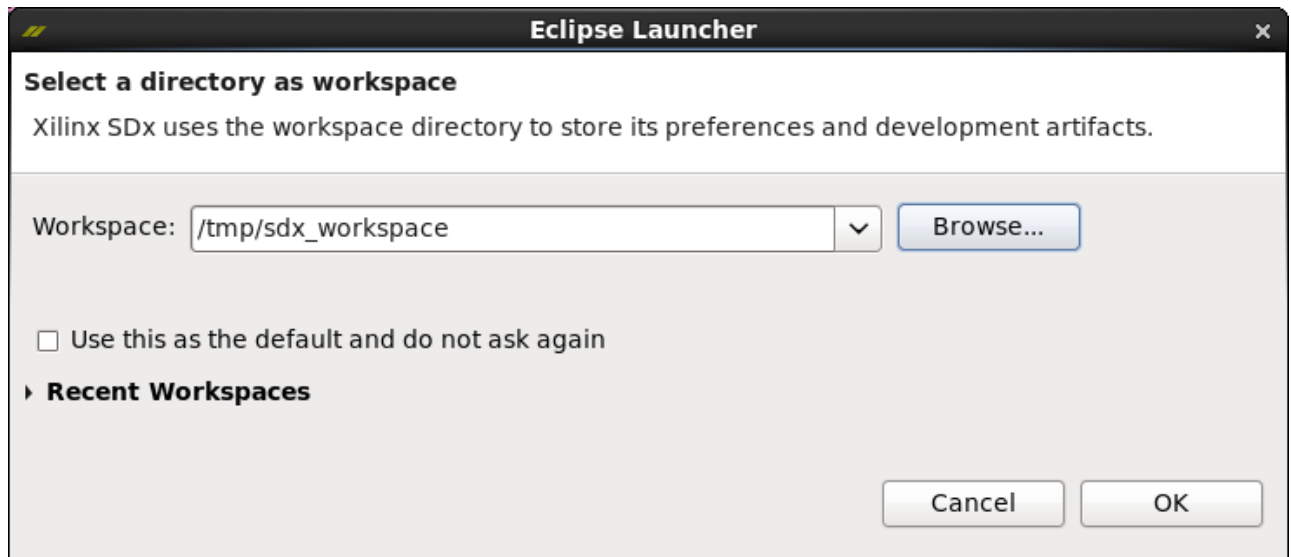
- Click the Vivado desktop icon
- From the Start menu, select Xilinx Design Tools > Vivado 2018.2 > Vivado 2018.2
- From a Command prompt window, type the following commands:

```
1. <Xilinx_Install_Directory>/SDx/<Version>/settings64.bat  
2. sdx
```

The first command sets the environment variables before launching the SDx IDE, and the second command launches the tool.

After the SDx IDE opens, you will be prompted to specify an SDx workspace. The SDx workspace will contain the platform and application projects you develop in the SDx tool. You can change the workspace when creating new platforms or application projects.

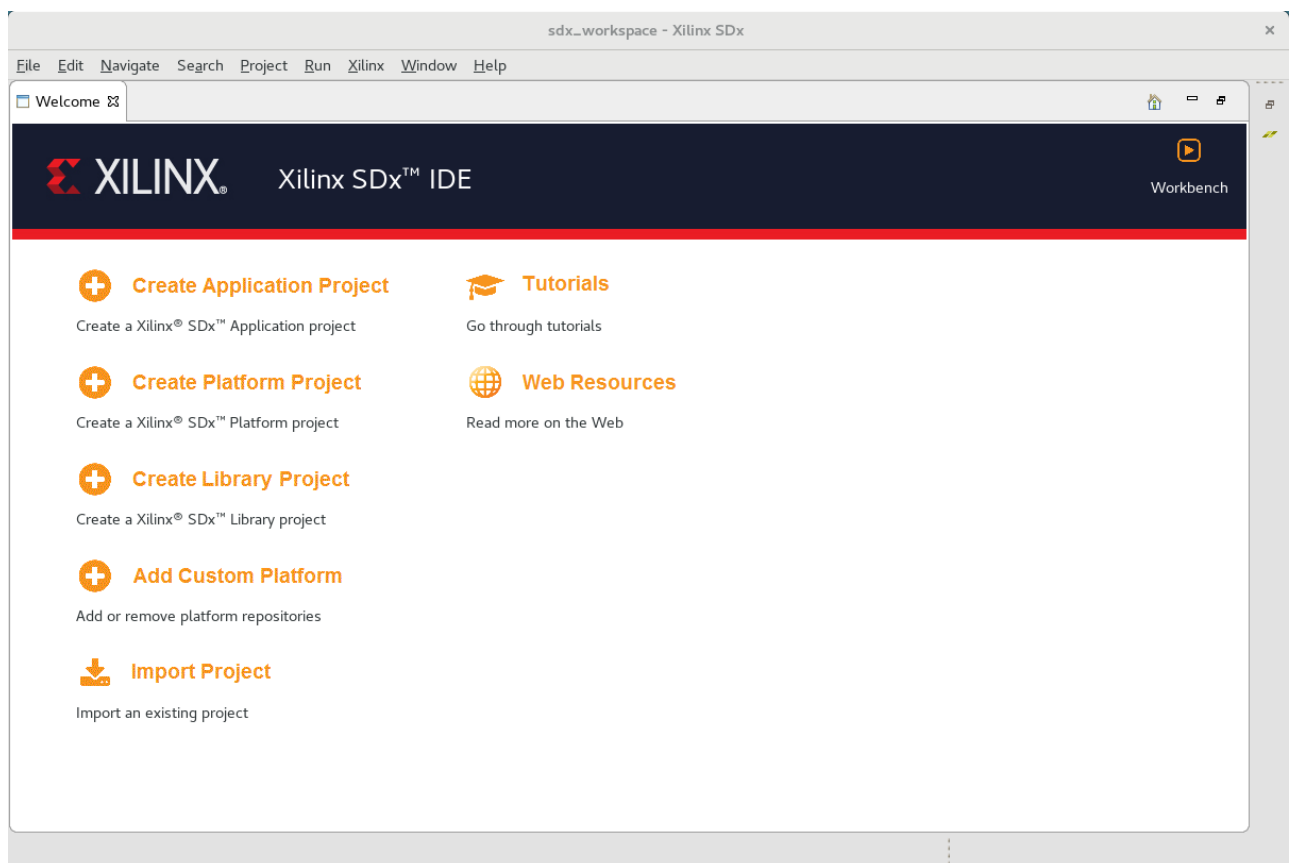
1. For this lab enter **/tmp/sdx\_workspace** for the Workspace as shown in the figure below.



