Tutorial: Building an Embedded Processor System on a Xylinx Zynq FPGA (Profiling)

Shawki Areibi, Matt Saunders

October 28, 2019

Contents

Introduction

This tutorial will guide you through the process of using Vivado and IP Integrator to create a complete Zynq ARM Cortex-A9 based processor system targeting the ZedBoard Zynq development board. You will use the Block Design feature of IP Integrator to configure the Zynq PS and add IP to create the hardware system, and SDK to create an application to verify the design functionality. It will also guide you through the process of profiling an application and analyzing the output.

Objectives

After completing this tutorial, you will be able to:

- Create an embedded system design using Vivado and SDK flow
- Configure the Processing System (PS)
- Add Xilinx standard IP in the Programmable Logic (PL) section
- Use and route the GPIO signal of the PS into the PL using EMIO
- Use SDK to build a software project and verify the functionality in hardware.
- Set up the board support package (BSP) for profiling an application
- Set the necessary compiler directive on an application to enable profiling
- Setup the profiling parameters

Procedure

This lab is separated into steps that consist of general overview statements that provide information on the detailed instructions that follow. Follow these detailed instructions to progress through the tutorial. This tutorial comprises three stages (each consisting of several steps): You will create a top-level project using Vivado, create the processor system using the IP Integrator, add two instances of the GPIO IP, validate the design, generate the bitstream, export to the SDK, create an application in the SDK, and, test the design in hardware. You will then be able to profile the application and produce statistics that will help you understand the main bottlenecks of your application.

Requirements

The following is needed in order to follow this tutorial:

- Vivado **2016.3** with Xilinx SDK
- ZedBoard, **version D.**

Part 1: Building a Zynq-7000 Processor Hardware

Introduction

In this part of the tutorial you create a Zynq-7000 processor based design and instantiate IP in the processing logic fabric (PL) to complete your design. Then you take the design through implementation, generate a bitstream, and export the hardware to SDK.

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

Step 1: Start the Vivado IDE and Create a Project

1. Start the Vivado IDE (Figure 1) by opening the program from the Start Menu.

Figure 1: Vivado 2016.3 Getting Started screen.

2. From the Getting Started screen, select **Create New Project**. The New Project Wizard opens (Figure 2).
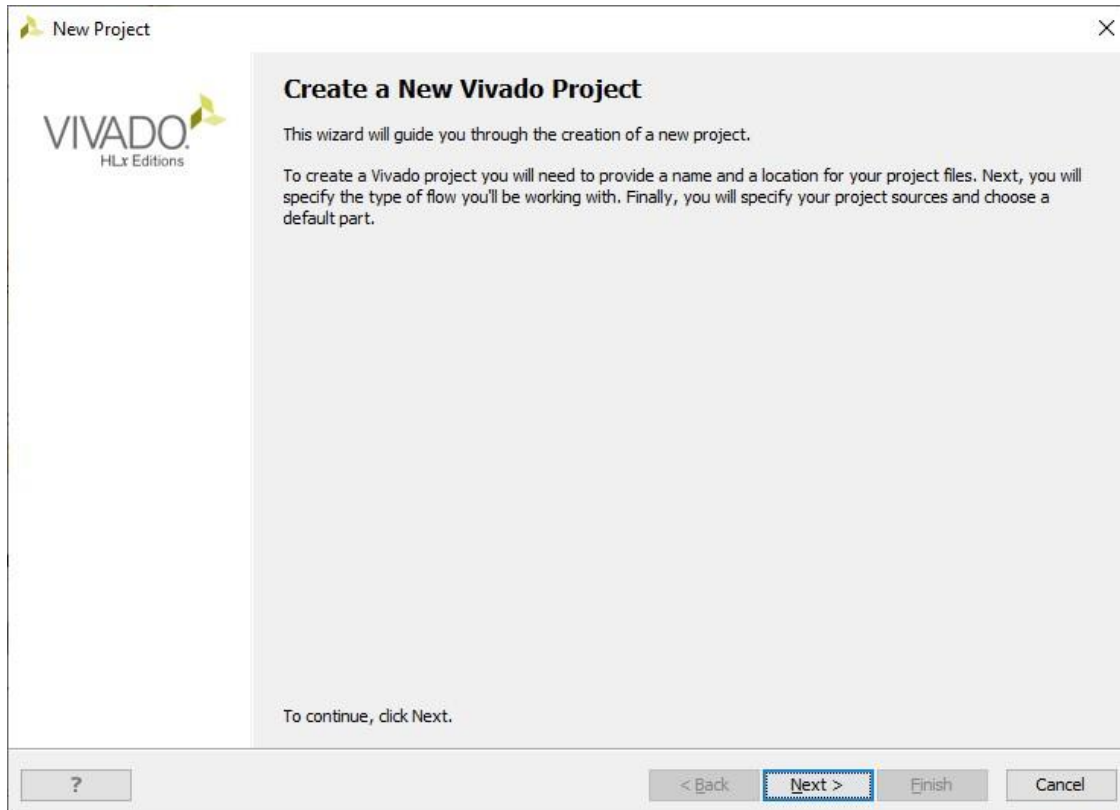3. Click **Next**.

