

# SDSoC Environment Tutorial

## *Platform Creation*

UG1236 (v2017.4) January 26, 2018



# Table of Contents

<b>Revision History.....</b>	<b>5</b>
<b>Chapter 1: SDSoC Platform Creation Tutorial.....</b>	<b>7</b>
Hardware and Software Requirements.....	7
Tutorial Design Descriptions.....	7
Locating Tutorial Design Files.....	9
<b>Chapter 2: Lab 1: Creating DSA for a Zynq-7000 AP SoC Processor</b>	
<b>Design.....</b>	<b>11</b>
Step 1: Start the Vivado IDE and Create a Project.....	11
Step 2: Create an IP Integrator Design.....	13
Step 3: Setting Platform Properties.....	26
Step 4: Generating HDL Design Files.....	28
Step 5: Writing Out the DSA.....	29
Conclusion.....	29
Lab Files.....	30
<b>Chapter 3: Lab 2: Creating Software Components for the</b>	
<b>Platform.....</b>	<b>31</b>
Step 1: Launching SDK.....	31
Step 2: Creating an FSBL Application.....	32
Step 3: Generating the Boot Image File (BIF).....	35
Step 4: Creating a New Software Application and a Linker Script.....	38
Step 5: Copying and Editing Files.....	43
Conclusion.....	44
Lab Files.....	44
<b>Chapter 4: Lab 3: Creating a Custom Platform Using the SDx IDE....</b>	<b>45</b>
Step 1: Invoking the SDx IDE and Creating a Project.....	45
Step 2: Defining System Configuration.....	47
Step 3: Adding Processor Domain.....	48
Step 4: Generating the Platform.....	50

Step 5: Adding Custom Platform to Repository.....	51
Step 6: Creating an SDx Application Targeting the Custom Platform.....	52
Conclusion.....	57
Lab Files.....	58
<b>Appendix A: Additional Resources and Legal Notices.....</b>	<b>59</b>
References.....	59
Please Read: Important Legal Notices.....	60

# Revision History

The following table shows the revision history for this document.

Date	Version	Revision
01/26/2018	2017.4	Minor updates to screenshots
12/20/2017	2017.4	Initial Release



# SDSoC Platform Creation Tutorial

This tutorial shows how to create an SDSoC™ platform that can be used to accelerate software functions using the SDx™ Integrated Design Environment (IDE) tool. An SDSoC platform defines a base hardware and software architecture and application context, including processing system, external memory interfaces, custom input/output, and software runtime - including operating system (possibly "bare metal"), boot loaders, drivers for platform peripherals and root file system. Every project you create within the SDx environment Integrated Design Environment (IDE) targets a specific hardware platform, and you employ the tools within the SDx IDE to customize the platform application-specific hardware accelerators and data motion networks. In this way, you can easily create highly tailored application-specific systems-on-chip for different platforms, and can reuse platforms for many different application-specific systems-on-chip.

In this tutorial, you use the Vivado® Design Suite to build a hardware system, and create a platform using the software runtime environment, including an operating system kernel, boot loaders, file system, and libraries.

---

## Hardware and Software Requirements

This tutorial requires that you have the SDx Integrated Design Environment installed. See the *SDx Environments Release Notes, Installation, and Licensing Guide* ([UG1238](#)) for instructions. The SDx tool installs the SDx IDE as well as the required Vivado Design Suite and Software Development Kit (SDK) software.

---

## Tutorial Design Descriptions

No design files are required for these labs, if step-by-step instructions are followed as outlined; however, for subsequent iterations of the design or to build the design quickly, Tcl command files for these labs are provided in the zip file in [Locating Tutorial Design Files](#). For cross-probing hardware and software, manual interaction with Vivado® and Platform boards is necessary. No Tcl files are provided for that purpose.

In this tutorial, users will be guided step-by-step through a series of three labs where:

1. A Device Support Archive (DSA) is created to define the hardware component for the platform.
2. A First Stage Boot Loader (FSBL), Boot Image File (BIF), and Linker Script are created to define the software components for the platform.
3. Finally, the hardware and software components are tied together in the SDx IDE to create a SDx Platform.

## Lab 1: Creating DSA for a Zynq-7000 AP SoC Processor Design

[Chapter 2: Lab 1: Creating DSA for a Zynq-7000 AP SoC Processor Design](#) uses the Zynq®-7000 AP SoC Processing System (PS) IP, and a Clocking Wizard IP to generate several clocks for potential use in the DSA. Multiple Processor System Reset blocks are used to synchronize the resets to these different clock sources. The Lab uses the following IP in the Programmable Logic (PL):

- Clocking Wizard
- Processor System Reset
- ZYNQ7 Processing System
- Concat

Lab 1 shows how to graphically build a design in the Vivado® IP integrator and use the Designer Assistance feature to configure and connect the Zynq processor to the available interfaces on the ZC702 board. Clocking Wizard and Processor System Reset blocks are then added and connected to complete the design.

After you construct a design, you add properties to the design using Tcl to encapsulate hardware-related information needed to create an SDSoC platform. Finally, the Device Support Archive (DSA) is written.

The `lab1.tcl` design files are included in the zip file for this guide

See [Locating Tutorial Design Files](#).

## Lab 2: Creating Software Components for the Platform in SDK

[Chapter 3: Lab 2: Creating Software Components for the Platform](#) requires that you have the Software Development Kit (SDK) software installed on your machine.

In Lab 2, you use the SDK software to build the boot image, the Boot Image File (BIF), First Stage Boot Loader (FSBL), and the Linker Script file that are needed to create a platform.

If users want to skip over the SDK portion of the lab (Lab 2), four files are provided to the user for use in Lab 3. These files are:

- BOOT.bin



- fsbl.elf
- lscript.ld
- standalone.bif

### Lab 3: Creating a Custom Platform Using the SDx IDE

In this lab you use the SDx™ IDE to create a platform that uses the DSA you created in Lab 1 and the software components you created in Lab 2, and then generate the platform. The platform is then added to the SDx workspace so new applications can be targeted to it.

No design files are provided for this step as the intent is for you to use the SDx IDE to generate the platform.

---

## Locating Tutorial Design Files

Design data is in the associated [Reference Design File](#).



# Lab 1: Creating DSA for a Zynq-7000 AP SoC Processor Design

In this lab you create a Zynq®-7000 AP SoC processor-based design and instantiate IP in the programmable logic (PL) to complete your design. Then you set properties on the block design to identify, clocks, resets, interrupts, and so forth, and finally write out the Device Support Archive (DSA). The custom hardware platform created in this lab has been targeted to a ZC702 platform, although it is not necessary that a predefined board in Vivado® be used to create the DSA.

If you are not familiar with the Vivado Integrated Development Environment (IDE), see the *Vivado Design Suite User Guide: Using the Vivado IDE* ([UG893](#)).

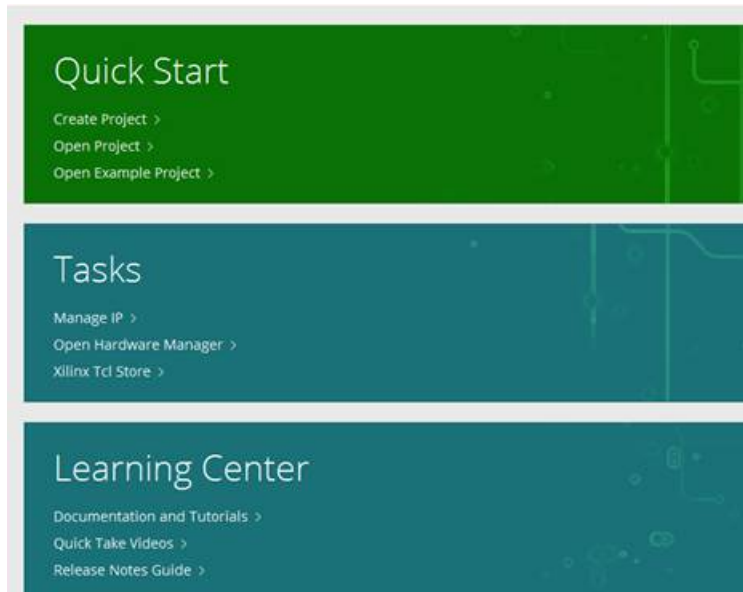
**Note:** You can use the Tcl file lab1.tcl that is included with this tutorial design files to perform all the steps in this lab. To use the Tcl script, launch Vivado and type `source lab1.tcl` in the Tcl console.

---

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

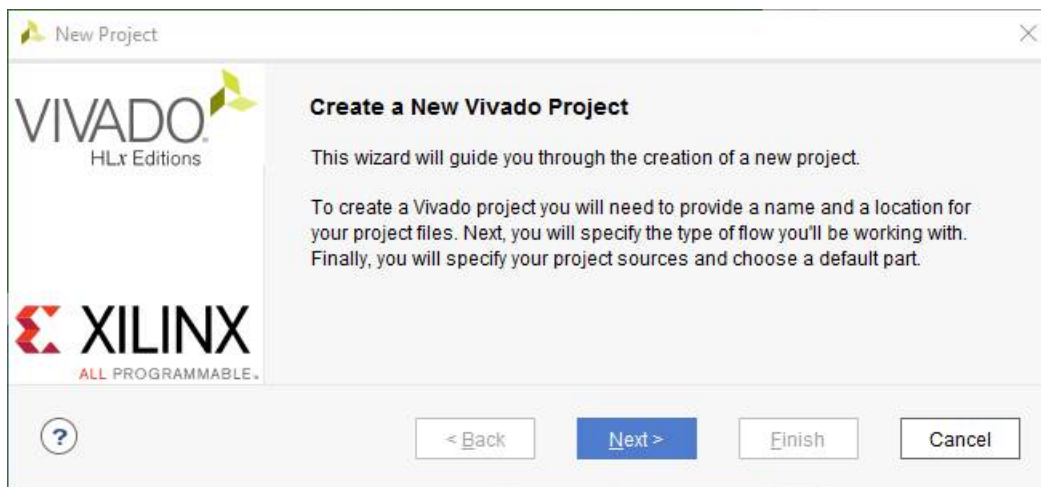
1. Start the Vivado® Design Suite IDE by clicking the Vivado desktop icon. You can also start Vivado by selecting the **Start** menu of your computer. Vivado can also be launched from a command prompt by sourcing the settings file by typing `C:\Xilinx\SDx\<sdx version>\settings64.bat`. Then you can type **vivado** at the prompt to launch Vivado.
2. From the **Quick Start** section, click **Create Project**, as shown in the the following figure:

Figure 1: Vivado Quick Start Page



The New Project wizard opens.

Figure 2: Create New Project Wizard



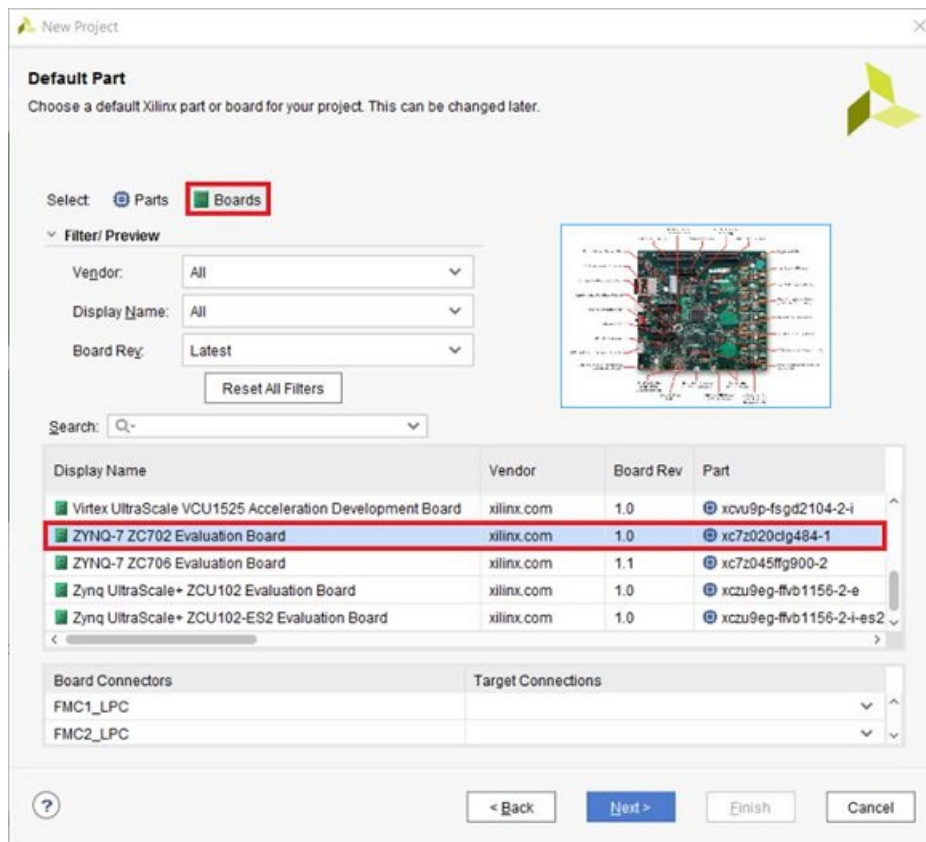
3. Click **Next**.

The Project Name page opens.

4. In the Project Name page, type the project name `zynq7_board` and select a location for the project files. Ensure that the **Create project subdirectory** check box is checked, and then click **Next**.
5. In the Project Type page, select **RTL Project**, and then click **Next**.

6. In the Add Sources page, set the Target language to your desired language, and the Simulator language to **Mixed**. Click **Next**.
7. In the Add Constraints page, click **Next**.
8. In the Default Part page, click the **Boards** icon. From the available boards, select the **ZYNQ-7 ZC702 Evaluation Board**, as shown in the following figure.

Figure 3: New Project Wizard: Default Part Page



**CAUTION!:** Multiple versions of boards are supported in Vivado. Ensure that you are targeting the design to the right hardware.

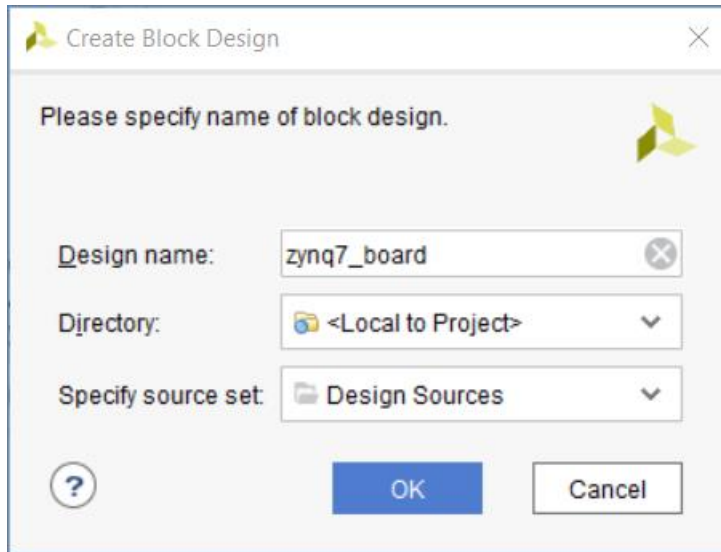
9. Click **Next**.
10. Review the project summary in the New Project Summary page, and then click **Finish** to create the project.

## Step 2: Create an IP Integrator Design

1. In the Flow Navigator → IP Integrator, select **Create Block Design**.

2. In the Create Block Design dialog box, specify a name for your IP subsystem design such as `zynq7_board`. Leave the Directory field set to the default value of `<Local to Project>`, and leave the Specify source set field to its default value of **Design Sources**, as shown in the following figure:

Figure 4: Create Block Design Dialog Box



**IMPORTANT!:** You need not name the the IP integrator block design the same as the Vivado project, or the SDSoC platform, unless the Vivado project contains multiple block designs. In this case, the block design containing the SDSoC platform design must have the same name as that of the SDSoC platform.


3. Click **OK**.

## Adding IP to the Block Design

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

Alternatively, you can click the **Add IP** button in the IP integrator canvas, as shown in the following figure.

Figure 5: Add IP Link in IP Integrator Canvas

This design is empty. Press the  button to add IP.

The IP catalog opens.

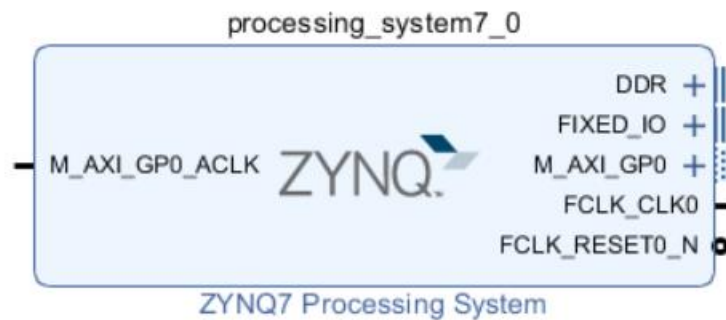
2. In the search field, type **zynq** to find the ZYNQ7 Processing System IP.
3. In the IP catalog, select the ZYNQ7 Processing System, and press **Enter** on the keyboard to add it to your design.