2. Click **OK**.

3. In the SDx IDE Welcome screen, select **Create Platform Project**.

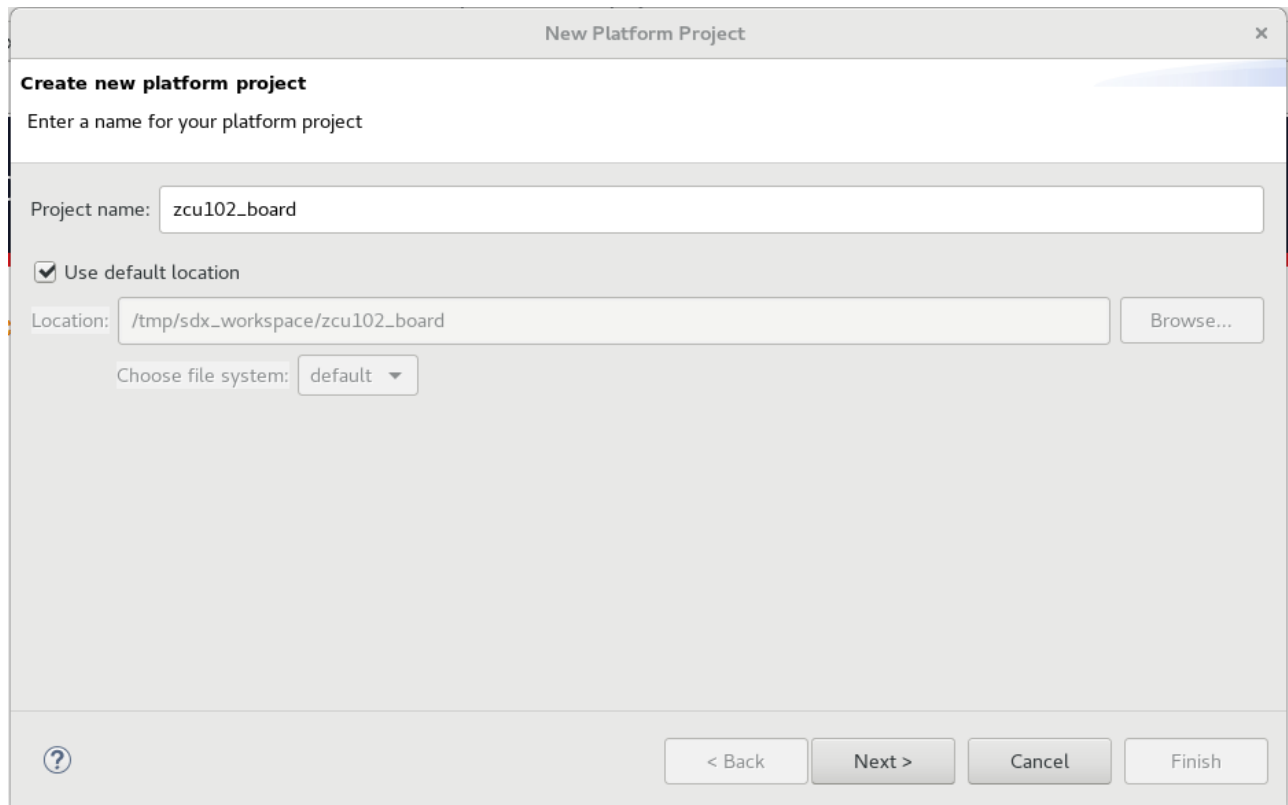
As an alternative, the SDx IDE menu selection **File > New > SDx Platform Project** can be used.



In this lab you will create the software components necessary for the SDSoC platform, and in the next lab you will combine the hardware and software components in a Platform project to define the SDSoC platform.

## ► Step 2: Creating a New Platform Project

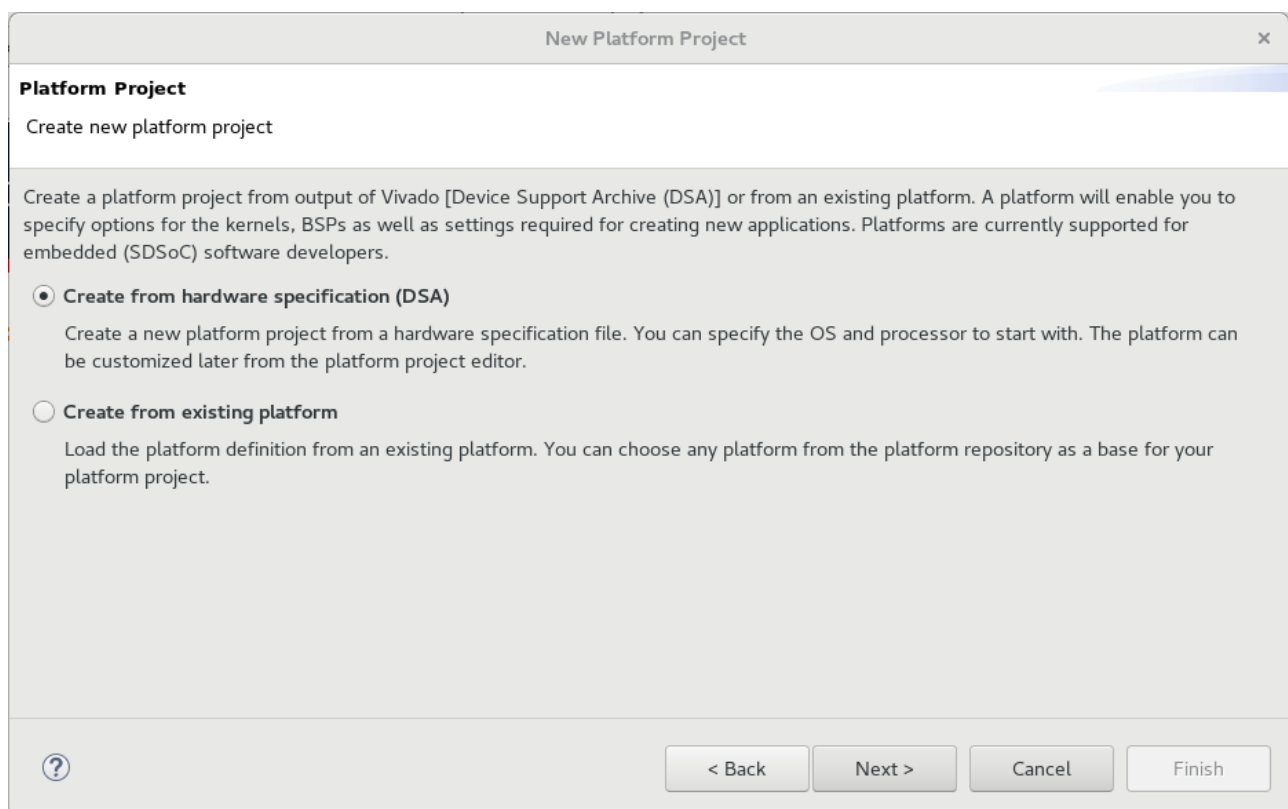
1. In the New Platform Project dialog, type **zcu102\_board** as the Project name.



The 'New Platform Project' dialog box is shown. It has a title bar with 'New Platform Project' and a close button. The main area is titled 'Create new platform project' with a subtitle 'Enter a name for your platform project'. Below this is a text field for 'Project name:' containing 'zcu102\_board'. There is a checked checkbox for 'Use default location'. Below that is a text field for 'Location:' containing '/tmp/sdx\_workspace/zcu102\_board' and a 'Browse...' button. A 'Choose file system:' dropdown menu is set to 'default'. At the bottom, there is a help icon, a '< Back' button, a 'Next >' button, a 'Cancel' button, and a 'Finish' button.