Figure 2: Create New Project Wizard.

4. In the **Project Name** dialog box, type the project name and location. Ensure that **Create Project Subdirectory** is selected, and click **Next**.
5. In the **Project Type** dialog box, select **RTL Project**, then click **Next**.
6. In the **Add Sources** dialog box, select **RTL Project**, then click **Next**.
7. In the **Add Existing IP** dialog box, click **Next**.
8. In the **Add Constraints** dialog box, click **Next**.
9. In the **Default Part** dialog box, select **Boards** and choose the **ZedBoard Zynq Evaluation and Development Kit**. Make sure that you have selected the proper Board Version to match your hardware. Click **Next**.
10. Review the project summary in the **New Project Summary** dialog box before clicking **Finish** to create the project.

Step 2: Create an IP Integrator Design

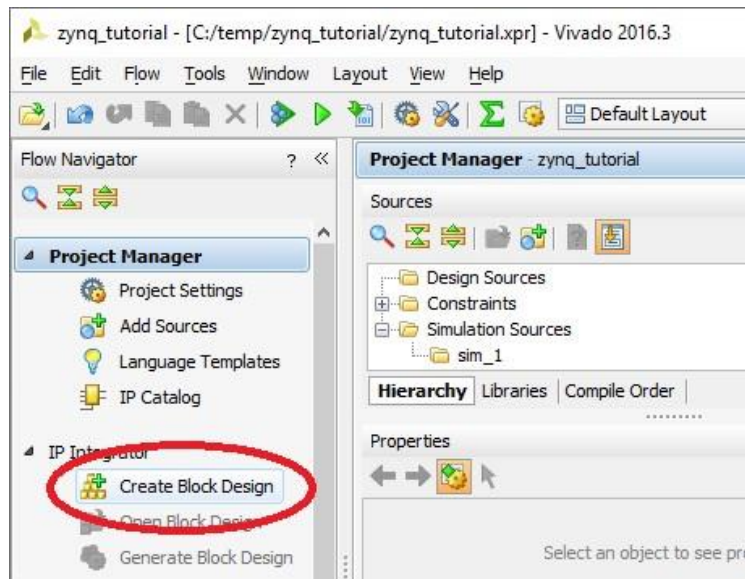1. In the Flow Navigator, select **Create Block Design** (Figure 3).

Figure 3: Create Block Design option in Flow Navigator.