In the Tcl Console, you see the following message:

```
create_bd_cell -type ip -vlnv xilinx.com:ip:processing_system7:5.5
processing_system7_0
```

The following figure shows the Zynq7 processing system diagram that shows in the IP integrator canvas.

Figure 6: **Zynq7 Processing System IP**

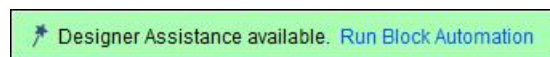


There is a corresponding Tcl command for most actions performed in the IP integrator block design. Those commands are not shown in this document; instead, the tutorial provides Tcl scripts to run this lab.

**Note:** Tcl commands are documented in the *Vivado Design Suite: Tcl Command Reference Guide* ([UG8835](#)).

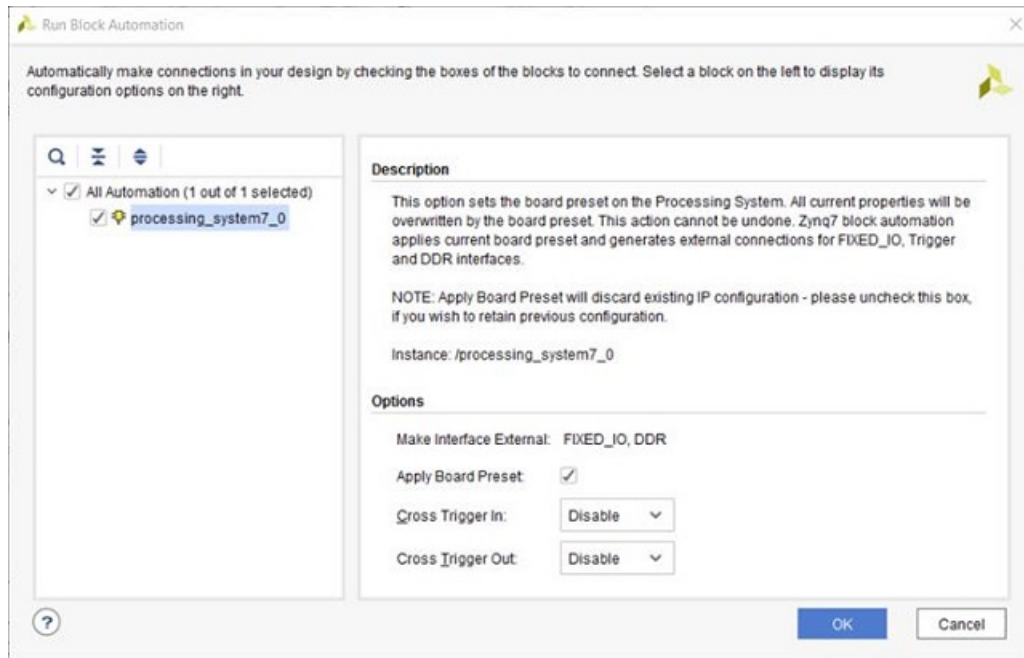
4. In the IP integrator window, click the **Run Block Automation** link, shown in the following figure.

Figure 7: **Run Block Automation link**



The Run Block Automation dialog box opens, as shown in the below figure. This dialog box states that the FIXED\_IO and DDR interfaces will be created for the Zynq®-7000 AP SoC IP core. The **Apply Board Preset** field is also checked by default.

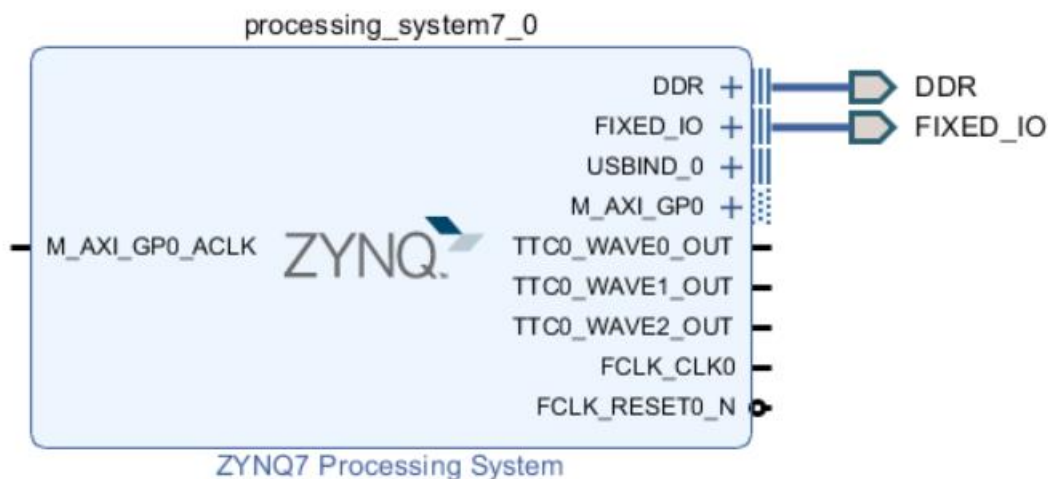
Figure 8: Zynq-7 Run Block Automation Dialog Box



5. Click **OK**.

After running block automation on the Zynq-7000 AP SoC processor, the IP integrator diagram looks like the following figure.

Figure 9: Zynq-7000 AP SoC Processing System After Running Block Automation



6. Now you can add peripherals to the PL. Right-click in the IP integrator diagram, and select **Add IP**.
7. In the search field, type **proc sys res** to find the Processor System Reset, and then press **Enter** to add it to the design.



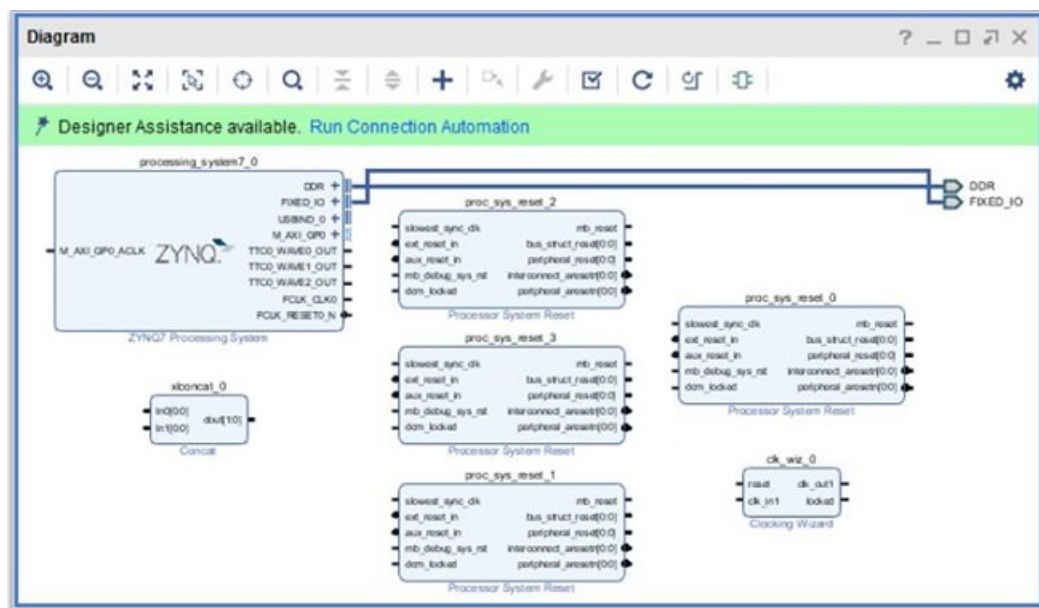
8. Repeating the previous two steps, add three more instances of the Processor System Reset IP.
9. Similarly, add a Clocking Wizard.
10. Finally, add a Concat IP.

Your Block Design window looks similar to the following figure. The relative positions of the IP might vary.



**TIP:** You can zoom in and out in the Diagram Panel using the **Zoom In** ( or **Ctrl+=**) and **Zoom Out** ( or **Ctrl+-**) tools.

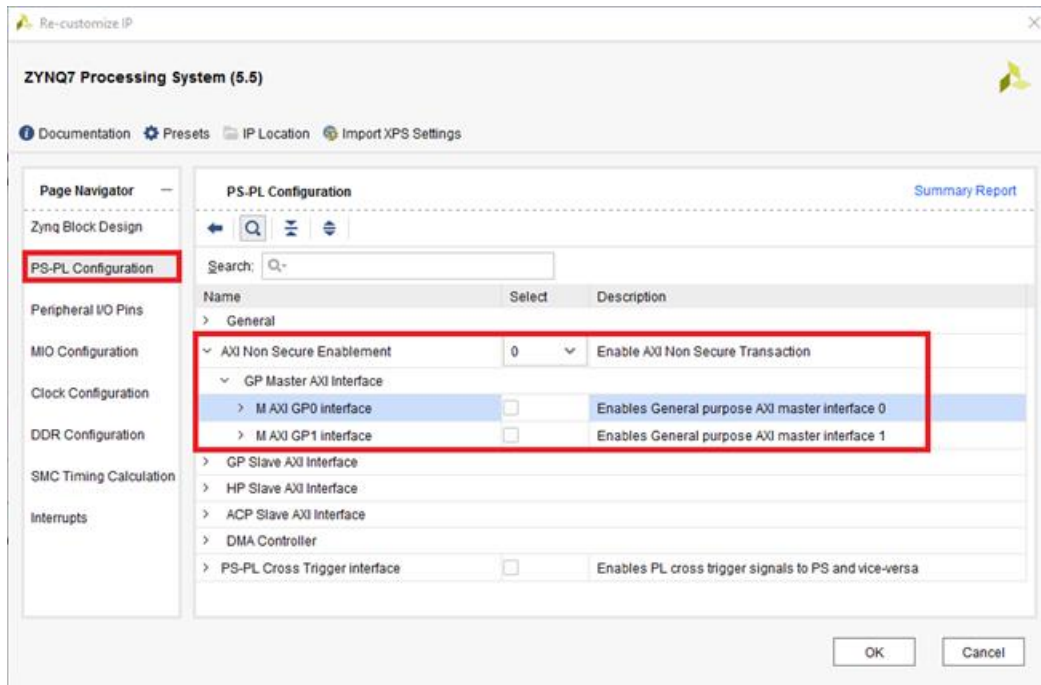
Figure 10: Block Design After Instantiating IP



## Re-Customizing IP

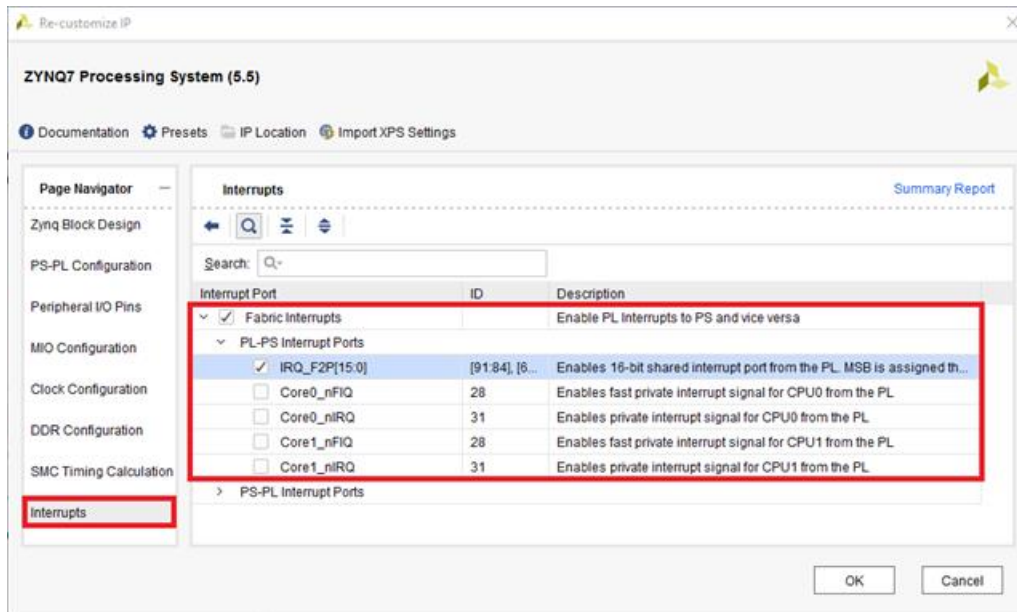
1. Double-click the **Zynq** IP to bring up the customization wizard.
2. In the Re-customize IP dialog box, shown in the following figure, do the following:
  - Select **PS-PL Configuration** in the Page Navigator.
  - Expand the **AXI Non Secure Enablement** drop-down menu.
  - Expand the **GP Master AXI Interface** menu.
  - Uncheck the **M AXI GPO interface** checkbox.

Figure 11: Disable M AXI GP0 Interface on Zynq-7000 AP SoC Processing System



3. In the Page Navigator, do the following, as shown in the following figure.
  - Select **Interrupts**.
  - Check the **Fabric Interrupts** check box.
  - Expand the **Fabric Interrupts** drop-down menu.
  - Expand the **PL-PS Interrupt Ports** drop-down menu and check the **IRQ\_F2P[15:0]** check box.

Figure 12: Enable Interrupts on Zynq-7000 AP SoC Processing System

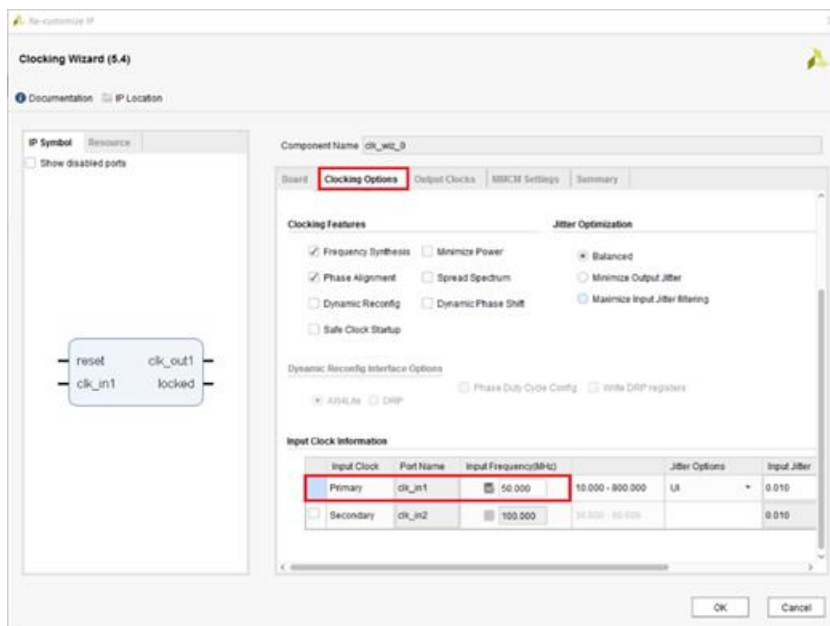


4. Click **OK**.

Next, you re-customize the Clocking Wizard IP, as shown in the following figure.

- Double-click the **Clocking Wizard** IP to bring up the customization dialog box.
- In the Clocking Options tab of the Re-customize IP dialog box, click the **Input Frequency** check box for Primary Clock and change the value in the adjacent box to **50.000**.