2. Click **Next**.

3. On the Platform dialog, select the **Create from hardware specification (DSA)** option.



The 'Platform Project' dialog box is shown. It has a title bar with 'New Platform Project' and a close button. The main area is titled 'Platform Project' with a subtitle 'Create new platform project'. Below this is a paragraph explaining that a platform project can be created from Vivado [Device Support Archive (DSA)] or from an existing platform. There are two radio button options: 'Create from hardware specification (DSA)' (selected) and 'Create from existing platform'. The 'Create from hardware specification (DSA)' option has a description: 'Create a new platform project from a hardware specification file. You can specify the OS and processor to start with. The platform can be customized later from the platform project editor.' The 'Create from existing platform' option has a description: 'Load the platform definition from an existing platform. You can choose any platform from the platform repository as a base for your platform project.' At the bottom, there is a help icon, a '< Back' button, a 'Next >' button, a 'Cancel' button, and a 'Finish' button.

4. Click **Next**.

5. Click the **Browse** button to add a **DSA file**.

6. Navigate to the **zcu102\_board.dsa** file written in Lab 1.

**Note:** It should be located at /tmp/zcu102\_board/zcu102\_board.dsa.

The SDx tool will read the selected DSA file and populate the Platform Project dialog box with available software specifications, as shown in the following figure.

7. In the Software Specification select:

- Operating system: **standalone**
- Processor: **psu\_cortex53\_0**

8. Click **Finish**.

### ► Step 3: Defining the System Configuration and Generating the Platform

The platform project is created, and the **Platform Configuration Settings** opens in the Editor area of the SDx IDE as shown in the figure below. The SDx tool automatically creates a system configuration, called **sysconfig1**, and processor domain with a name based on the operating system and processor you selected.








At this point, you will edit the platform project to add new configurations and domains.

1. Select the top-level platform project, **zcu102\_board**, in the Platform Configuration Settings tree view. You can edit the description for the platform. The fields of the platform project are edited by selecting the Edit command (pencil icon).
2. You can browse for sample application files to add to the ./samples folder of the platform. These files are optional, and the folder is not created unless you populate it.

3. Enable **Generate prebuilt data** to populate the prebuilt folder for the platform. You can also select **Use existing prebuilt data** to specify prebuilt data from another platform, for instance.




Providing a platform with prebuilt data containing software files with port interface specifications and a bitstream allows platform users to quickly compile and run software applications that do not invoke hardware accelerated functions.

#### System Configuration: sysconfig1

Display Name:	sysconfig1 		
Description:	System configuration for the zcu102 custom platform 		
Readme:	<input type="text"/>	Browse...	 
<input checked="" type="radio"/> <b>Generate software components</b> Choosing this option would generate boot artifacts, fsbl, bif file etc.			
<input type="radio"/> <b>Use pre-built software components</b>			
Boot Directory:	<input type="text"/>	Browse...	 
Bif File:	<input type="text"/>	Browse...	 

4. Select the **System configuration:sysconfig1** in the tree view, as displayed in the image above. You can enter a description for the system configuration by selecting the **Edit** command (pencil icon).
5. You can add a **Readme** file for the system configuration if you have one.
6. For this lab, enable the **Generate software components** radio button to have the SDx IDE automatically generate the files required for the current system configuration of the platform. Alternativley you can enable **Use pre-built software components** to specify the required files.

#### Domain: standalone\_domain

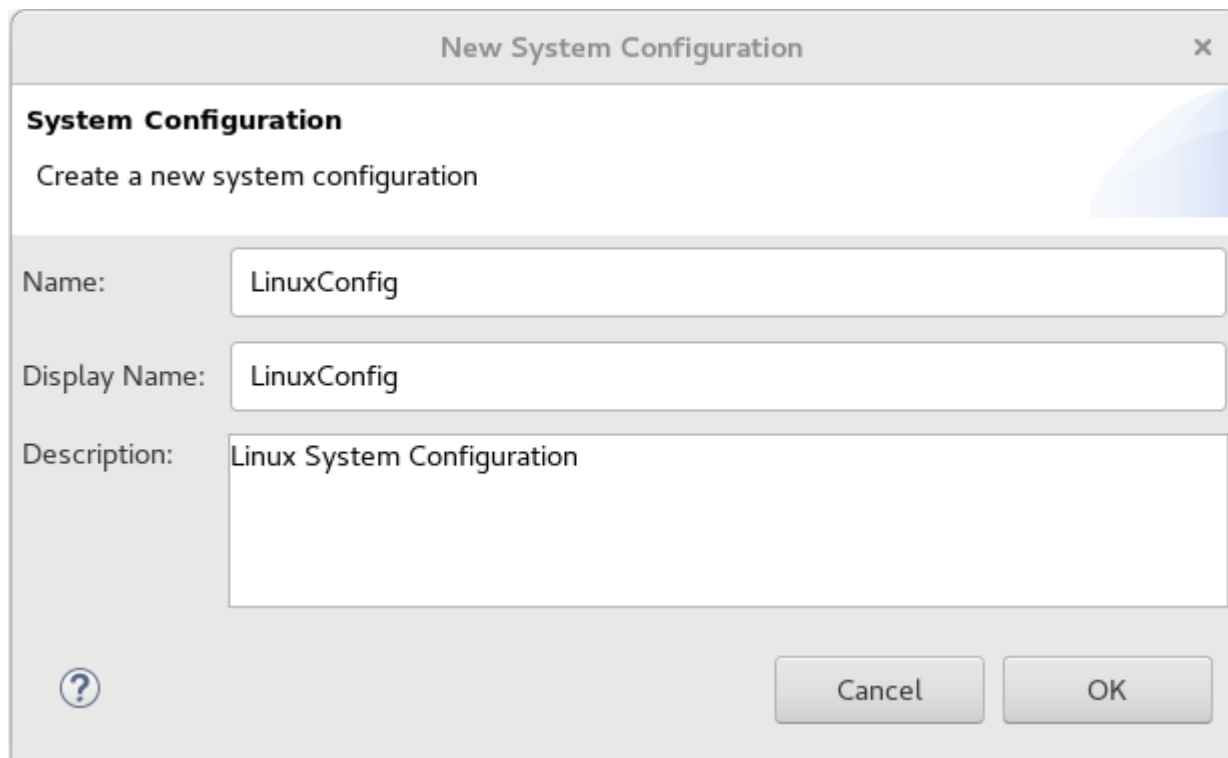
OS:	standalone		
Processor:	psu_cortexa53_0		
Supported Runtimes:	C/C++ ▼		
Display Name:	standalone on psu_cortexa53_0 		
Description:	standalone_domain 		
Repositories:	<input type="text"/>	Browse...	 
QEMU Data:	<input type="text"/>	Browse...	 
QEMU Arguments:	<input type="text"/>	Browse...	 
PMU QEMU Arguments:	<input type="text"/>	Browse...	 