2. In the **Create Block Design** dialog box, specify a name for your IP subsystem design (Figure 4).
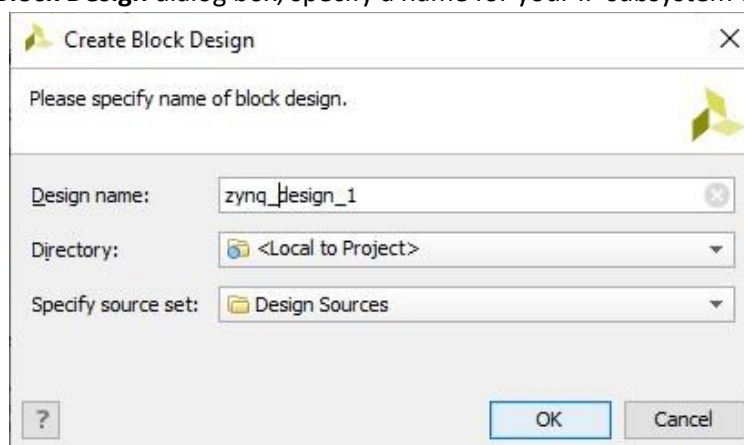

Figure 4: Create Block Design dialog box.

3. Right-click in the Vivado IP Integrator workspace, and select **Add IP** (Figure 5).
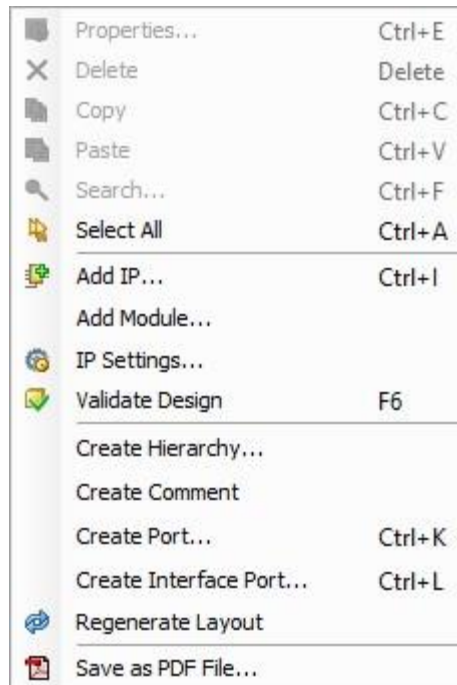
Figure 5: Add IP.

4. In the search field, type **zynq** to find the ZYNQ7 Processing System IP, and then press **Enter** on the keyboard (Figure 6).
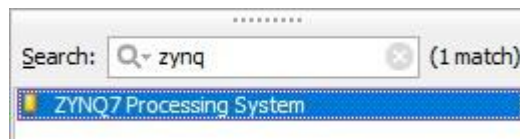

Figure 6: The IP Integrator IP Catalog.

Because you selected the ZedBoard when you created the project, the Vivado IP Integrator configures the design appropriately.

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

There is a corresponding Tcl command for all actions performed in the IP Integrator. Those commands are not shown in this document; see the Tcl Console for each action for information those commands.

5. In the IP Integrator workspace header, click **Run Block Automation** (Figure 7).

Figure 7: Run Block Automation.