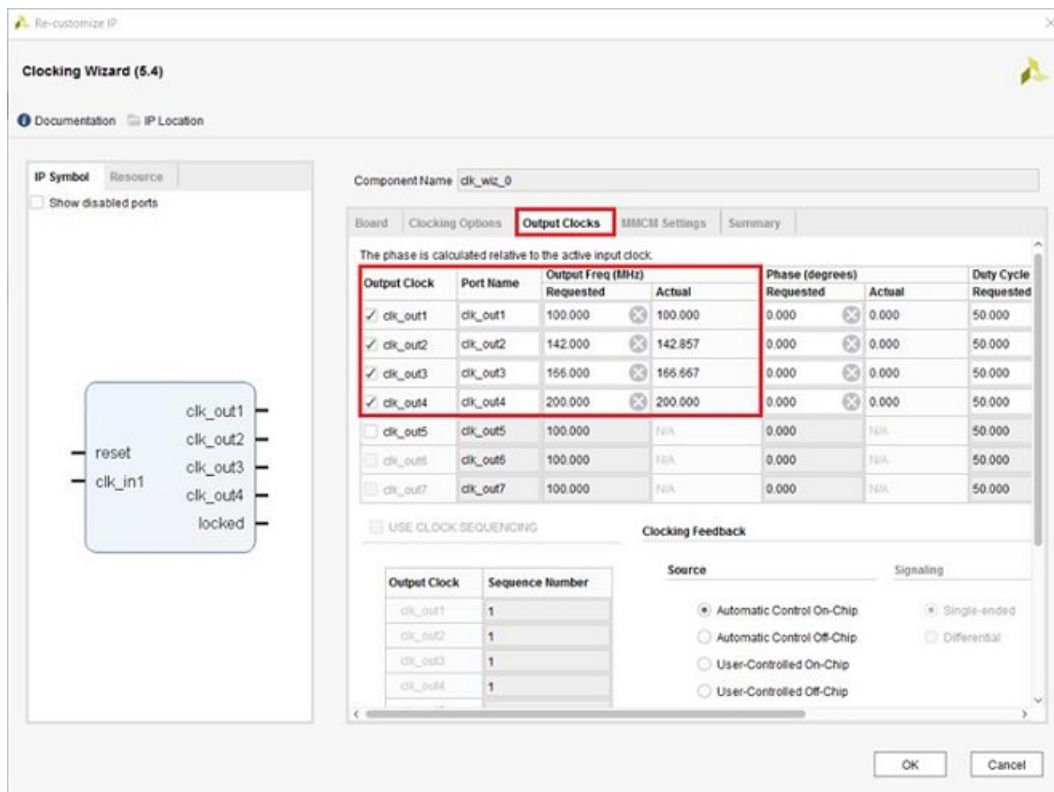
Figure 13: Change Input Frequency of the Primary Input Clock



7. Select the Output Clocks tab, check the check-boxes for **clk\_out2**, **clk\_out3**, and **clk\_out4** and change the output frequencies under the **Requested** column as shown in the following figure. The frequencies are as follows:

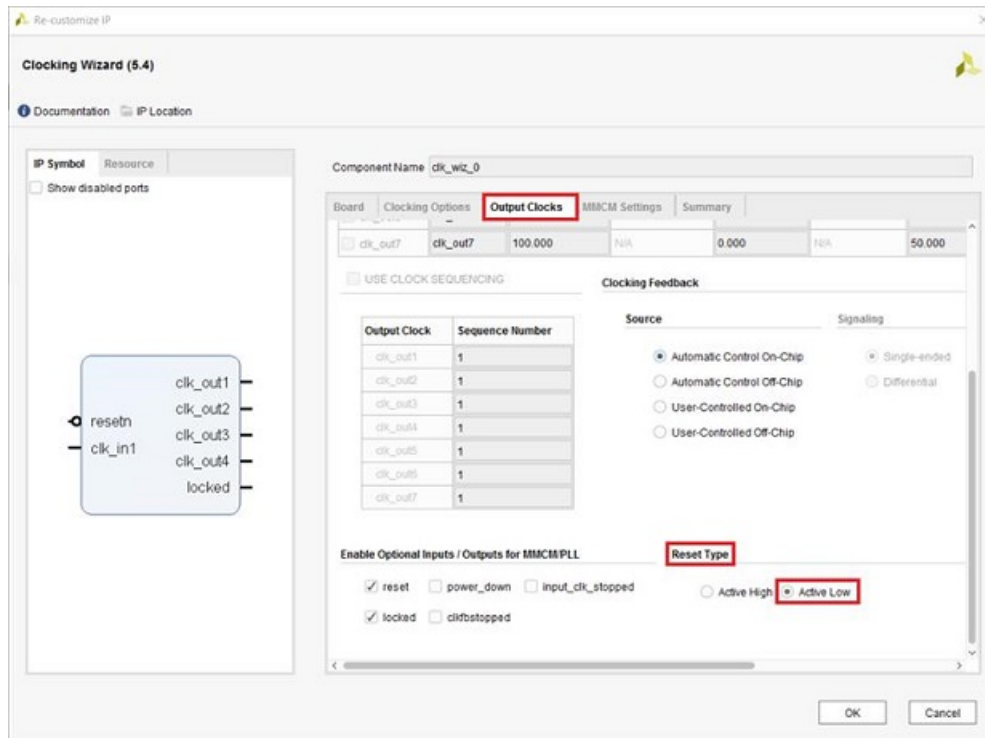
- clk\_out1 – 100
- clk\_out2 – 142
- clk\_out3 – 166
- clk\_out4 – 200

**Figure 14: Enable Output Clocks and Change the Desired Output Frequencies of the Output Clocks**



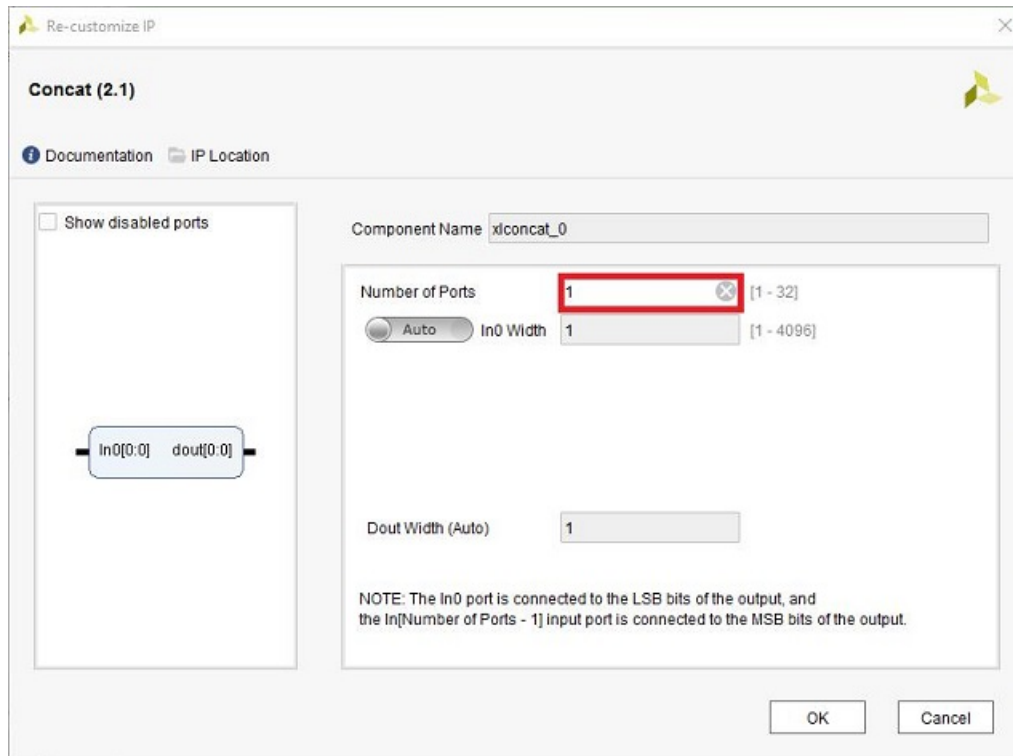
8. With the **Output Clocks** tab still selected, scroll down to the bottom of the page, and set the **Reset Type** to **Active Low**, as shown in the following figure.

Figure 15: Change Reset Type



9. Click **OK** to close the Re-Customize IP dialog box for the Clocking Wizard IP.
10. Double-click the **Concat** IP to open the Re-customize IP dialog box.
11. Change the Number of Ports field to 1, as shown in the following figure.

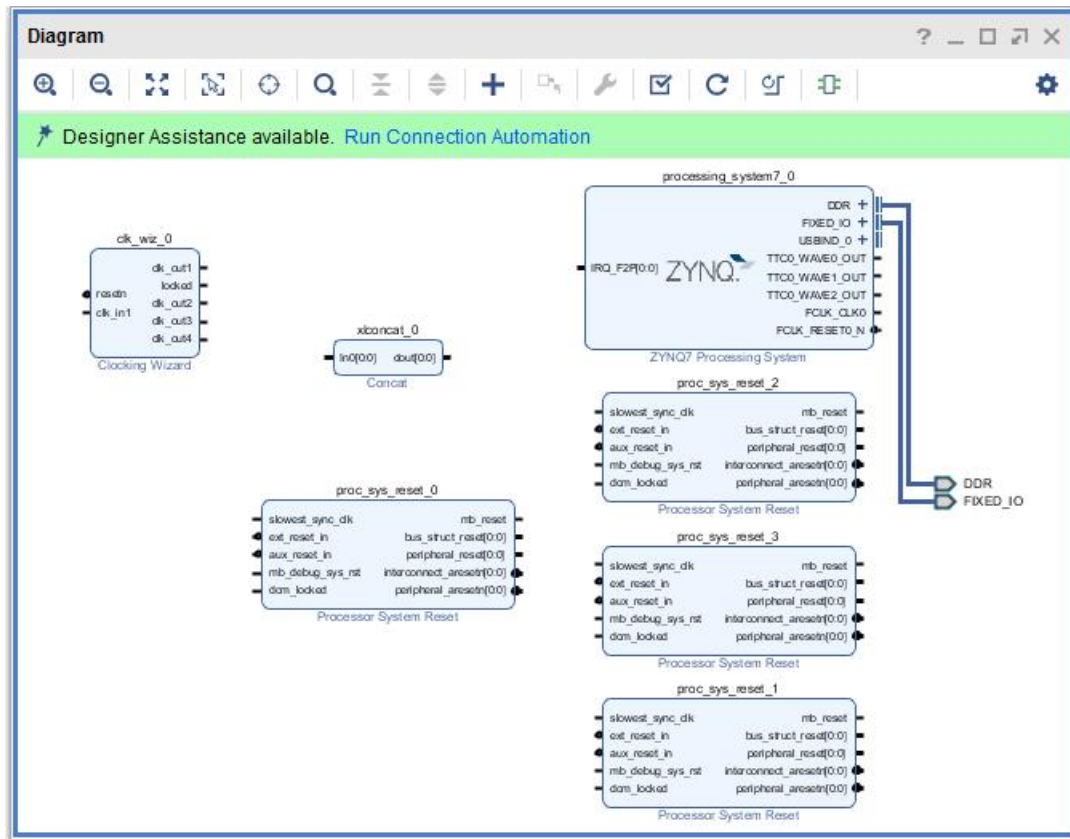
Figure 16: Re-Customize the Concat IP



12. Click **OK** to close the Re-Customize IP dialog box for the Concat IP.

The block diagram should look like the following figure.

Figure 17: Block Design After Re-Customizing IP

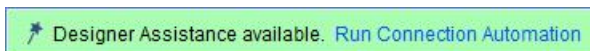


## Using Designer Assistance

Designer Assistance helps connect the Clocking Wizard and the Processor System Reset Blocks to the Zynq®-7000 AP SoC processing system.

1. Click **Run Connection Automation**, as shown in the following figure:

Figure 18: Run Connection Automation

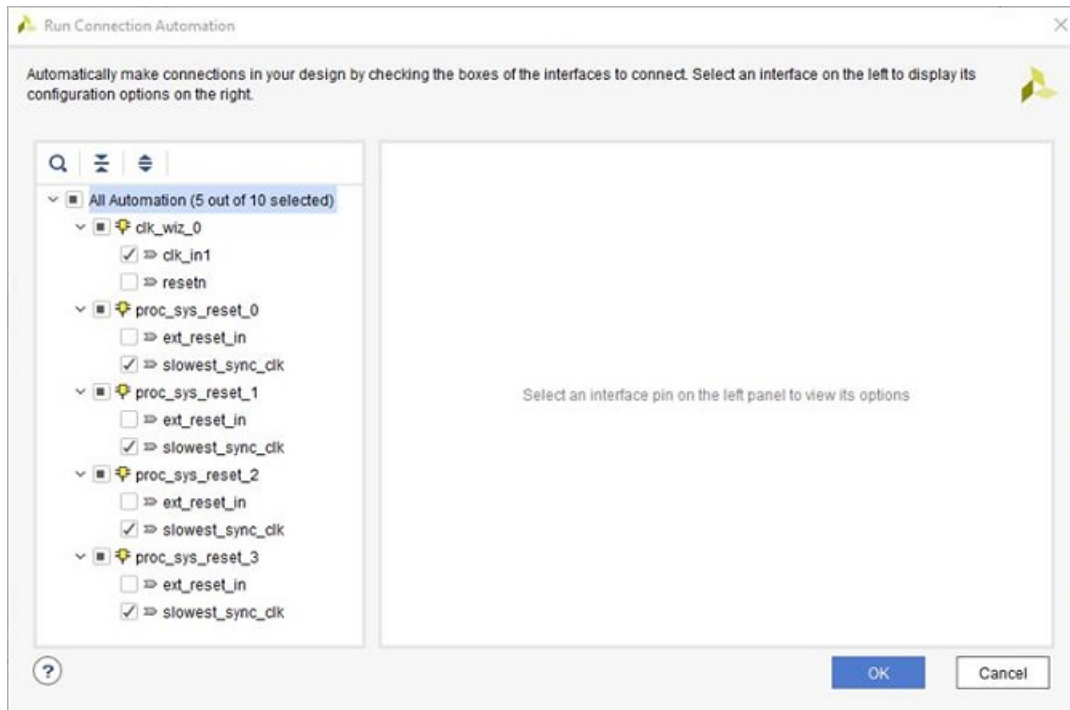


The Run Connection Automation dialog box opens.

2. Select the clock options for all the IP as shown in the following dialog box.



Figure 19: Run Connection Automation Options



As you select each interface for which connection automation is to be run, the description and options available for that interface appear in the right pane.


3. Ensure that all the automation options are set as follows:

Connection	More Information	Setting
<b>clk_wiz_0</b> <ul style="list-style-type: none"> <li>clk_in1</li> </ul>	The input clock to the clocking wizard.	<b>/processing_system7_0/FCLK_CLK0 (50 MHz)</b> is selected by default. Leave it to its default setting.
<b>proc_sys_reset_0</b> <ul style="list-style-type: none"> <li>slowest_sync_clk</li> </ul>	The clock source that the input reset for the Processor System Reset IP should be synchronized to.	From the drop-down menu for Clock Source, select <b>/clk_wiz_0/clk_out1 (100 MHz)</b> .
<b>proc_sys_reset_1</b> <ul style="list-style-type: none"> <li>slowest_sync_clk</li> </ul>	The clock source that the input reset for the Processor System Reset IP should be synchronized to.	From the drop-down menu for Clock Source, select <b>/clk_wiz_0/clk_out2 (142 MHz)</b> .
<b>proc_sys_reset_2</b> <ul style="list-style-type: none"> <li>slowest_sync_clk</li> </ul>	The clock source that the input reset for the Processor System Reset IP should be synchronized to.	From the drop down menu for Clock Source, select <b>/clk_wiz_0/clk_out3 (166 MHz)</b> .
<b>proc_sys_reset_3</b> <ul style="list-style-type: none"> <li>slowest_sync_clk</li> </ul>	The clock source that the input reset for the Processor System Reset IP should be synchronized to.	From the drop down menu for Clock Source, select <b>/clk_wiz_0/clk_out4 (200 MHz)</b> .

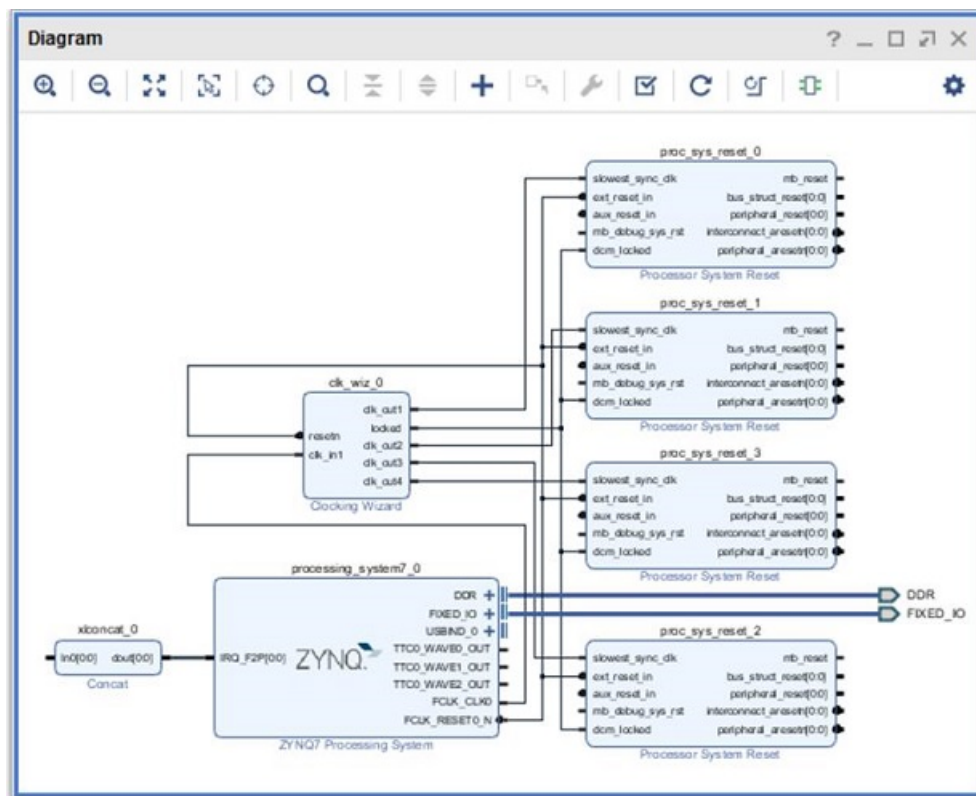
4. Click **OK**.




To make manual connections to connect the rest of the design.

1. Connect the **FCLK\_RESET0\_N** pin of the ZYNQ7 Processing System IP to the **resetn** pin of the Clocking Wizard.
2. Connect the **FCLK\_RESET0\_N** pin of the ZYNQ7 Processing System IP to the **ext\_reset\_in** pin of Processor System Reset IP (all instances `proc_sys_reset_0`, `proc_sys_reset_1`, `proc_sys_reset_2`, and `proc_sys_reset_3`).
3. Connect the **locked** pin of the Clocking Wizard IP to **dcm\_locked** pins of `proc_sys_reset_0`, `proc_sys_reset_1`, `proc_sys_reset_2`, and `proc_sys_reset_3`.
4. Connect the **dout[0:0]** output pin of the Concat block to the **IRQ\_F2P[0:0]** input pin of the ZYNQ7 Processing System.
5. Click the **Regenerate Layout** button  to regenerate an optimal layout of the block design.
6. At this point, the block diagram should look like the following figure.