7. Select the **standalone on psu\_cortexa53\_0** domain in the tree view, as displayed in the image above. This view is where you would specify the board support package, the application linker script, and included libraries. You will not make any changes to the selections on this page. Just examine the fields, and refer to the *SDSoC Environment Platform Development Guide* ([UG1146](#)) for more information.



You have just completed the first system configuration that was automatically defined from your choices when you set up the SDSoC platform project. However, in this tutorial, you will also be defining a second system configuration for the Linux operating system. This way, your custom platform can be used in standalone applications, or in Linux based applications.

1. In the Project Editor, select the Add command (green '+' icon) to add a new **System Configuration**.



This opens the **New System Configuration** dialog box, as shown above, letting you specify the Name, Display Name, and Description of the configuration. **Note:** Generating the software components for the Linux Operating System requires the PetaLinux tool to be installed and accessible to the SDSoC development environment. PetaLinux provides a complete, reference Linux distribution that has been integrated and tested for Xilinx devices. Refer to the *PetaLinux Tools Documentation: Reference Guide* ([UG1144](#)) for more information. After installing PetaLinux on your system, you must also set the **PetaLinux Install Location** in your SDSoC development environment, using the **Windows > Preferences** command, and selecting the **Xilinx SDx > Platform Project** in the Preferences dialog box.

2. Enter the following:

- Name: LinuxConfig
- Display Name: Linux Config
- Description: Add a suitable description.

3. Click **OK** to add the new system configuration to the platform project.

4. In the Project Editor, select the Add command (green '+' icon) to add a new **Domain**.

**New Domain in 'LinuxConfig'**

**Domain**  
Create a new domain.

Name:

Display Name:

OS:

Processor:

Supported Runtimes:

☐ Use pre-built software components

Boot Directory:

Bif File:

Note: These parameters will be applied to system configuration.

☒ Generate software components

This opens the **New Domain** dialog box, as shown above, letting you specify the Name, Display Name, and Description of the configuration.

5. Enter the following:

- Name: **linux\_domain**
- Display Name: **linux\_domain**
- OS: Select **linux** from the drop down menu.
- Processor: **psu\_cortexa53**
- Supported Runtimes: Select **C/C++**
- Enable the **Generate software components** radio button

6. Click **OK** to add the new domain to your platform project.

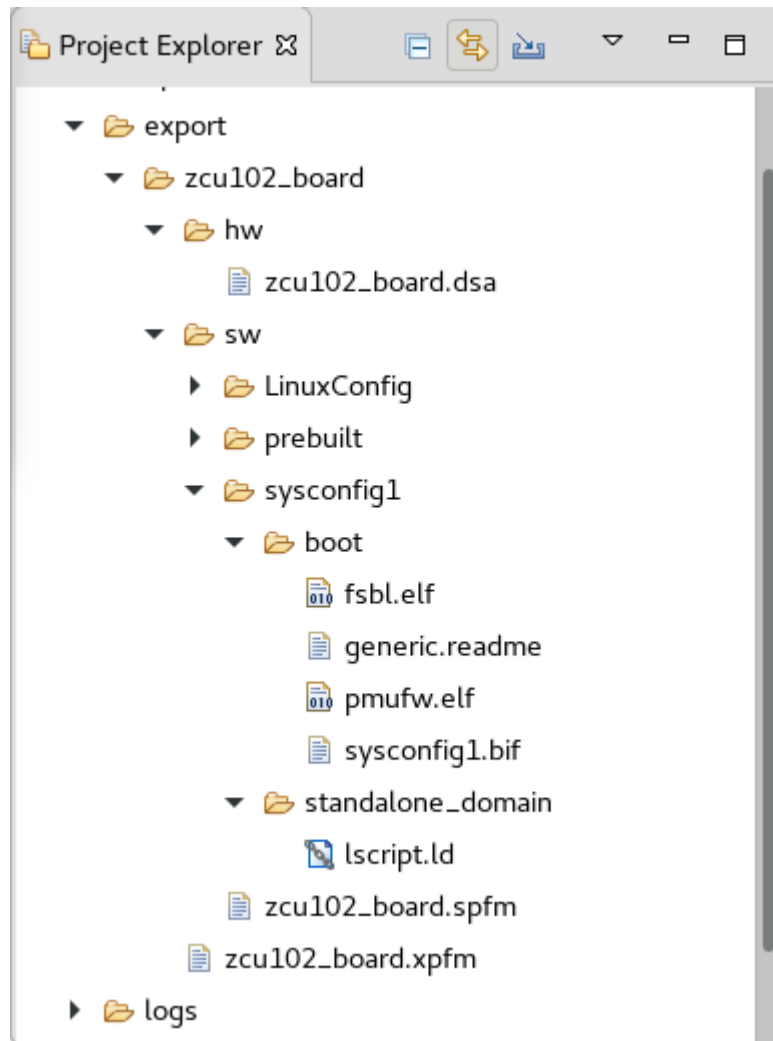
7. Select the **3: Generate Platform** command, under the Quick Links at the bottom of the page, to generate the platform files from the current project.

#### Quick Links

- 1** [Define System Configuration](#) **2** [Add Processor Group/Domain](#) **3** [Generate Platform](#)

The SDx tool compiles the ./sw folder to generate the required files for FSBL, linker script (lscript.ld), and the BIF file for each system configuration. In addition, prebuilt application files are generated to let you

test the platform if specified. Finally, the hardware platform DSA file from Lab 1 is copied into the ./hw folder in the platform. The contents of the exported platform are shown in the following figure.