6. The **Run Block Automation** dialog box opens (Figure 8), allowing you to select the interfaces to connect to the ZYNQ7 core. Click **OK**.
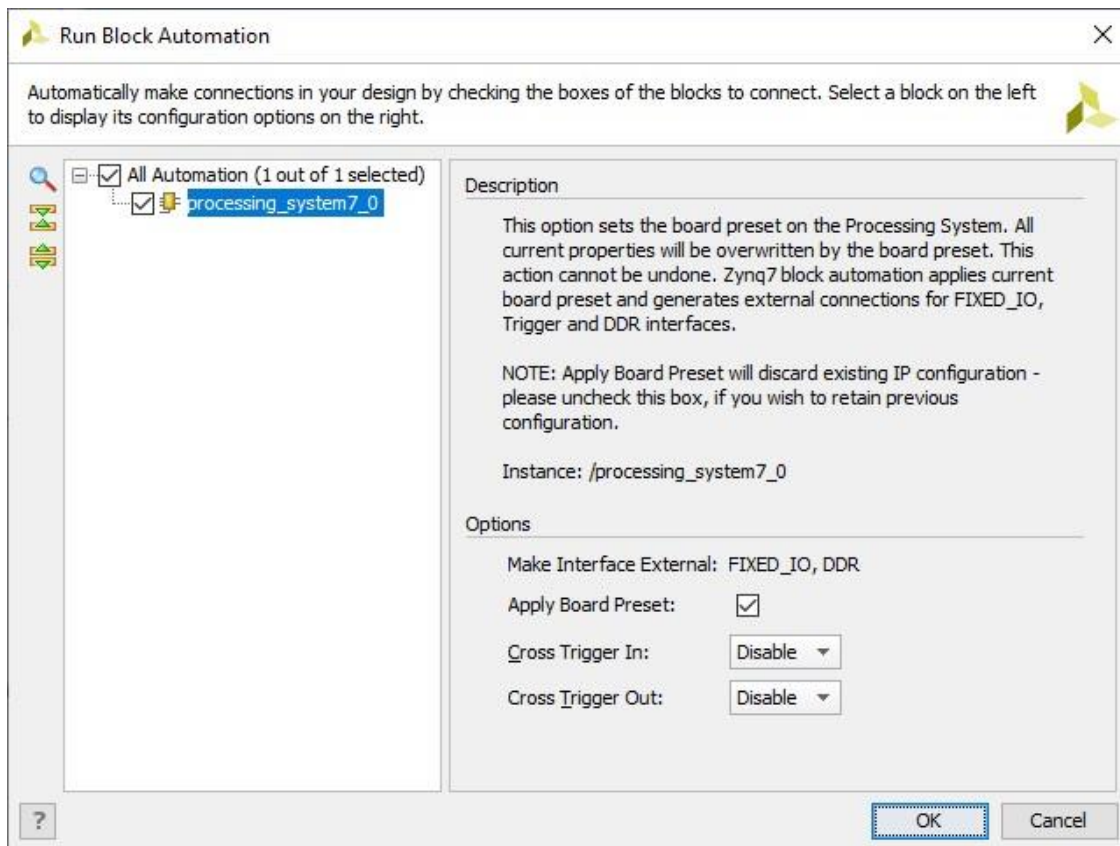

Figure 8: Block Automation dialog box.

After running block automation on the Zynq processor, the IP Integrator block diagram should look as follows (Figure 9).
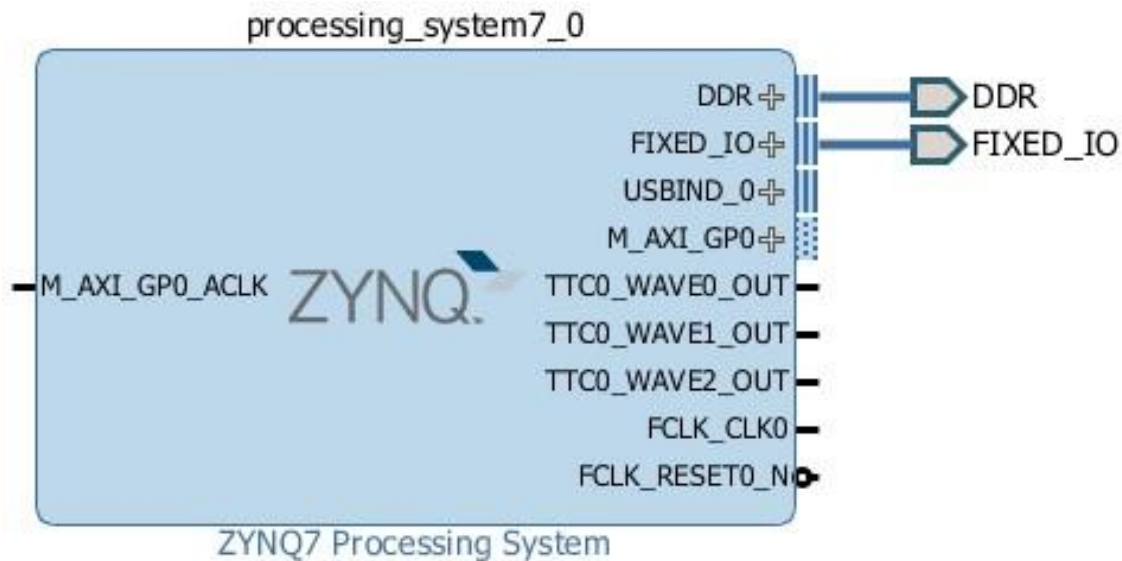
Figure 9: Zynq Processing System after running Block Automation.