**Figure 20: Regenerate an Optimal Layout of the Block Design**



7. Validate the design by clicking the **Validate** button .

8. The Validate Design dialog box pops up. Click **OK**.
9. Save the block design by clicking the  button in the toolbar or by pressing Ctrl+S.

## Step 3: Setting Platform Properties

After you complete the hardware platform design project in the Vivado Design Suite, you must add platform properties (PFM) to define the platform name and configure platform interfaces such as clocks, interrupts, and bus interfaces. These properties are set once and stored in the project.

Platforms typically consists of multiple clocks. In this custom board, our design contains four different clocks that are produced using a Clocking Wizard. While only one of the clocks out of the four is enabled in this lab, you can choose the desired clock frequency in SDx IDE for the hardware functions to be accelerated. Likewise, AXI Ports to be used also need to be identified and tagged with the Platform properties so the hardware functions can use these ports to move data to and from the hardware functions. While these AXI ports may or may not be visible in the block design used for creating the hardware design in IP integrator, tagging these ports with platform properties, makes them available for use by the hardware functions later using SDx IDE.

1. The Platform Identification property (PFM\_NAME) must be set in the hardware design to define the Vendor, Library, Name, and Version (VLNV) of the platform. In the Tcl Console, type the following and press **Enter**. This sets the hardware platform name.

```
set_property PFM_NAME "xilinx.com:zynq7_board:zynq7_board:1.0" \
[get_files [get_property FILE_NAME [get_bd_designs]]]
```

The PFM\_NAME property needs the following syntax:

```
<vendor>:<library>:<platform>:<version>
```

2. You can export any clock source with the platform, but for each clock you must also export synchronized reset signals using a Processor System Reset IP block in the platform. The PFM.CLOCK property can be set on BD cell, external port, or external interface. Accordingly, define clocks by typing the following Tcl commands in the Tcl Console and press **Enter**.

```
set_property PFM.CLOCK { \
  clk_out1 {id "2" is_default "true" proc_sys_reset
"proc_sys_reset_0" } \
  clk_out2 {id "1" is_default "false" proc_sys_reset
"proc_sys_reset_1" } \
  clk_out3 {id "0" is_default "false" proc_sys_reset
"proc_sys_reset_2" } \
  clk_out4 {id "3" is_default "false" proc_sys_reset
"proc_sys_reset_3" } \
} [get_bd_cells /clk_wiz_0]
```

- Define the AXI ports in the design by typing the following in the TCL console and press Enter.

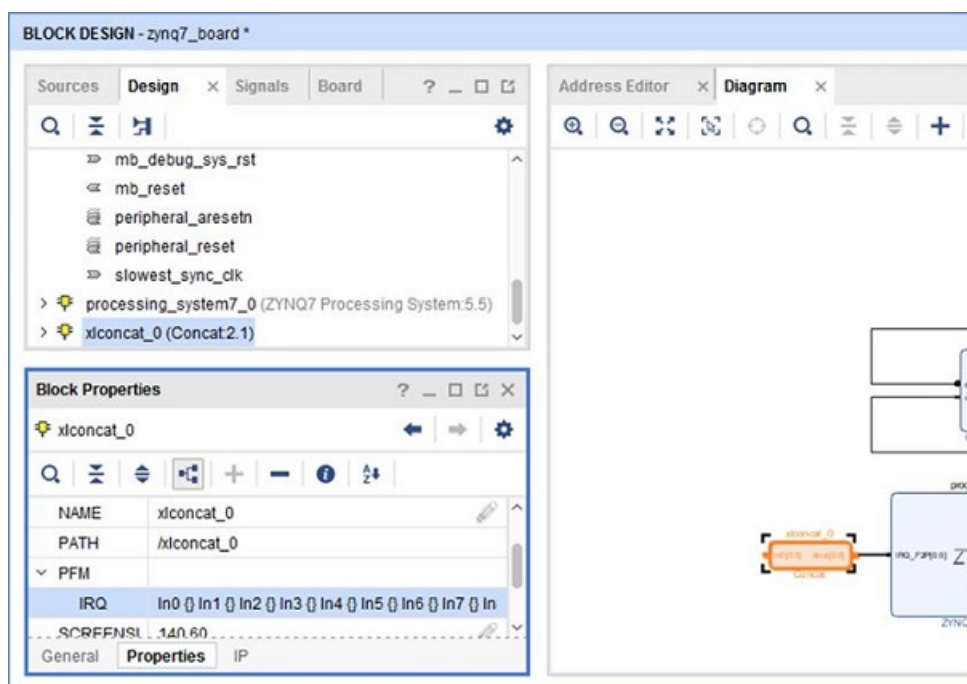
```
set_property PFM.AXI_PORT { \
    M_AXI_GP0 {memport "M_AXI_GP"} \
    M_AXI_GP1 {memport "M_AXI_GP"} \
    S_AXI_ACP {memport "S_AXI_ACP" sptag "ACP" memory
"processing_system7_0 ACP_DDR_LOWOCM"} \
    S_AXI_HP0 {memport "S_AXI_HP" sptag "HP0" memory
"processing_system7_0 HP0_DDR_LOWOCM"} \
    S_AXI_HP1 {memport "S_AXI_HP" sptag "HP1" memory
"processing_system7_0 HP1_DDR_LOWOCM"} \
    S_AXI_HP2 {memport "S_AXI_HP" sptag "HP2" memory
"processing_system7_0 HP2_DDR_LOWOCM"} \
    S_AXI_HP3 {memport "S_AXI_HP" sptag "HP3" memory
"processing_system7_0 HP3_DDR_LOWOCM"} \
} [get_bd_cells /processing_system7_0]
```

- Interrupts must be connected to IP integrator Concat (xlconcat) blocks that are connected to the processing system. For Zynq®-7000 family it's the F2P\_irq port. Type the following commands on the TCL console and press Enter.

```
set intVar []
for {set i 0} {$i < 16} {incr i} {
    lappend intVar In$i {}
}
set_property PFM.IRQ $intVar [get_bd_cells /xlconcat_0]
```

- These properties can be viewed in the Properties windows by selecting the appropriate cells in the block design canvas. As an example, when you select the Concat cell in the canvas, you will see a PFM pull down menu in the Block Properties of the Concat block. When the drop down menu is expanded you will see the IRQ properties for your platform.

Figure 21: Looking up applied PFM Properties



## Step 4: Generating HDL Design Files

You now generate the HDL files for the design.

1. To generate the HDL, right-click the **zynq7\_board** block design in the Sources window and from the context menu select **Generate Output Products**.
2. The Generate Output Product dialog box pops up. Select **Synthesis** → **Global**, and click **Generate**.

You can also generate the Output Products by typing the following on the Tcl Console.

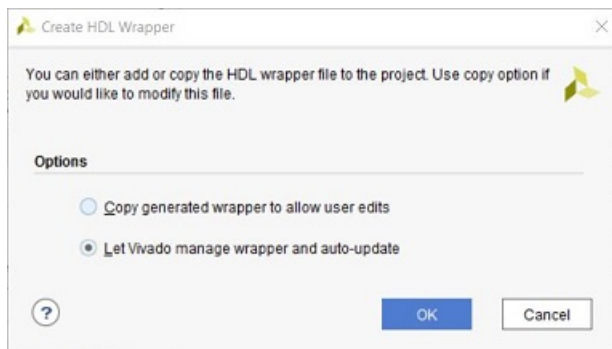
```
generate_target all [get_files <path_to_project>/zynq7_board/
zynq7_board.srsc/sources_1/bd/zynq7_board/zynq7_board.bd]
```

3. After output products have been generated, the Generate Output Products dialog box opens. Click **OK** to dismiss the dialog box.
4. Right-click the **zynq7\_board** block design in the Sources window, and select **Create HDL Wrapper** from the context menu.

The Create HDL Wrapper dialog box opens.

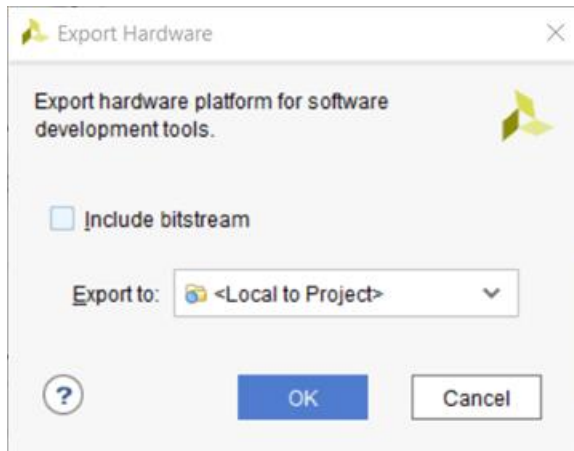
5. The Let Vivado manage wrapper and auto-update is selected by default. Click **OK**.

Figure 22: Create HDL Wrapper dialog box



6. Finally, from the menu select **File** → **Export** → **Export Hardware**.
7. The Export Hardware dialog box opens, as shown in the following figure. Click **OK**.

Figure 23: Export Hardware Dialog Box



**Note:** A bitstream for the design has not been generated yet. The bitstream will be generated when the hardware functions are included as a part of the overall design. Hardware functions are not a part of the Hardware Platform therefore, the design is incomplete at this point.

## Step 5: Writing Out the DSA

Finally, you are ready to write out the Device Support Archive (DSA).

1. To write the DSA, type the following in the Tcl Console and press **Enter**.

```
write_dsa -force <path_to_project>/zynq7_board.dsa
```

2. Validate the DSA by typing the following on the Tcl Console.

```
validate_dsa <path_to_project>/zynq7_board.dsa
```

## Conclusion

This lab introduced you to creating a Zynq-based design in the IP integrator, specifying properties on the block design to encapsulate platform related data, and finally, generating the DSA.

---

## Lab Files

You can use the Tcl file `lab1.tcl` that is included with this tutorial design files to perform all the steps in this lab. To use the Tcl script, launch Vivado and type `source lab1.tcl` in the Tcl console.

Alternatively, you can also run the script in the batch mode by typing `Vivado -mode batch -source lab1.tcl` at the command prompt.

## Lab 2: Creating Software Components for the Platform

In this lab you create software components that are required to create a platform. This process can be very involved depending on the type of configuration that the platform is being targeted. For example, to create a Linux-based platform you need to use the PetaLinux environment to create the required software files. In this lab you create files that are needed for targeting our platform to a Standalone or bare-metal system. There are three files needed for targeting a platform to a Standalone system – First Stage Boot Loader (FSBL), the linker script and the Boot Image Format Files (.bif) file.

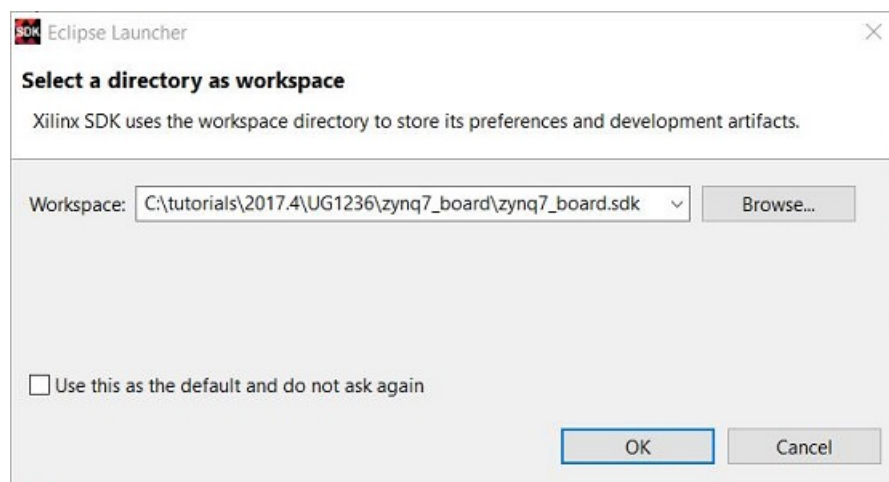
### Step 1: Launching SDK

If you are continuing on from Lab 1, you can launch SDK from Vivado, by selecting **File** → **Launch SDK**. This opens the Launch SDK dialog box.

1. Click **OK** with default options.

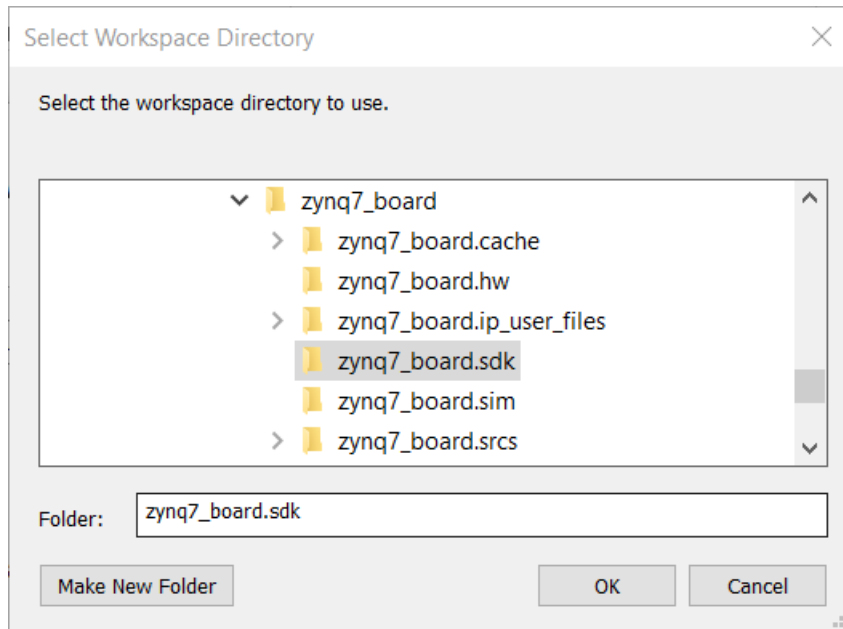
You can also launch SDK from the Start Menu by selecting **Xilinx SDK 2017.x**. The Eclipse Launcher dialog box opens, as shown in the following figure.

*Figure 24: Point to the Appropriate Workspace Directory*



2. Click the **Browse** button of the Eclipse launcher dialog box.
3. In the Select Workspace Directory dialog box, navigate to the appropriate workspace for SDK to start in and click **OK**.

Figure 25: Select Workspace Directory

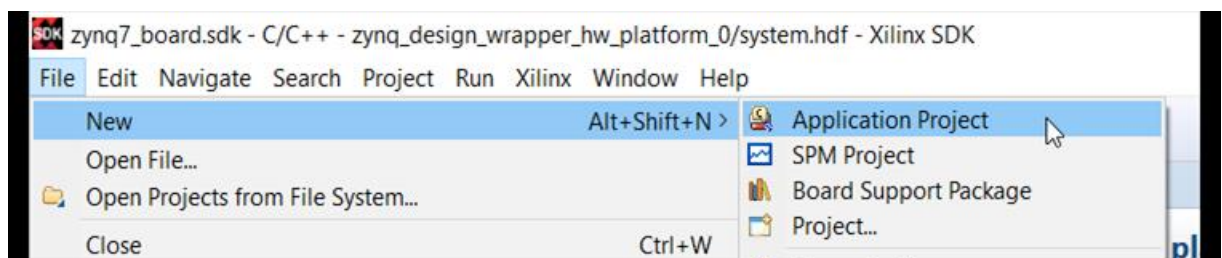


## Step 2: Creating an FSBL Application

After SDK launches, it imports the hardware specification that was previously exported from Vivado. At this point, you can create a software application.

1. From the menu, select **File** → **New** → **Application Project**, as shown in the following figure:

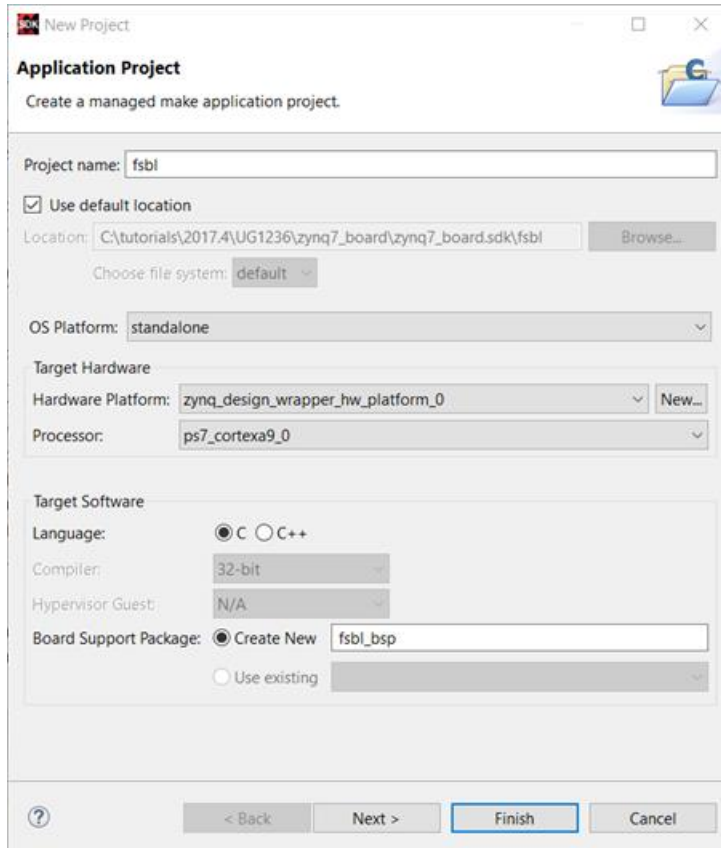
Figure 26: Create a New Application Project



2. In the New Project Wizard Application Project page, type in a name of the application project such as fsbl. Note that the OS Platform selected by default is standalone.