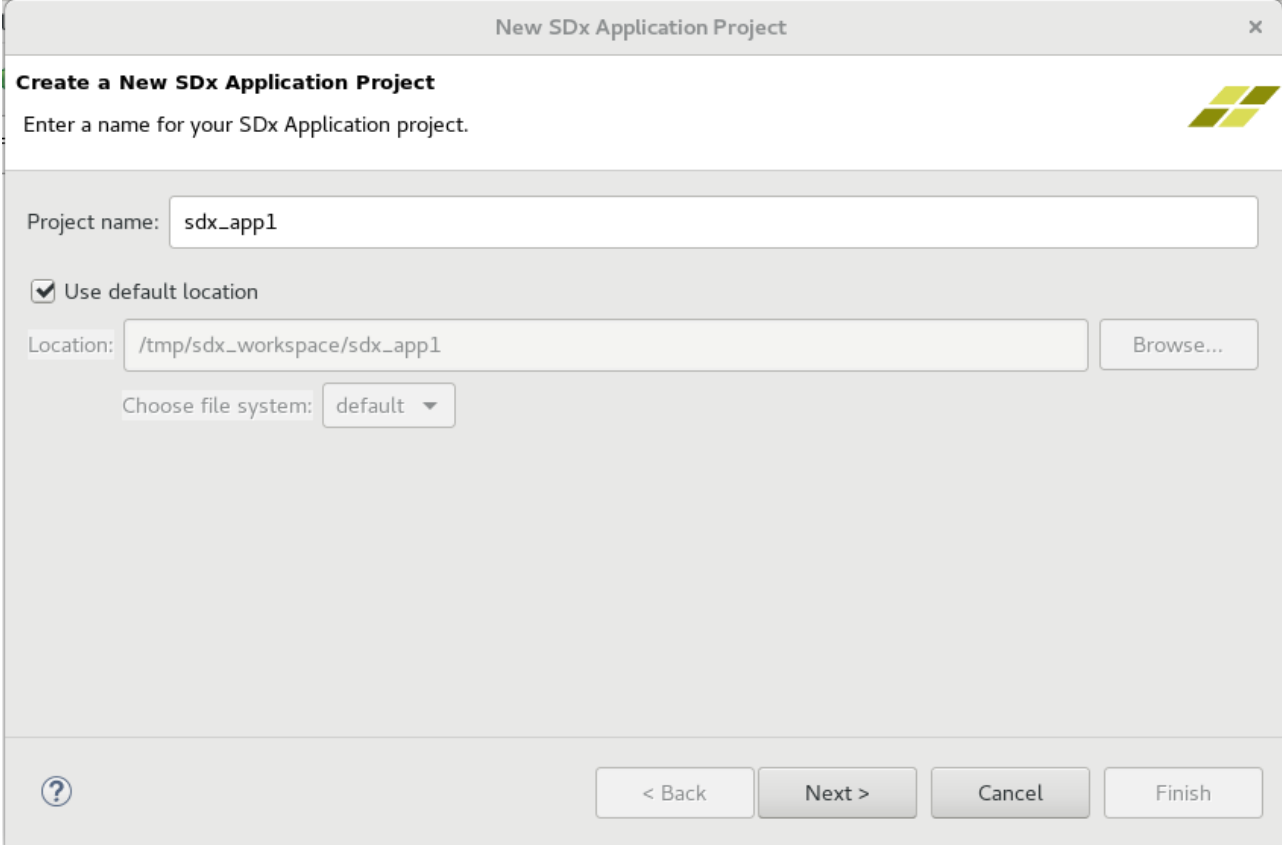


## Conclusion

In completing Lab 2, you used the DSA file you created in the Vivado Design Suite in Lab 1 and brought it into a Platform Project in the SDx environment. Using the SDx IDE, you specified a Standalone system configuration and a Linux system configuration for your platform. With the details of the platform specified, you generated the actual platform files for use in SDSoc application projects. In Lab 3 you will use the platform you have created.

## Lab 3: Using Your Custom Platform

1. On the SDx IDE menu, select **File > New > SDx Application Project** to begin creating a new Application project.
2. In the Create a New SDx Application Project dialog, type **sdx\_app1** as the Project name.
3. Click **Next**.



**New SDx Application Project**

**Create a New SDx Application Project**

Enter a name for your SDx Application project.

Project name:

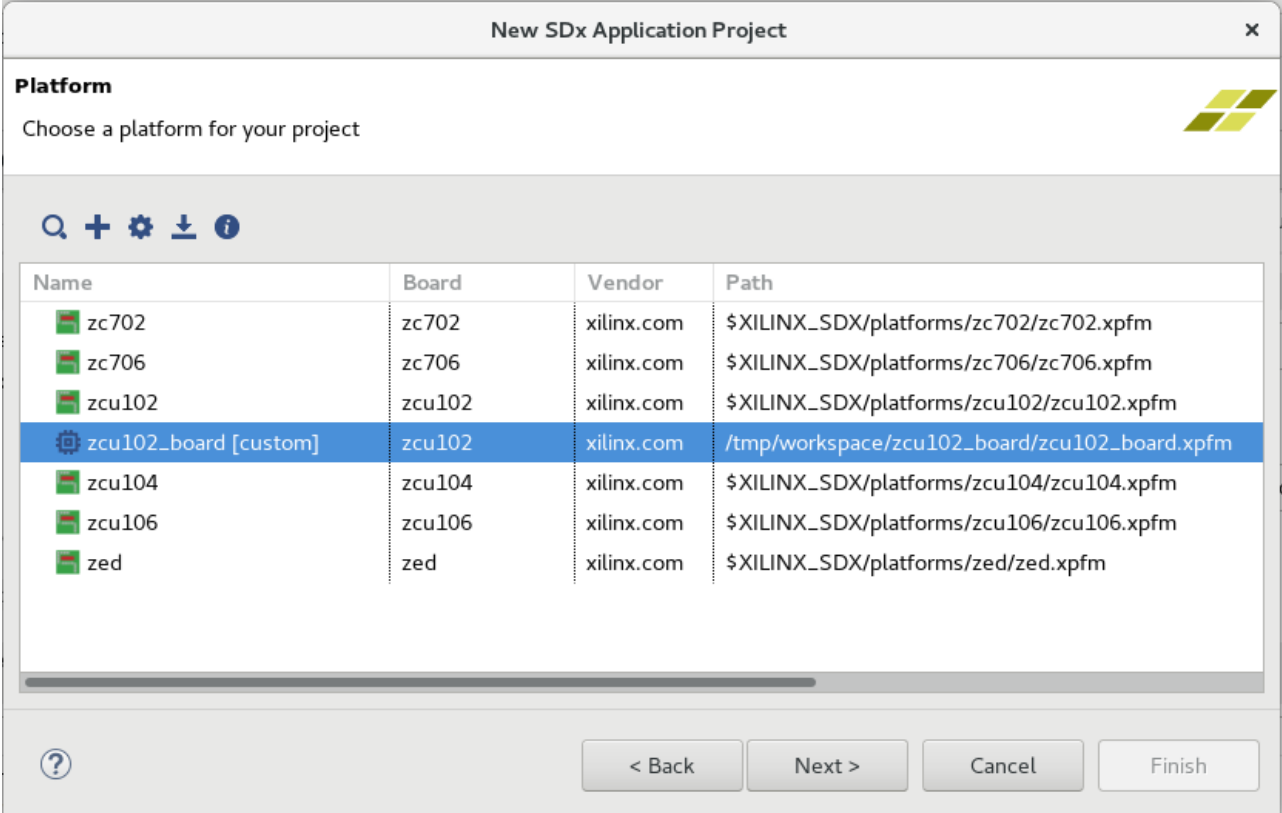
☒ Use default location

Location:

Choose file system:

4. In the Platform dialog, select **Add Custom Platform** command (the '+' icon).








This opens the Specify Custom Platform Location dialog box, letting you navigate to the platform folder. If you were following the instructions in this tutorial, you will find the platform at `/tmp/sdx_workspace/zcu102_board/export/zcu102_board`.