7. Now you can add peripherals to the processing logic (PL). To do this, right-click the IP Integrator diagram and select **Add IP**.
8. In the search field, type gpi to find the AXI GPIO IP, and then press **Enter** to add the AXI GPIO IP to the design.
9. Repeat the action, typing axi bram to find the AXI BRAM Controller, and typing block to find and add the Block Memory Generator. The Block Design window matches Figure 10. The relative positions of the IP will vary.
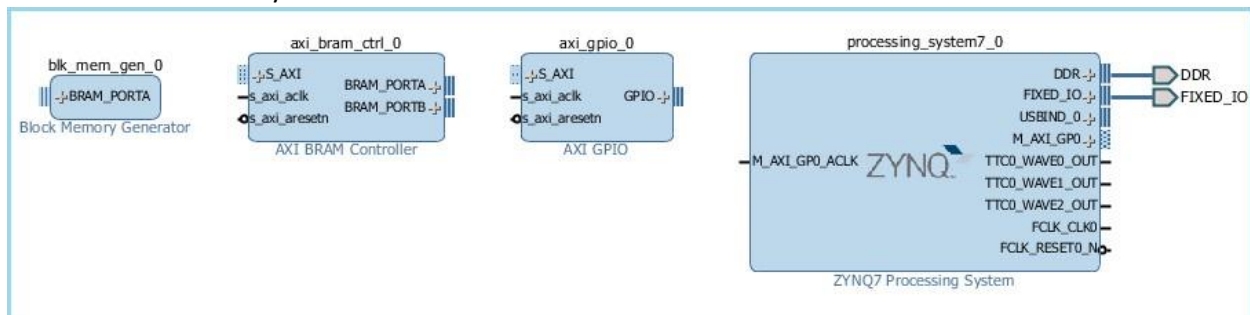

Figure 10: Block Design after instantiating IP.

Customize Instantiated IP

1. Double-click the Block Memory Generator IP to open the **Re-customize IP** dialog box.
2. In the Basic tab of the dialog box (Figure 11), set:

- **Mode** to **BRAM Controller**
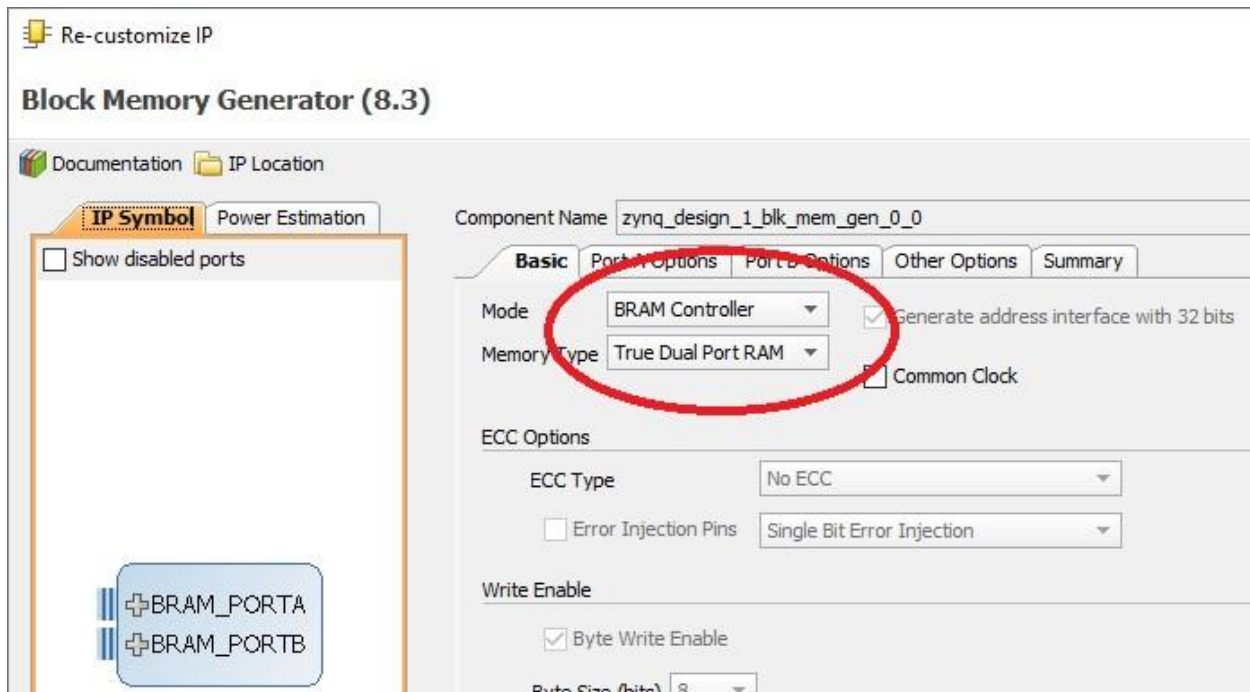- **Memory Type** to **True Dual Port RAM**.