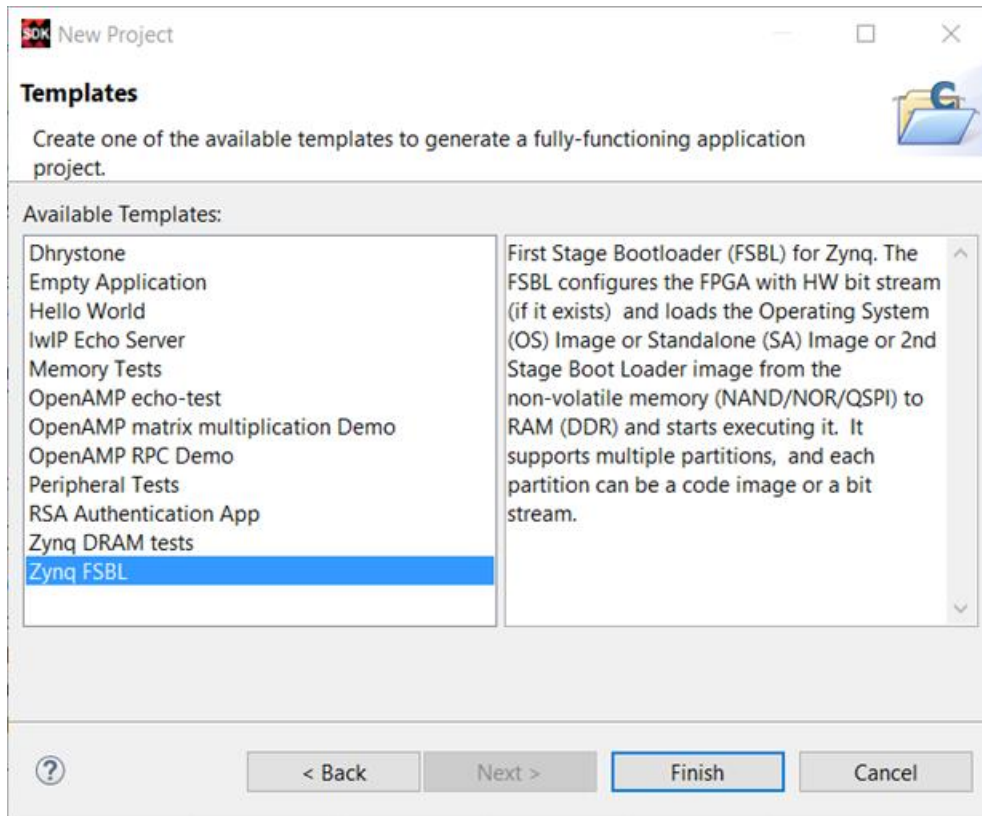


Figure 27: New Project Wizard: Application Project Page



3. Click **Next**.
4. In the Templates page, select **Zynq FSBL**.

Figure 28: Select Zynq FSBL Application



5. Click **Finish**.
6. Ensure that the application has finished compiling by checking the Console.

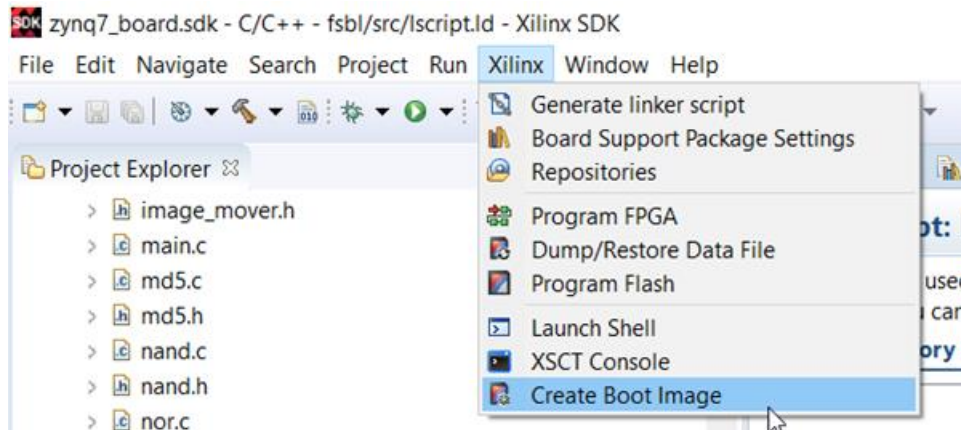
Figure 29: Ensure that the Application has Finished Compiling



## Step 3: Generating the Boot Image File (BIF)

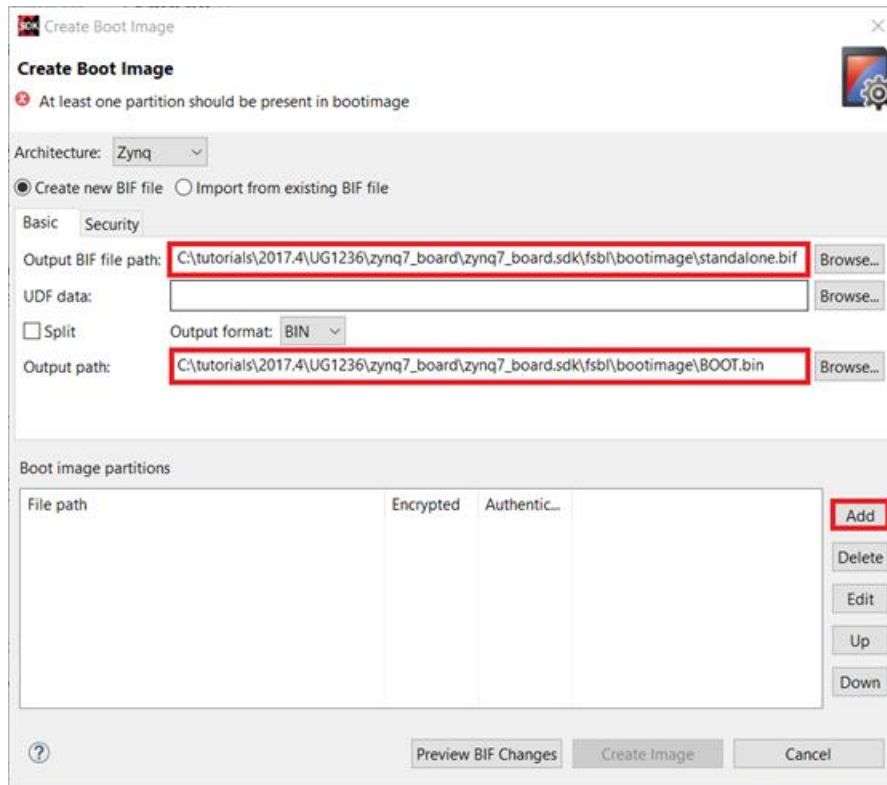
1. From the menu, select **Xilinx** → **Create Boot Image**, as shown in the following figure:

Figure 30: Create Boot Image



2. In the Create Boot Image dialog box, specify the location where the BIF file is to be created and specify a name for it as shown in the following figure.

Figure 31: Add Boot Image Partitions

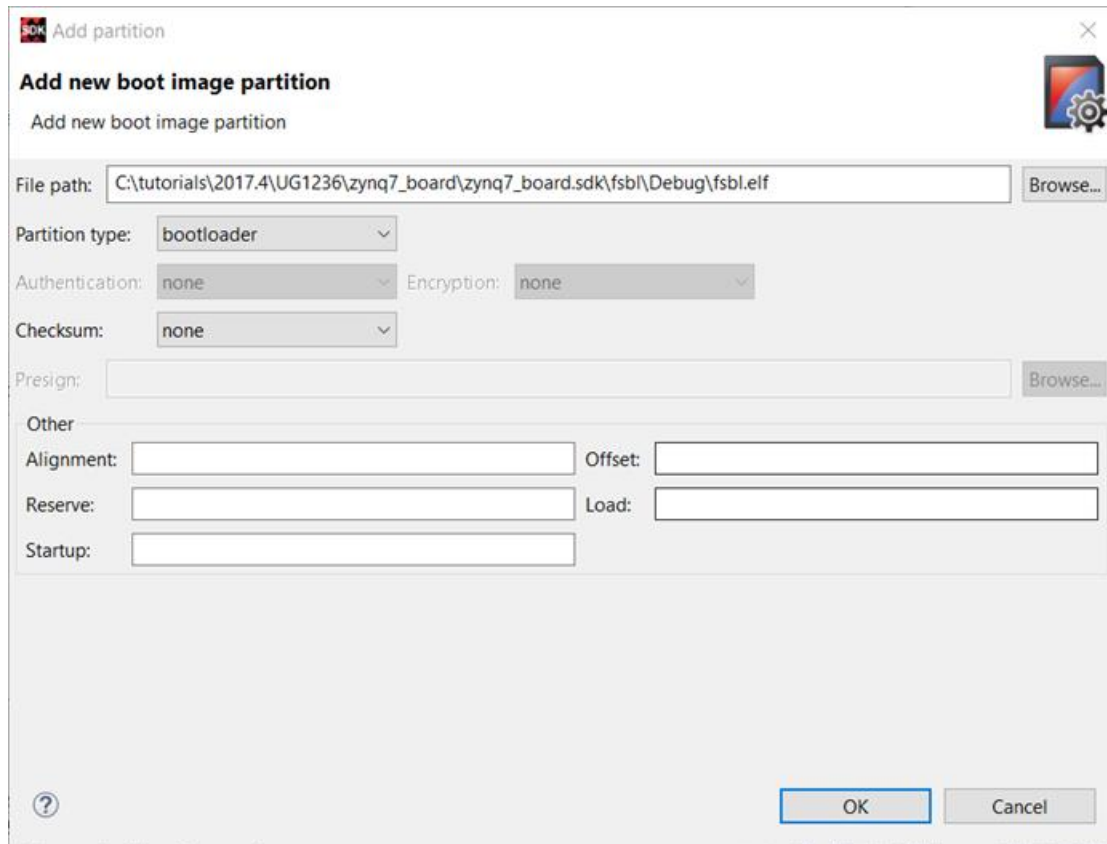


3. In the Create Boot Image dialog box, click the **Add** button in the Boot Image Partitions section.

The Add Partitions dialog box opens.

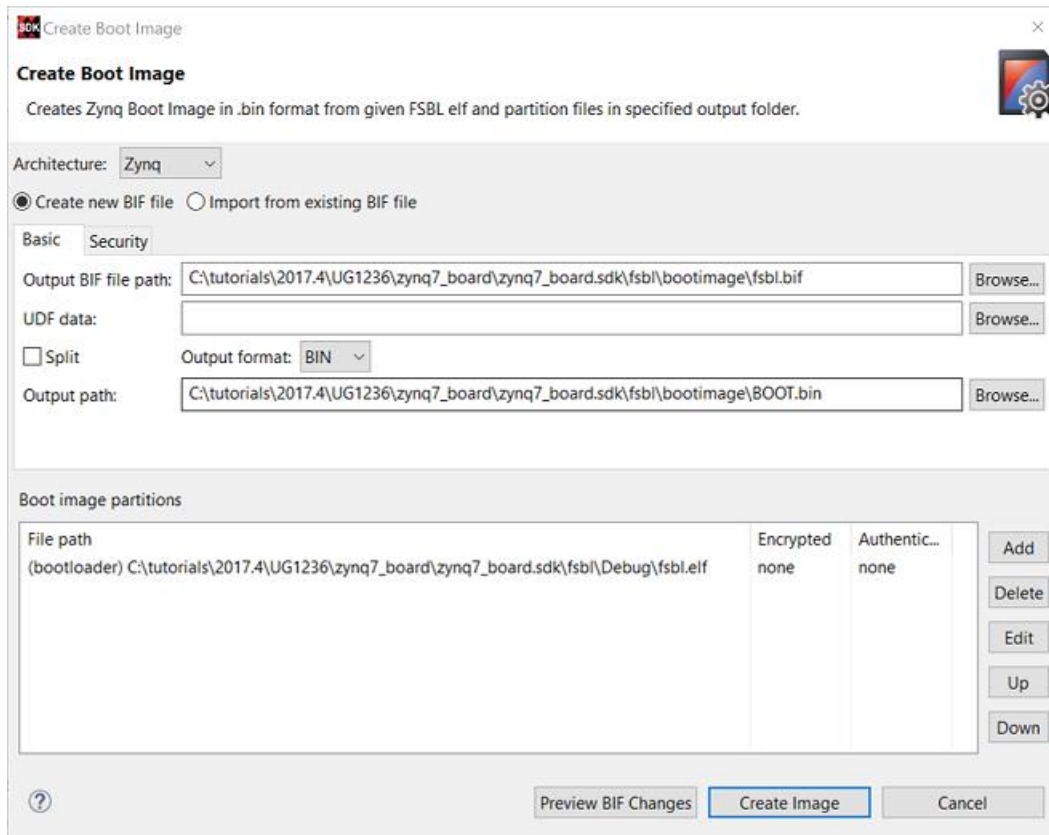
4. Navigate to the `fsbl.elf` file created earlier by clicking the **Browse** button.

Figure 32: Point to the fsbl.elf File



5. Click **OK**.
6. Click the **Create Image** button to create the BIF file, as shown in the following figure.

Figure 33: Create Boot Image

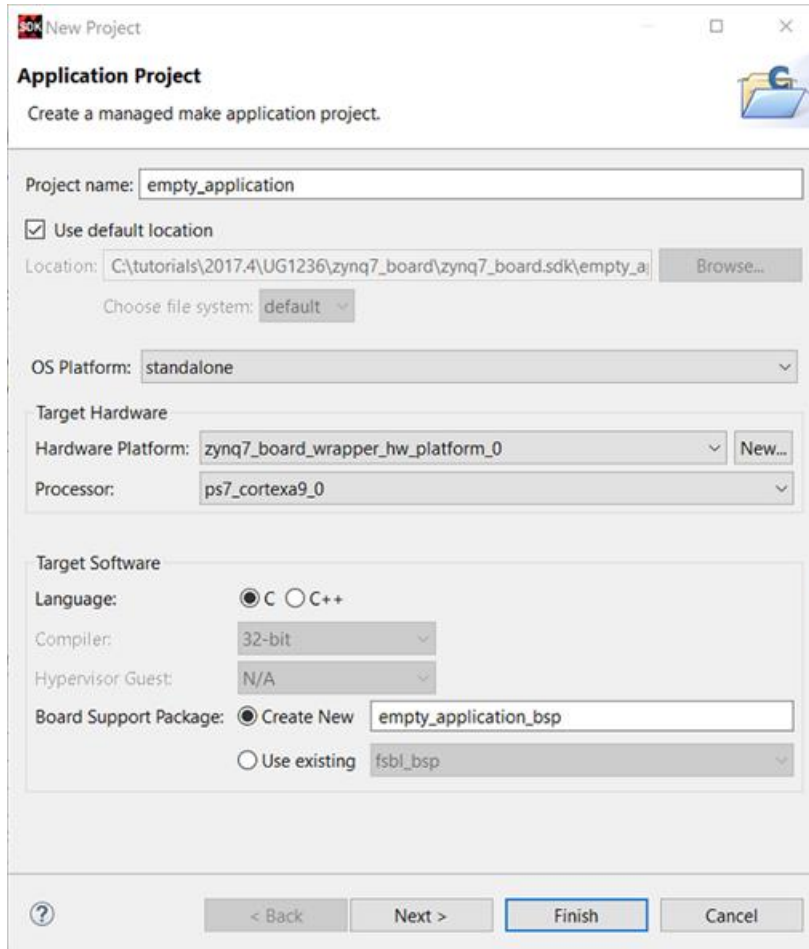


## Step 4: Creating a New Software Application and a Linker Script

You need a new linker script to create the Platform in SDx. This linker script ensures that any application code targeted to the custom platform, resides and runs from the DDR memory on the board. To create a new linker script, you should create an empty application.

1. From the SDK menu, select **File** → **New** → **Application Project**.
2. Specify a name for the project such as `empty_application`.