**New SDx Application Project**

**Platform**

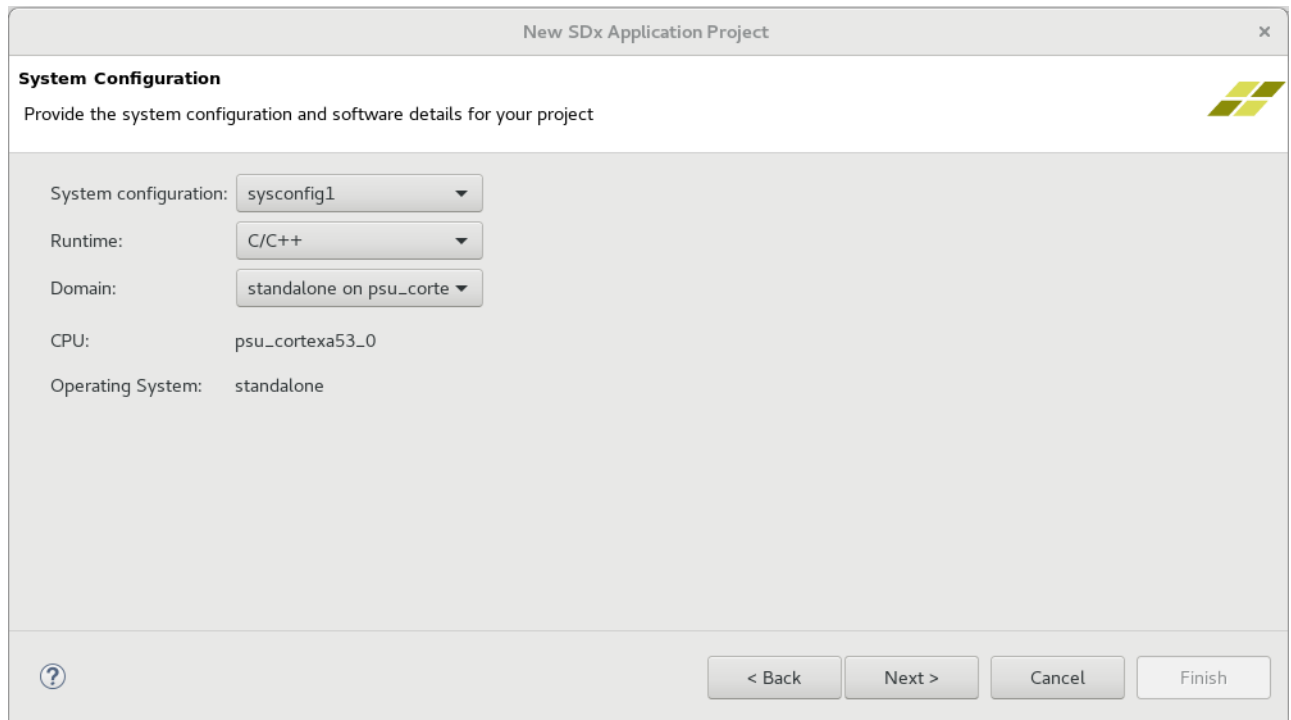
Choose a platform for your project

Name	Board	Vendor	Path
 zc702	zc702	xilinx.com	\$XILINX_SDX/platforms/zc702/zc702.xpfm
 zc706	zc706	xilinx.com	\$XILINX_SDX/platforms/zc706/zc706.xpfm
 zcu102	zcu102	xilinx.com	\$XILINX_SDX/platforms/zcu102/zcu102.xpfm
 zcu102_board [custom]	zcu102	xilinx.com	/tmp/workspace/zcu102_board/zcu102_board.xpfm
 zcu104	zcu104	xilinx.com	\$XILINX_SDX/platforms/zcu104/zcu104.xpfm
 zcu106	zcu106	xilinx.com	\$XILINX_SDX/platforms/zcu106/zcu106.xpfm
 zed	zed	xilinx.com	\$XILINX_SDX/platforms/zed/zed.xpfm

5. After adding the **zcu102\_board** platform, select it in the Platform dialog box, and click **Next**.

6. Accept the default settings in the System Configuration dialog box:

- System configuration: **sysconfig1**
- Runtime: **C/C++**
- Domain: **standalone on psu\_cortexa53**
- CPU: psu\_cortexa53\_0
- OS: standalone



The screenshot shows the 'New SDx Application Project' dialog box with the 'System Configuration' tab selected. The dialog has a title bar with the text 'New SDx Application Project' and a close button. Below the title bar, the tab is labeled 'System Configuration' with a subtitle 'Provide the system configuration and software details for your project'. The configuration fields are as follows:

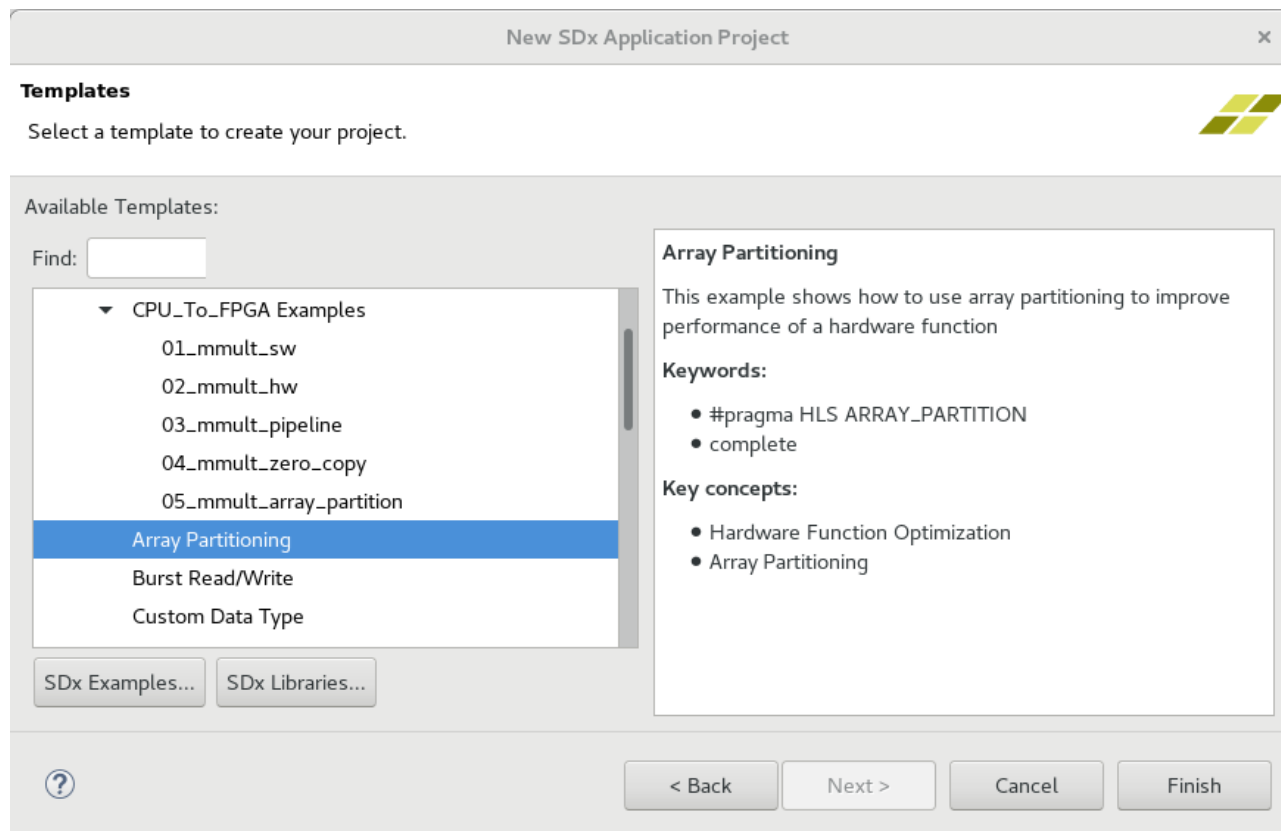
Field	Value
System configuration:	sysconfig1
Runtime:	C/C++
Domain:	standalone on psu_corte
CPU:	psu_cortexa53_0
Operating System:	standalone

At the bottom of the dialog, there is a help icon (question mark) on the left and four buttons: '< Back', 'Next >', 'Cancel', and 'Finish'.