Figure 34: Specify name of the application project



**New Project**

**Application Project**  
Create a managed make application project.

Project name:

☒ Use default location

Location:

Choose file system:

OS Platform:

Target Hardware

Hardware Platform:

Processor:

Target Software

Language: ☒ C ☐ C++

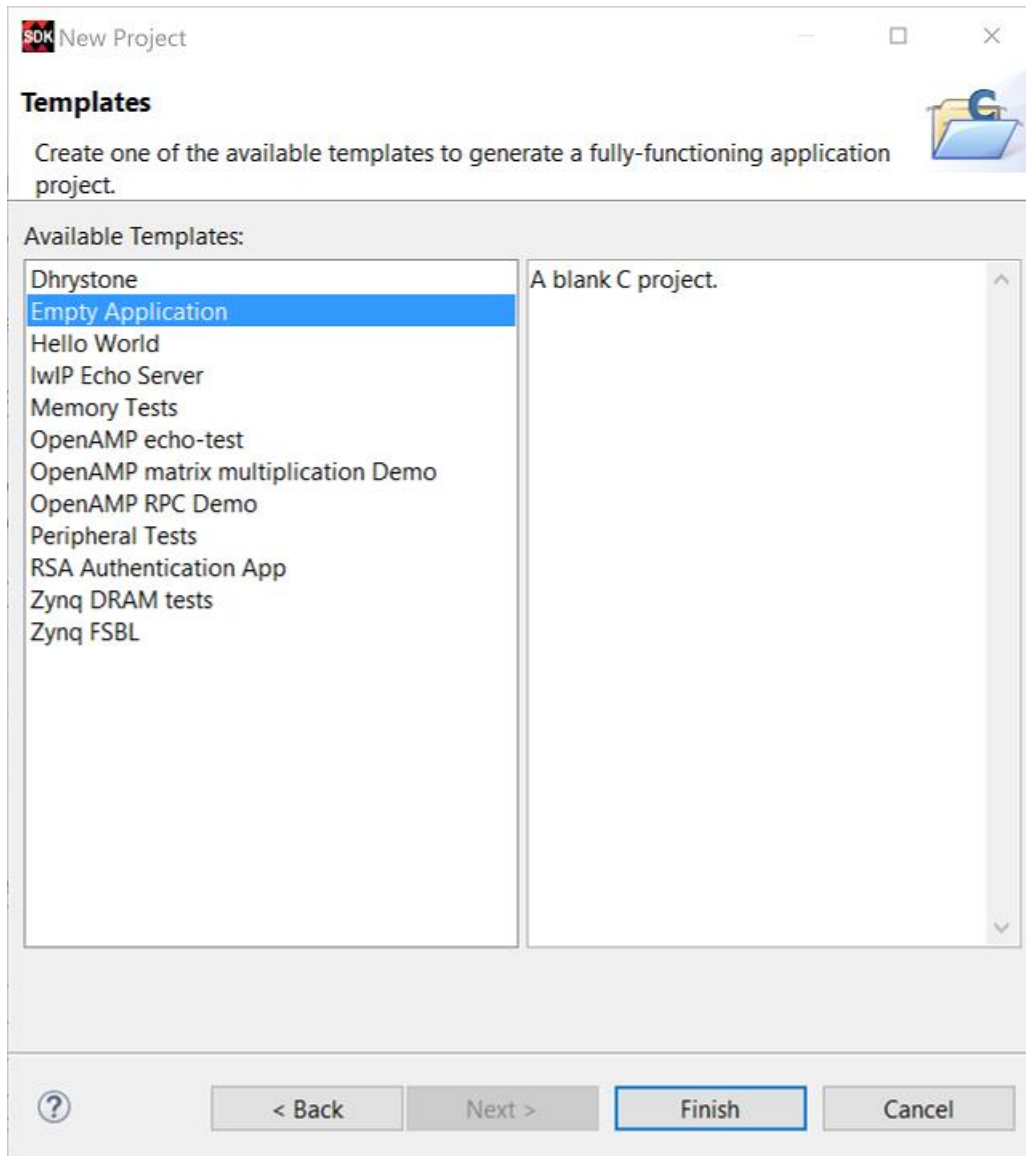
Compiler:

Hypervisor Guest:

Board Support Package: ☒ Create New  ☐ Use existing

3. Click **Next**.
4. In the Templates page select **Empty Application** from the Available Templates section and click **Finish**, as shown in the following figure.

Figure 35: Select Empty Application template

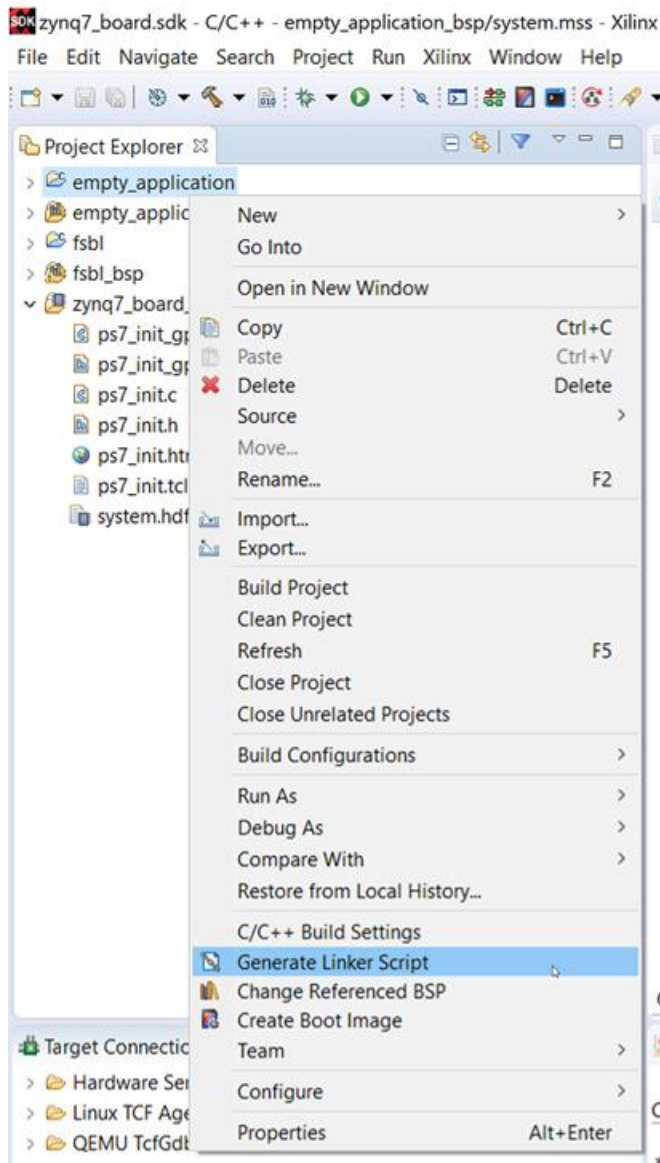


Wait for the code to finish compiling. Next, you must modify the linker script to change the stack size and the heap size.

5. Right-click **empty\_application** in the Project Explorer and from the context menu, select, **Generate Linker Script**, as shown in the following figure:

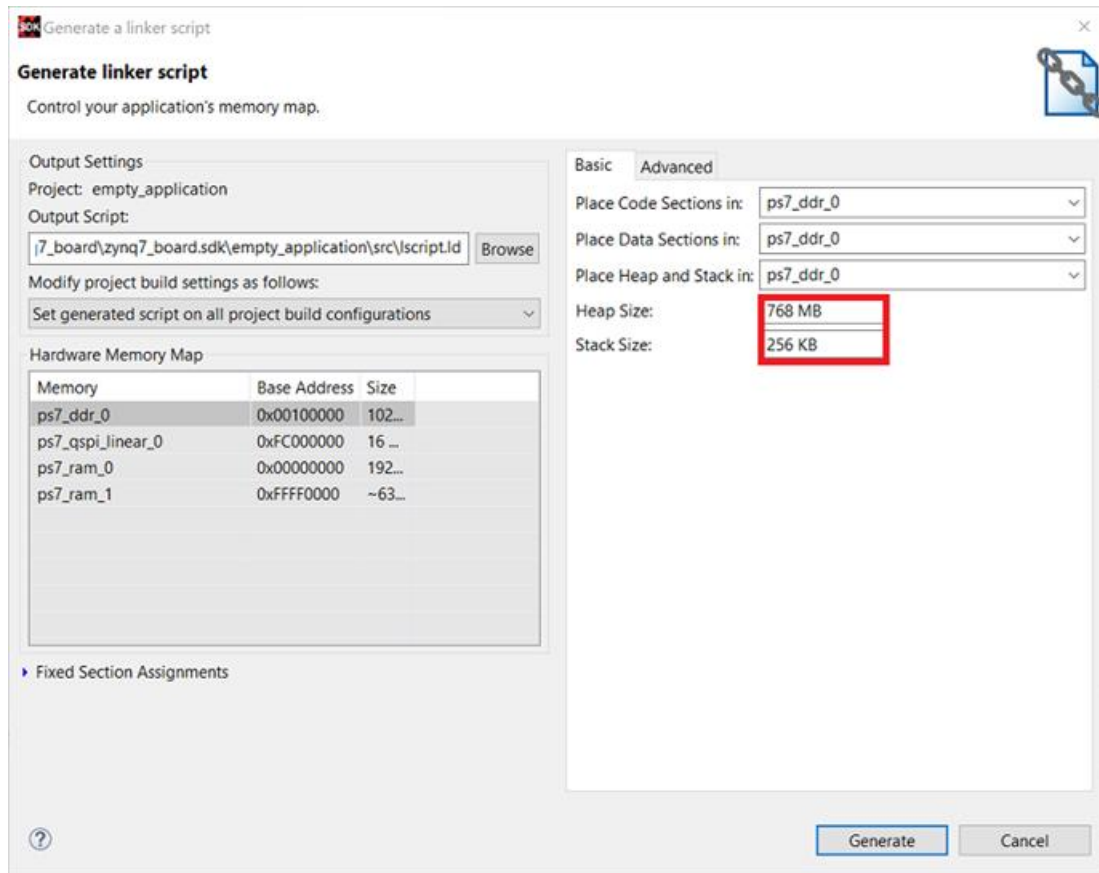


Figure 36: Generate Linker Script



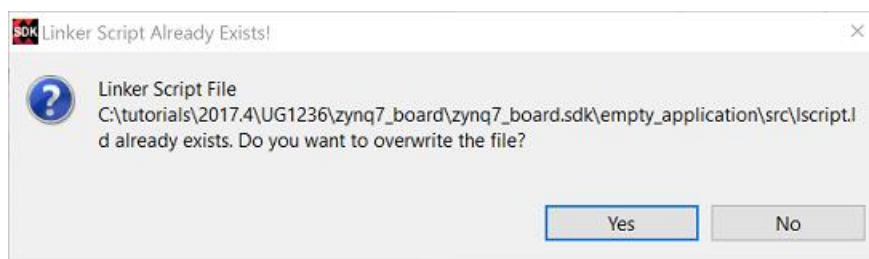
6. In the Generate linker script dialog box, change the Heap Size and the Stack Size fields to 805306368 (768 MB) and 262144 (256 KB) respectively, as shown in the following figure.

Figure 37: Change Heap Size and Stack Size



7. With all other values set to default, click **Generate**.
8. The Linker Script Already Exists! dialog box pops up. Click **Yes** to overwrite the file.

Figure 38: Overwrite Existing Linker Script

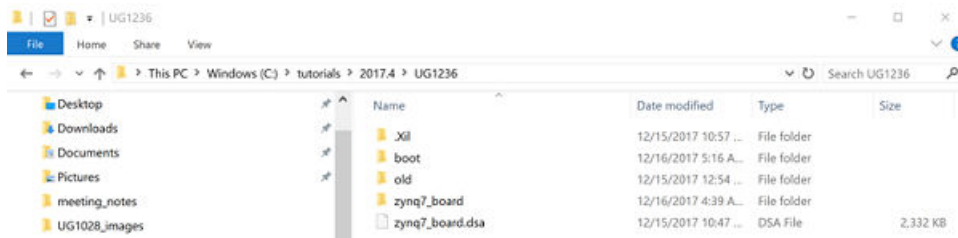


9. Wait for the application to compile with the new linker settings.
10. Exit SDK.

## Step 5: Copying and Editing Files

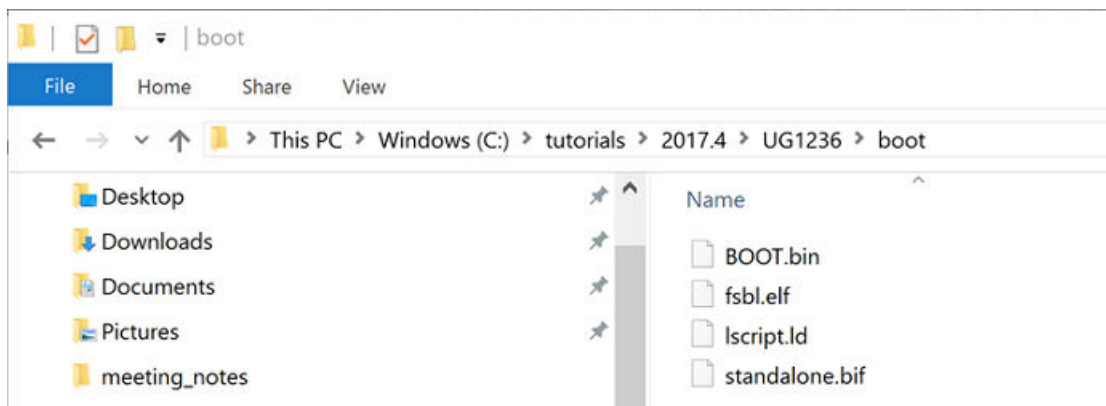
1. Now, copy the `fsbl.elf` file, the `bif` file and the linker script to a separate directory. These files will be needed for the next lab.
2. Create a directory called `boot` at the same directory level as the `zynq7_board` project.

Figure 39: Directory Structure



3. Copy the `fsbl.elf` file from `<path_to_project>\zynq7_board\zynq7_board.sdk\fsbl\Debug` to the `boot` directory created in the previous step.
4. Copy the `BOOT.BIN` and the `standalone.bif` files from `<path_to_project>\zynq7_board\zynq7_board.sdk\fsbl\bootimage` to the `boot` directory created earlier.
5. Copy the linker script for the empty application created in step 4, from `<path_to_project>\zynq7_board\zynq7_board.sdk\empty_application\src` to the `boot` directory. The `boot` directory should look as follows.

Figure 40: Files in the boot directory



6. Open the standalone.bif file from the boot directory into an editor and edit the file to look as follows:

```
/* standalone */
the_ROM_image:
{
    [bootloader]<fsbl.elf>
    <bitstream>
    <elf>
}
```

7. Save and close the file.

---

## Conclusion

In this lab, you learned how to create the software pieces required by the platform in SDK for a standalone system.

---

## Lab Files

Lab files are not required.

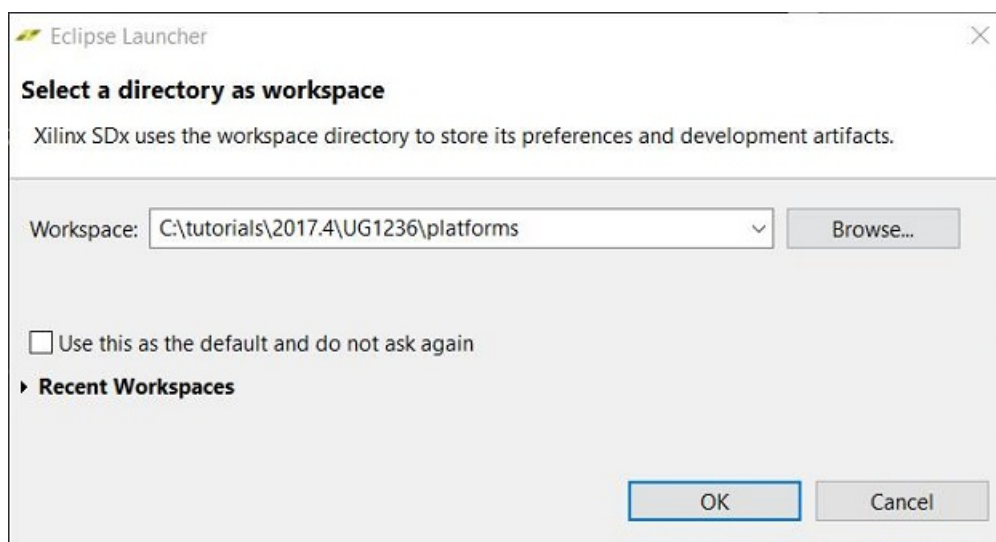
## Lab 3: Creating a Custom Platform Using the SDx IDE

In this tutorial, you use the SDx IDE to create a custom platform that you have created the DSA for in Lab 1 and the software components for the standalone configuration in Lab 2. After the custom platform is created, you add this platform to the available platforms in SDx and target software applications to the platform.

### Step 1: Invoking the SDx IDE and Creating a Project

1. Open the SDx IDE by clicking the desktop icon or from the Start menu select **Xilinx Design Tools** → **SDx IDE 2017.x**.
2. When you launch the SDx IDE, the Workspace Launcher dialog box opens. Click **Browse** to enter a workspace folder used to store your projects (you can use workspace folders to organize your work), then click **OK** to dismiss the Workspace Launcher dialog box.

Figure 41: Select a directory as workspace



3. In the Welcome tab of the SDx IDE, click **Create SDx Project**, as shown in the following figure.

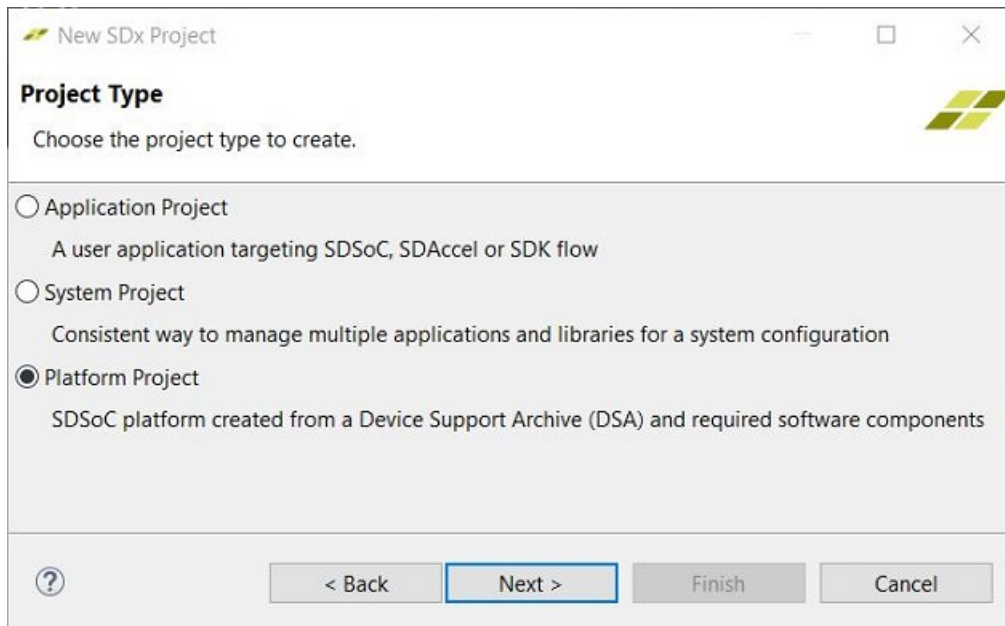
Figure 42: Create a New SDx Project



The New SDx Project wizard opens, and displays the Project Type page.

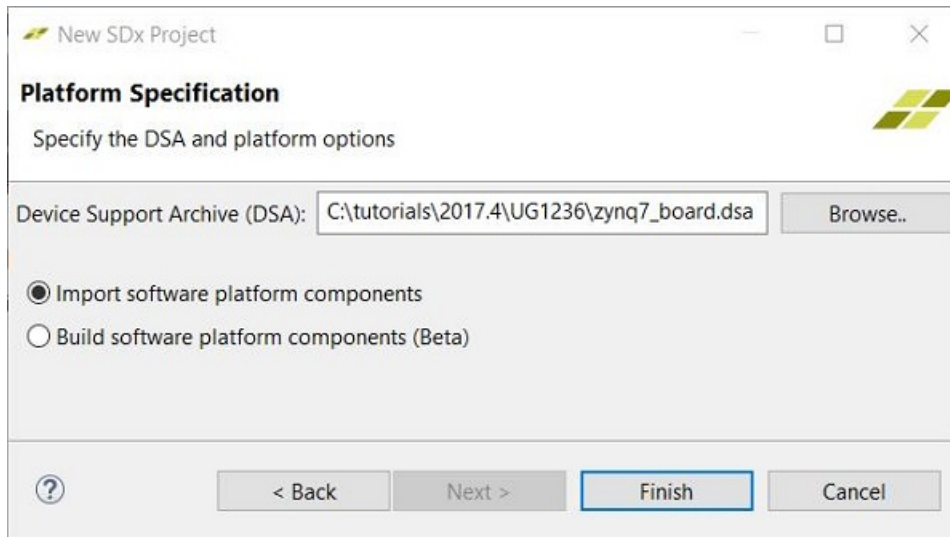
4. Select **Platform Project**, and click **Next**.

Figure 43: Select Platform Project



5. In the Platform Specification page, point to the DSA created in Lab 1 by clicking the **Browse** button, and click **Finish**.

Figure 44: New SDx Project: Platform Specification

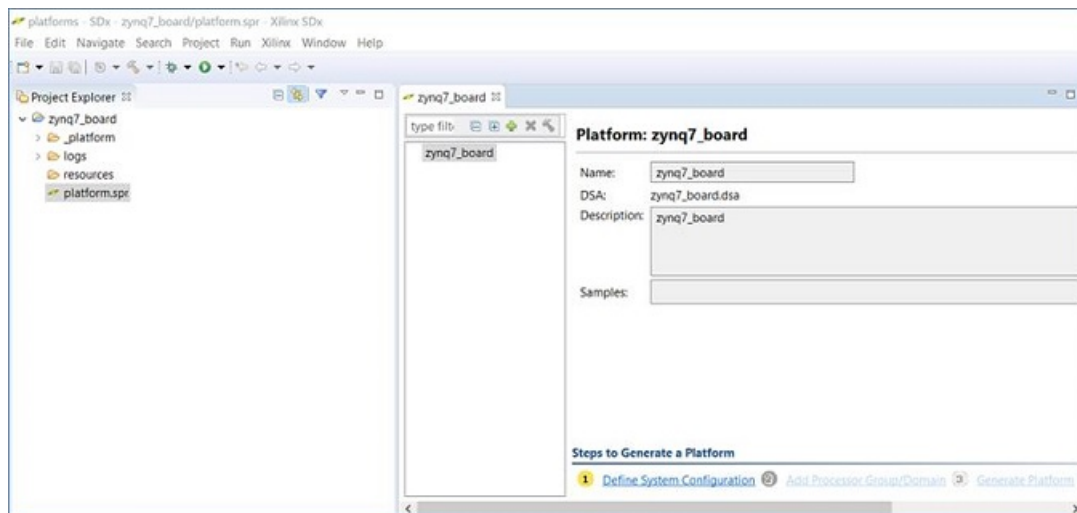


The platform project is created with the name automatically generated from the DSA file you specified in the prior step.

## Step 2: Defining System Configuration

The zynq7\_board project overview window opens, where you specify software components that were generated using the SDK tool. For more details, see the following figure.

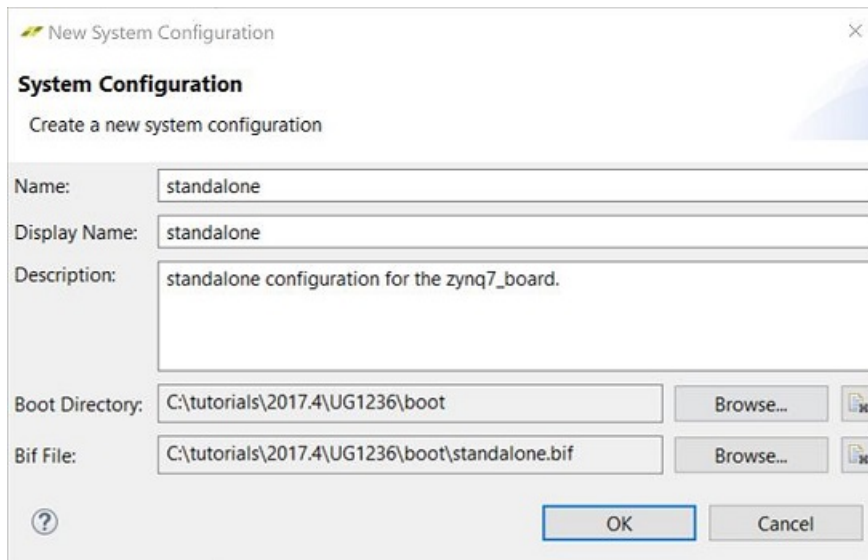
Figure 45: Project Overview Window



1. In the Overview window, click **Define System Configuration**.

2. Specify a name for this configuration such as **standalone**. The display name auto-populates to reflect the specified name; however, it can be changed as you want. Display Name is the name, which is shown while creating a new project. In this case use the default.
3. In the Description field, type a description for the configuration. This description is shown in the New System Project or New Application Project Wizard. Description can be edited by clicking on the Pen icon at the right most side of the Description field.
4. This is the directory which should have all the components referred to in the Boot Image File (BIF). If any of the components referred to by the BIF is not present in the Boot directory, the platform generation flags errors. For the Boot Directory, navigate to the folder where you created the boot image in SDK by clicking the **Browse** button.
5. Similarly, specify the BIF by clicking the **Browse** button for the Bif File field. At this point the New System Configuration should look like the following figure:

Figure 46: **New System Configuration Dialog Box**



6. After all the required fields have been filled, click **OK**.

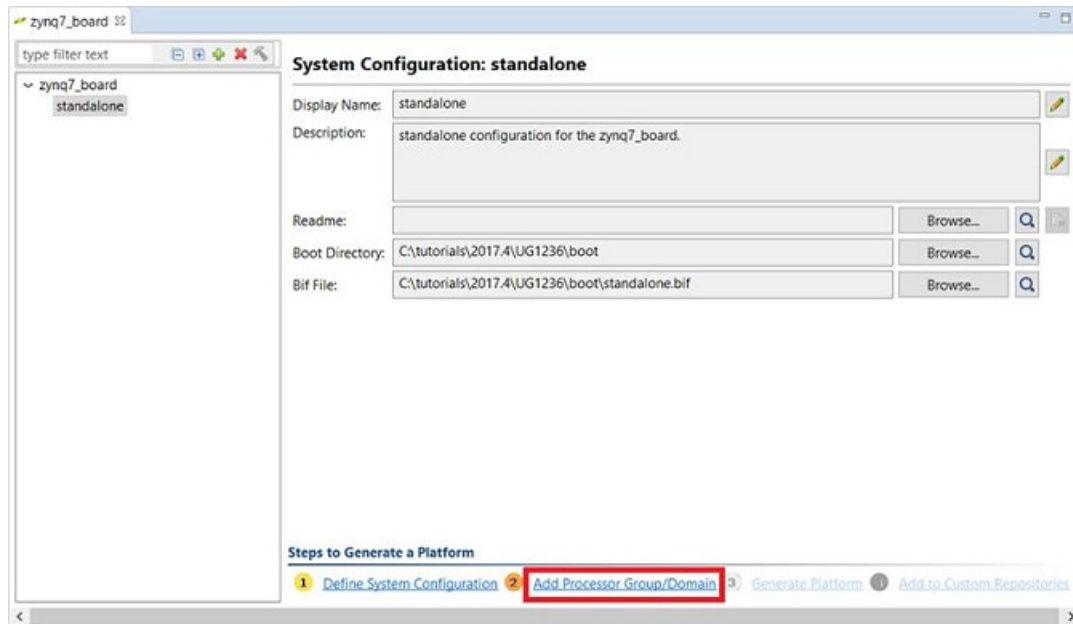
## Step 3: Adding Processor Domain

In the zynq7\_board project overview window you add a processor domain and specify software components that were generated using the SDK tool; see the following figure. A domain contains a processor or processors (in case of Linux), and the associated Operating System. It has settings which vary from standalone to Linux.

1. At this point the Add Processor Group/Domain link becomes active in the Project Overview window.
2. Click the Add Processor Group/Domain link.



Figure 47: Add Processor Group/Domain



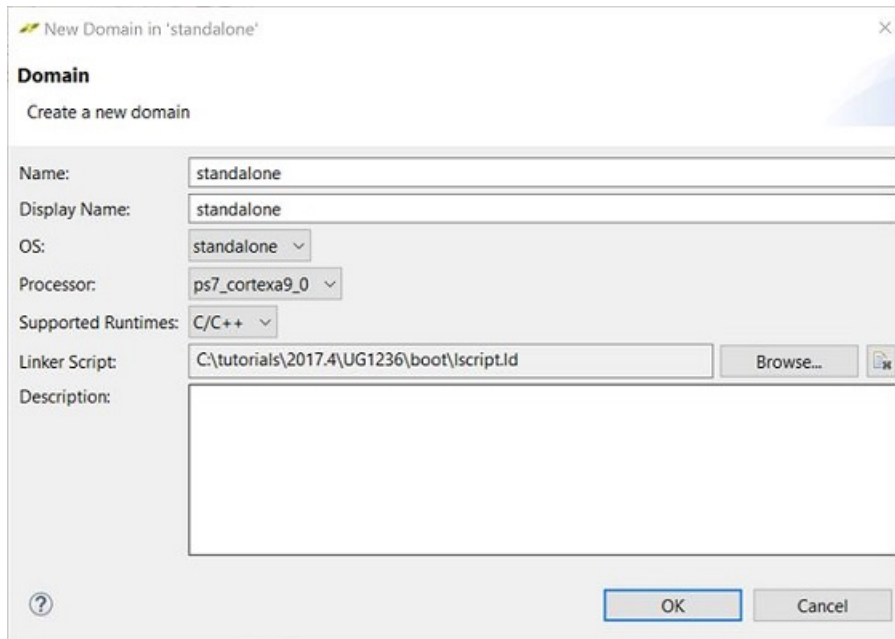
The New Domain in the standalone dialog box opens.

3. Specify the **Name** such as standalone.

Again, the Display Name auto-populates with the same value as specified in the Name field. This is the name shown in the SDx New Project wizard..

4. For **OS**, click the drop-down menu and select **standalone**.
5. For **Processor**, you can select one of the ARM processors that will be used by this configuration. Use the default, select **ps7\_cortexa9\_0**.
6. **Supported Runtimes** is by default set to **C/C++**. For **Linker Script**, navigate to the folder where the linker script was created in SDK and click **Browse** to navigate to the linker script. At this point the New Domain in standalone dialog box looks as follows:

Figure 48: New Domain Dialog Box



The dialog box is titled "New Domain in 'standalone'". It contains the following fields and controls:

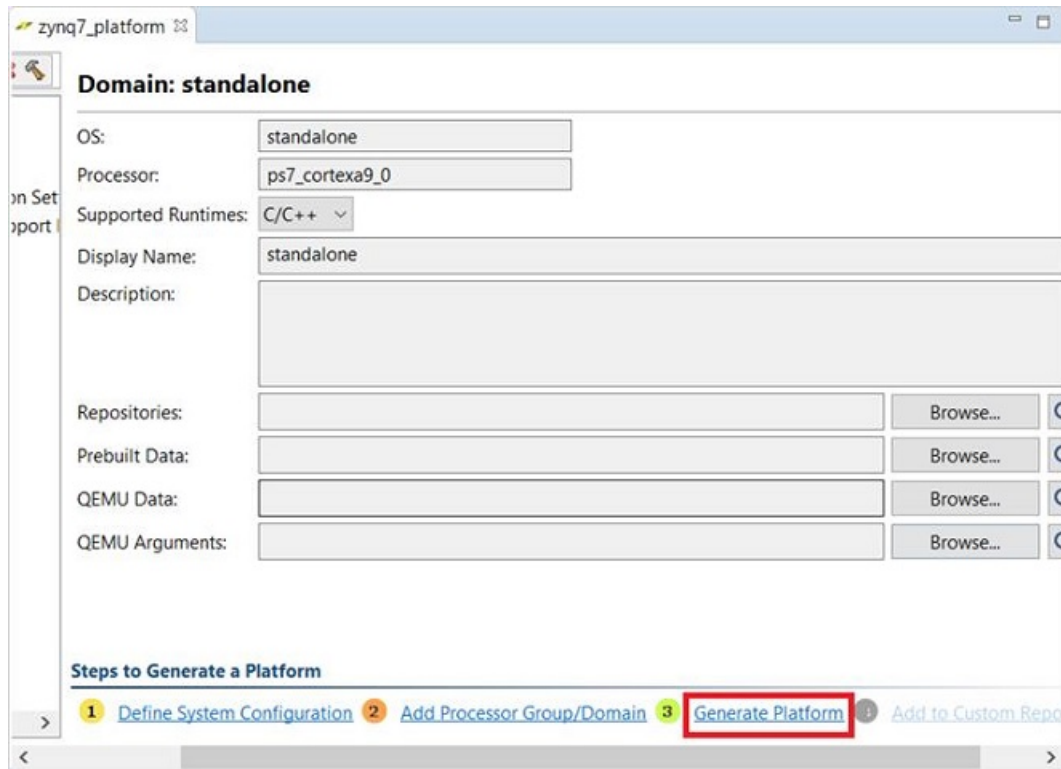
- Domain**: Create a new domain
- Name**: standalone
- Display Name**: standalone
- OS**: standalone (dropdown)
- Processor**: ps7\_cortexa9\_0 (dropdown)
- Supported Runtimes**: C/C++ (dropdown)
- Linker Script**: C:\tutorials\2017.4\UG1236\boot\lscript.ld (with a "Browse..." button)
- Description**: (empty text area)
- Buttons**: ? (help), OK, and Cancel.

7. Click OK.

## Step 4: Generating the Platform

In the zynq7\_board project overview window you generate the custom platform.

1. At this point the Generate Platform link becomes active in the Project Overview window; see the following figure:

Figure 49: **Generate Platform Link in Project Overview Window**


2. Click the **Generate Platform** link.

The Generation completed dialog box opens.