To generate a Linux application, the LinuxConfig system can be selected.

7. Click **Next**.

8. On the Templates dialog, select **Array Partitioning** and click **Finish**.



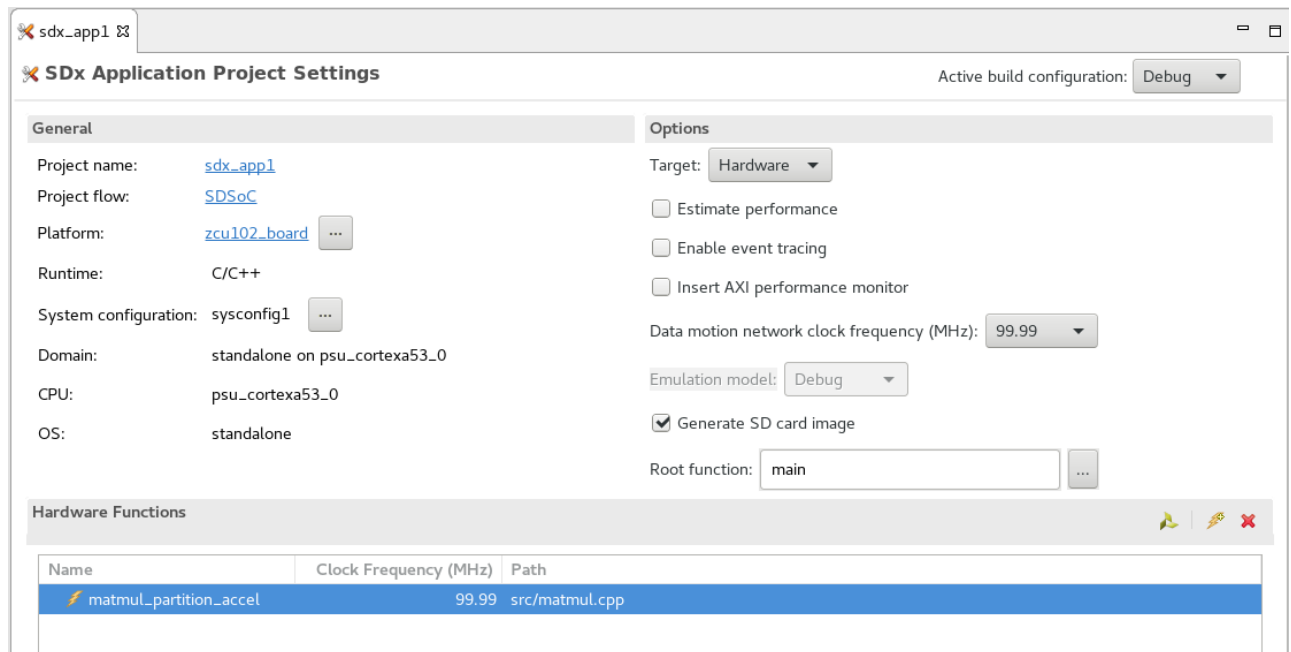
**Note:** To get or update the example applications, you can click the **SDx Examples** button. Refer to the *SDSoC Development Environment User Guide* (UG1027) for more information.

The newly created SDSoC application `sdx\_app1` is shown in the Project Explorer view and the Assistant view.

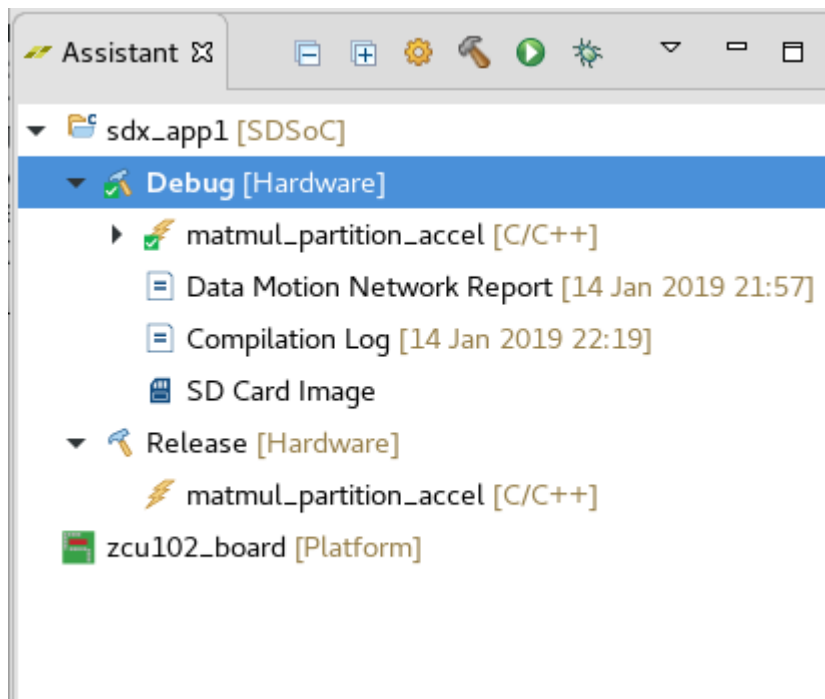
>:pushpin: **\*\*NOTE:\*\***

>The Assistant view shows a hardware accelerated function named `matmul\_partition\_accel` which is part of the Array Partitioning example.

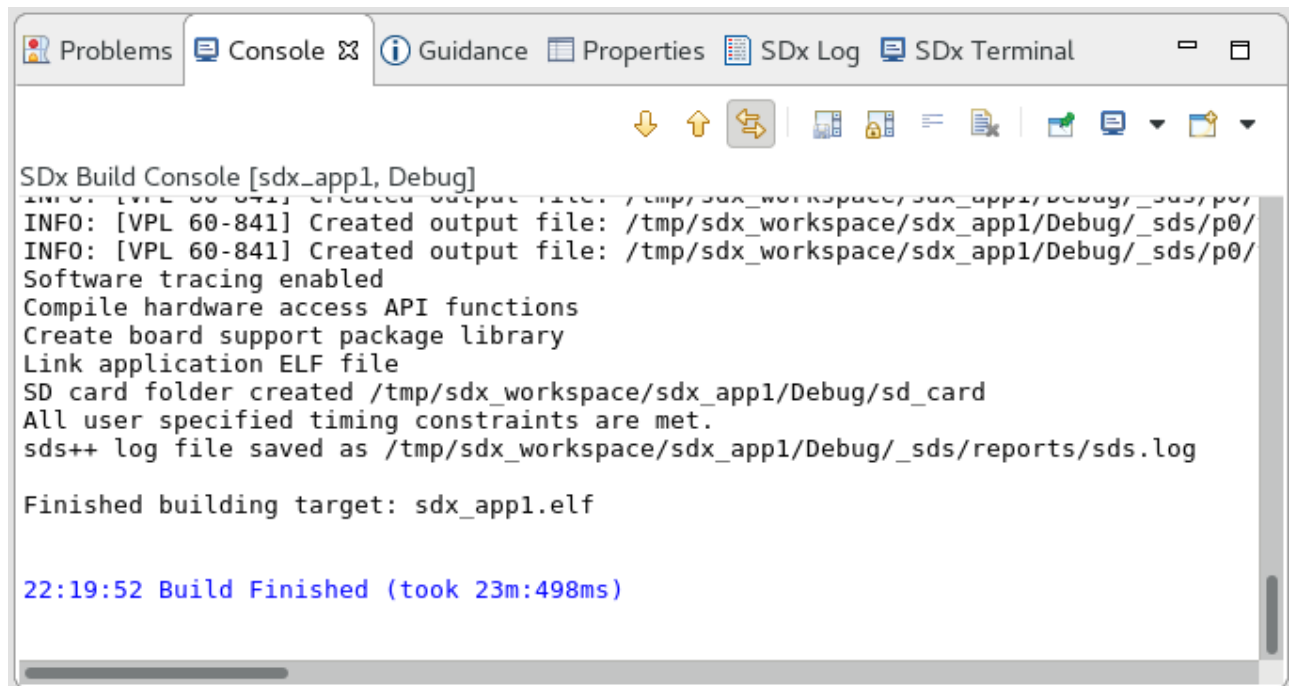
1. The Editor view shows settings for `sdx_app1` in the Application Project Settings window.



2. In the Assistant view, expand **sdx\_app1 [SDSoC]**, and right-click **Debug [Hardware]** and select **Build**.
3. The Assistant view provides build results through links to the following:
  - Compilation log
  - Data motion report for accelerator
  - Generated SD card image contents



4. You can also scroll through the Console window to view the transcript of the build process which is captured in the **sds.log**.

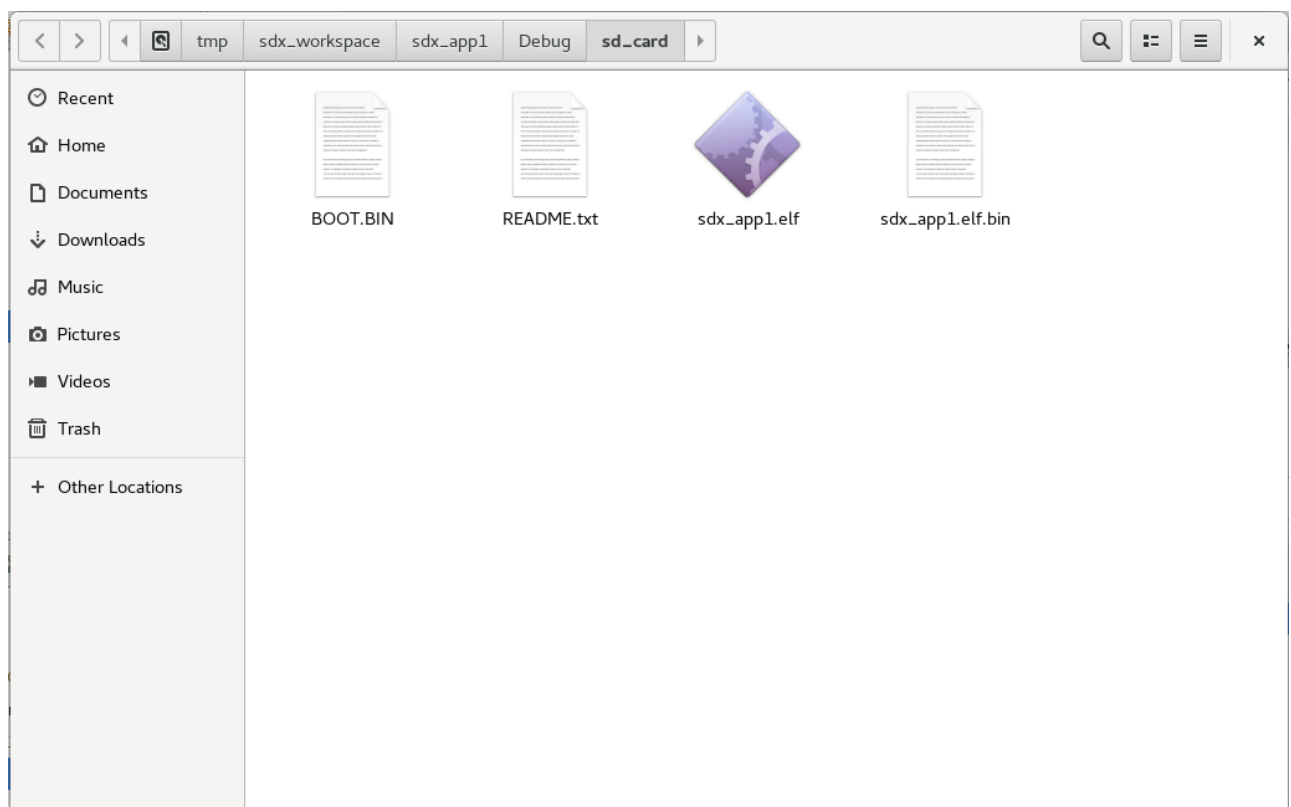


```
SDx Build Console [sdx_app1, Debug]
INFO: [VPL 60-841] Created output file: /tmp/sdx_workspace/sdx_app1/Debug/_sds/p0/
INFO: [VPL 60-841] Created output file: /tmp/sdx_workspace/sdx_app1/Debug/_sds/p0/
INFO: [VPL 60-841] Created output file: /tmp/sdx_workspace/sdx_app1/Debug/_sds/p0/
Software tracing enabled
Compile hardware access API functions
Create board support package library
Link application ELF file
SD card folder created /tmp/sdx_workspace/sdx_app1/Debug/sd_card
All user specified timing constraints are met.
sds++ log file saved as /tmp/sdx_workspace/sdx_app1/Debug/_sds/reports/sds.log

Finished building target: sdx_app1.elf

22:19:52 Build Finished (took 23m:498ms)
```

5. In the Assistant view, right-click **SD Card Image** and select **Open > Open in File Browser** to open a view to the SD Card contents on the disk. The `sd_card` directory is located in the SDx workspace at `/tmp/sdx_workspace/sdx_app1/Debug/sd_card`.



You can copy the SD card directory contents to the root directory of a FAT32 formatted SD card and boot a ZCU102 platform using this SD card to run and view the sdx\_app1 UART output on a terminal program.

## Conclusion



In completing Lab 3, you have successfully created a custom SDSoC platform that targets the Zynq UltraScale+ MPSoC with a standalone software runtime environment. You have also built the SDSoC array partitioning example on top of the custom SDSoC platform (zcu102\_board).

End Tutorial

---

Copyright© 2018 Xilinx