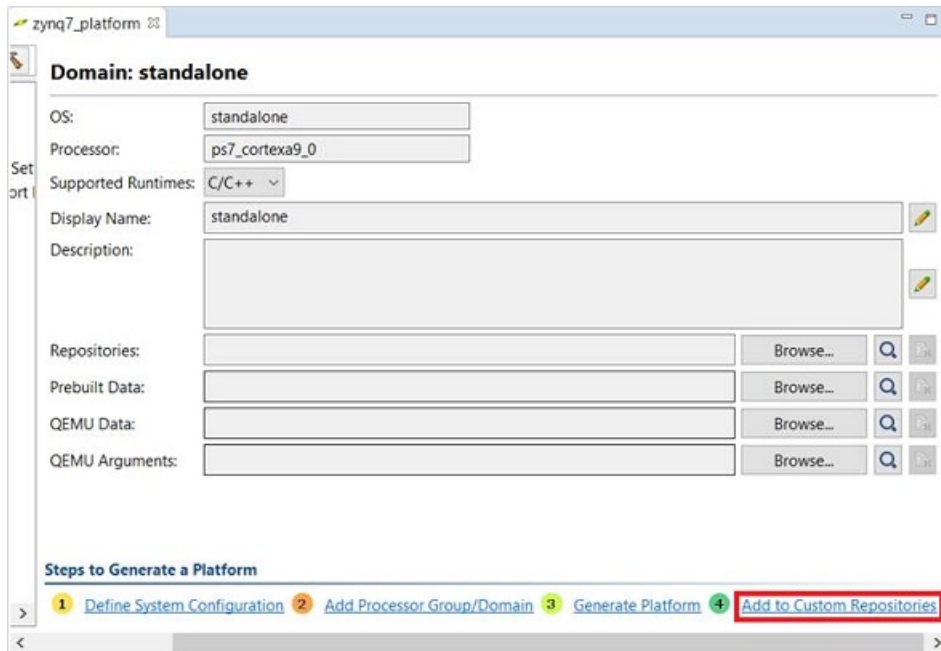
3. Click **OK** to dismiss the dialog box.

## Step 5: Adding Custom Platform to Repository

In the zynq7\_board project overview window, you add the custom platform to a platform repository; see the following figure.

1. At this point, the Add to Custom Repositories link becomes active in the Project Overview window.

Figure 50: Project Overview Window with Add to Custom Repositories Link



2. Click the **Add to Custom Repositories** link. This will add the custom repository to the current workspace and an application can be targeted to this platform.

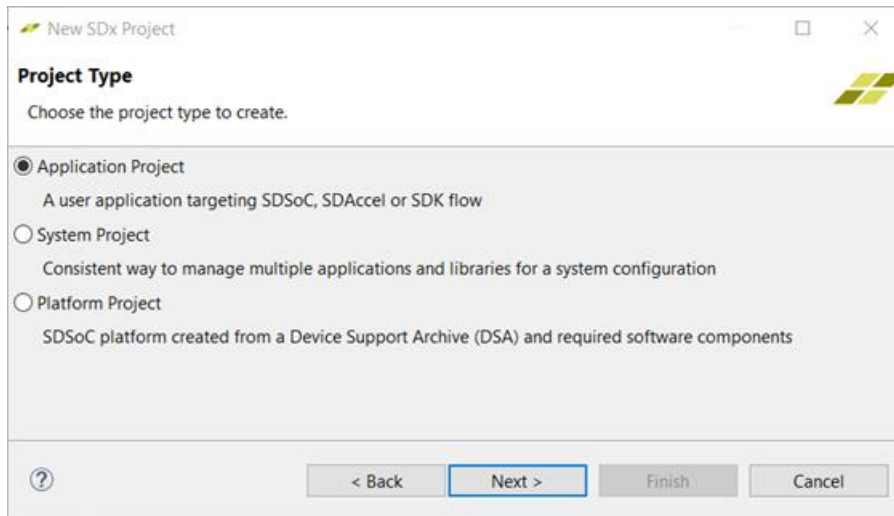
The Operation completed dialog box opens.

3. Click **OK** to dismiss the dialog box.

## Step 6: Creating an SDx Application Targeting the Custom Platform

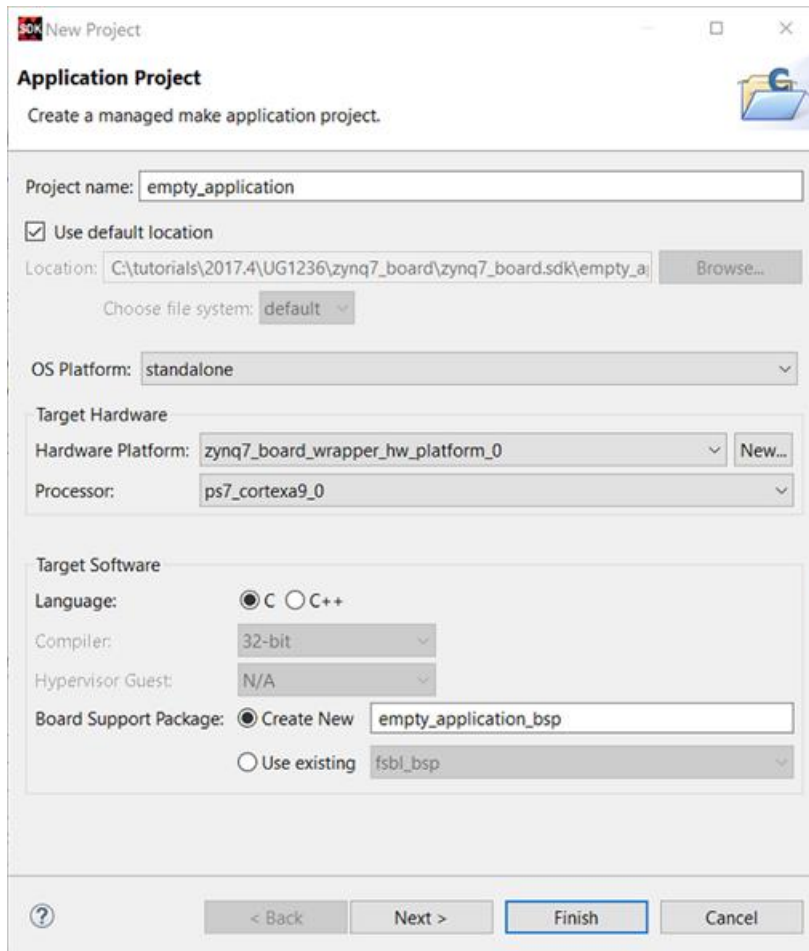
1. Create a new application by selecting **File** → **New** → **SDx Project** from the menu.
2. In the Project Type page, Application Project is selected by default. Click **Next**.

Figure 51: Select the Type of the Project



3. In the New SDx Project dialog box, type a name for the project such as empty\_application.

Figure 52: Specify Name of the SDx Application Project



**New Project**

**Application Project**  
Create a managed make application project.

Project name:

☒ Use default location

Location:

Choose file system:

OS Platform:

Target Hardware

Hardware Platform:

Processor:

Target Software

Language: ☒ C ☐ C++

Compiler:

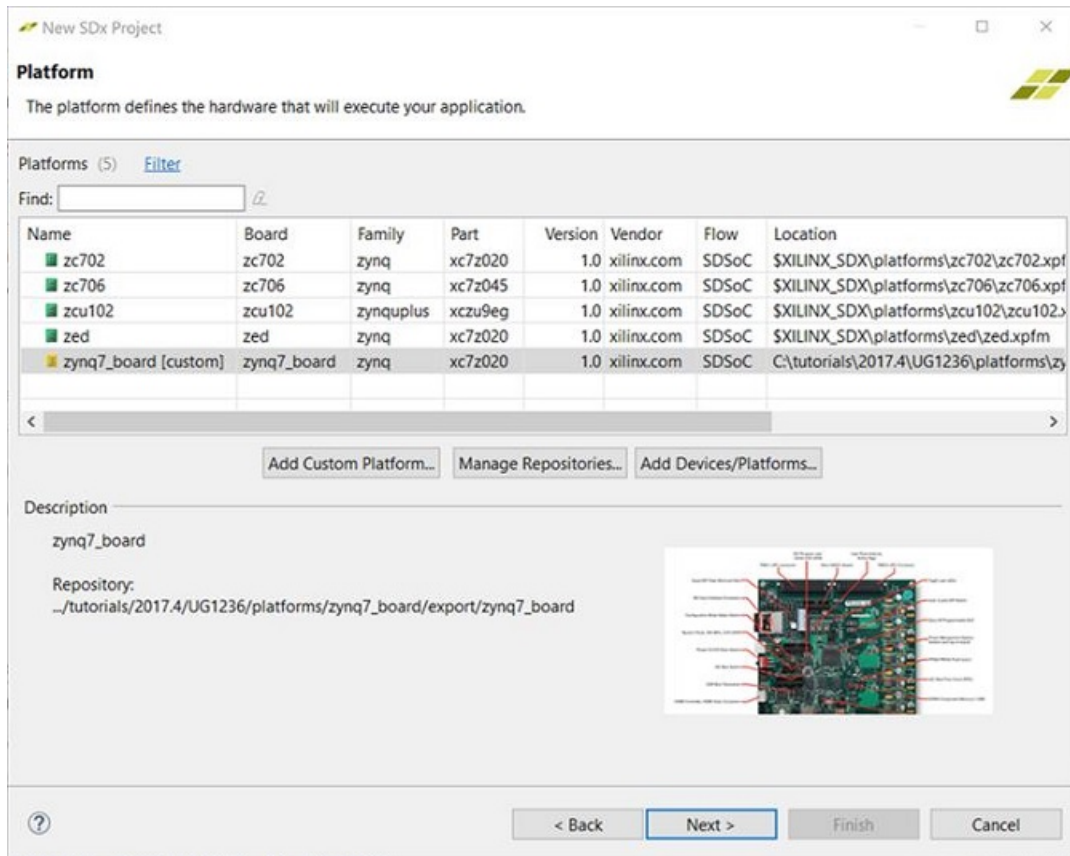
Hypervisor Guest:

Board Support Package: ☒ Create New

☐ Use existing

4. Click **Next**
5. In the Platform page, shown below, select the **zynq7\_platform [custom]** created earlier, and click **Next**.

Figure 53: Select the Custom Platform Created Earlier

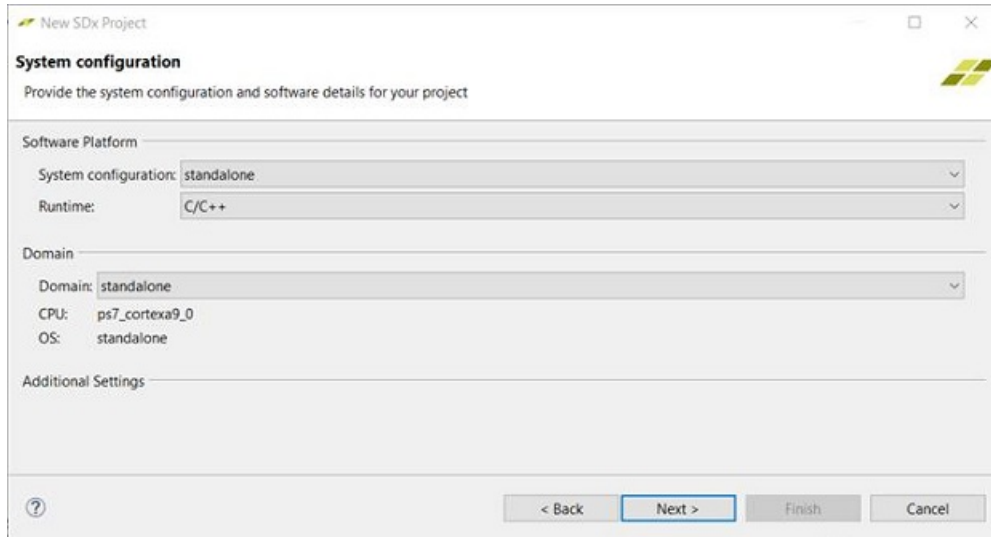


- The System configuration page shows the **System Configuration** as standalone and Runtime as C/C++.

Because the platform supports only the standalone configuration at this point, those are the only available choices. Likewise, the Domain information is auto-populated based on what this platform supports.

- Click **Next**.

Figure 54: Specify System Configuration for the Application



New SDx Project

**System configuration**  
Provide the system configuration and software details for your project

Software Platform

System configuration: standalone

Runtime: C/C++

Domain

Domain: standalone

CPU: ps7\_cortexa9\_0

OS: standalone

Additional Settings

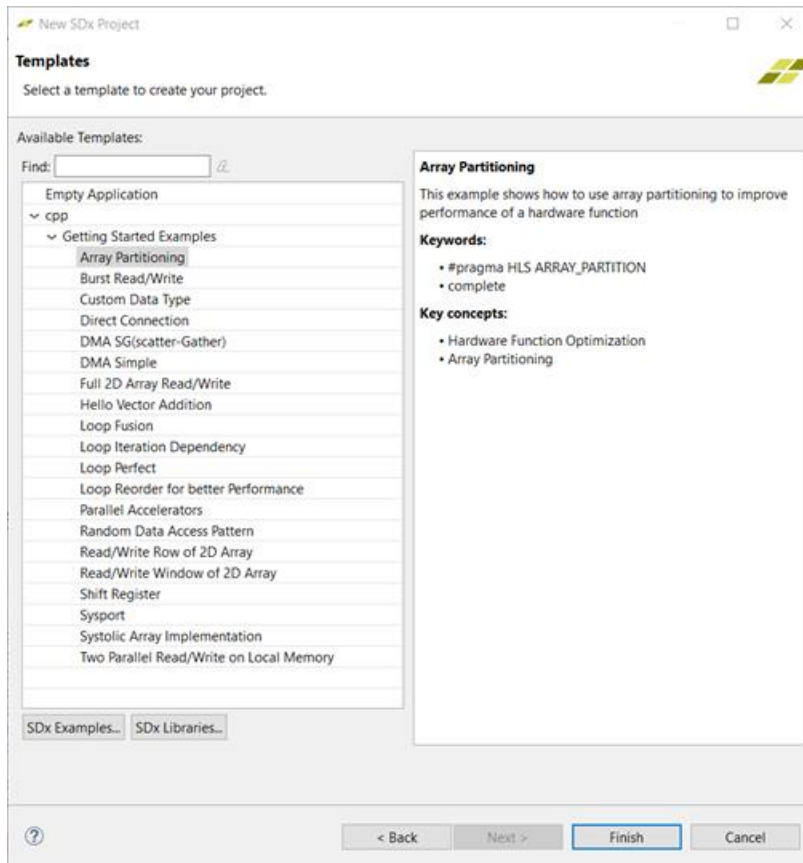
? < Back Next > Finish Cancel

- From the Templates page, select **Array Partitioning** example and click **Finish**. See the following figure.

**Note:** If the Templates page is blank, install the Examples as described in the *SDSoC Environment User Guide* (UG1027) in the Getting Started with Examples Appendix.



Figure 55: Select Application Example from one of the Available Templates



9. At this point you can compile your application and run the program on the ZC702 board. See the *SDSoC Environment Tutorial* ([UG1028](#)) for more information on compiling and running the application on the target board.

## Conclusion

In this tutorial, you:

- Created a platform project
- Added the software components to the project
- Generated the custom platform
- Added the repository containing the custom platform
- Created an SDx application targeting the custom platform

---

## Lab Files

Lab files are not required.

# Additional Resources and Legal Notices

## Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see [Xilinx Support](#).

## Solution Centers

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

---

## References

These documents provide supplemental material useful with this webhelp:

1. *SDx Environments Release Notes, Installation, and Licensing Guide* ([UG1238](#))
2. *SDSoC Environment User Guide* ([UG1027](#))
3. *SDSoC Environment Optimization Guide* ([UG1235](#))
4. *SDSoC Environment Tutorial: Introduction* ([UG1028](#))
5. *SDSoC Environment Platform Development Guide* ([UG1146](#))
6. [SDSoC Development Environment web page](#)
7. *UltraFast Embedded Design Methodology Guide* ([UG1046](#))
8. *Zynq-7000 All Programmable SoC Software Developers Guide* ([UG821](#))
9. *Zynq UltraScale+ MPSoC Software Developer Guide* ([UG1137](#))
10. *ZC702 Evaluation Board for the Zynq-7000 XC7Z020 All Programmable SoC User Guide* ([UG850](#))
11. *ZCU102 Evaluation Board User Guide* ([UG1182](#))
12. *PetaLinux Tools Documentation: Workflow Tutorial* ([UG1156](#))
13. *Vivado Design Suite User Guide: High-Level Synthesis* ([UG902](#))

14. *Vivado Design Suite User Guide: Creating and Packaging Custom IP* ([UG1118](#))
15. [Vivado® Design Suite Documentation](#)

---

## Please Read: Important Legal Notices

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

### **AUTOMOTIVE APPLICATIONS DISCLAIMER**

AUTOMOTIVE PRODUCTS (IDENTIFIED AS "XA" IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE ("SAFETY APPLICATION") UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD ("SAFETY DESIGN"). CUSTOMER SHALL, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.

## Copyright

© Copyright 2017-2018 Xilinx®, Inc. Xilinx®, the Xilinx® logo, Artix, ISE, Kintex, Spartan, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. OpenCL and the OpenCL logo are trademarks of Apple Inc. used by permission by Khronos. All other trademarks are the property of their respective owners.