Click **OK**.

Figure 11: Customize Block option.

3. Connect the Block Memory Generator to the AXI4 BRAM Controller by clicking the connection point and dragging a line between the two IP blocks (Figure 12).
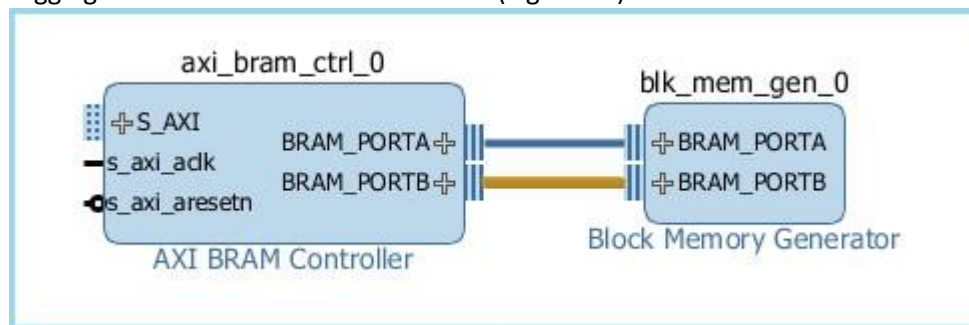

Figure 12: Connected AXI BRAM Controller and Block Memory Generator.

The AXI BRAM controller provides an AXI memory map interface to the Block Memory Generator.

Use Block Designer Assistance

Block Designer Assistance helps connect the AXI GPIO and AXI BRAM Controller to the Zynq-7000 PS.

1. In the IP Integrator workspace header, Click **Run Connection Automation** (Figure 13 to open the Run Connection Automation dialog box.
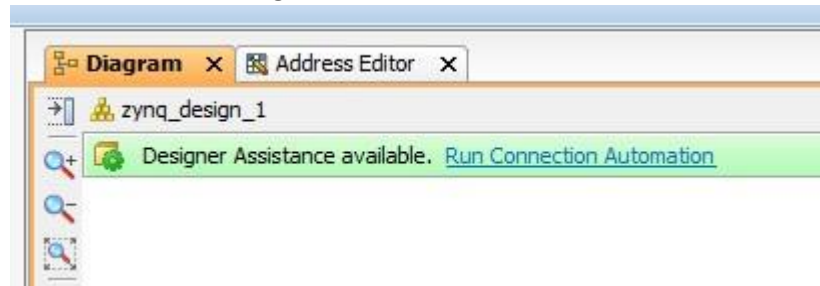

Figure 13: Connected AXI BRAM Controller and Block Memory Generator.

2. In the Run Connection Automation dialog box, under **axi bram ctrl #**, select **S AXI**. Ensure itscheckbox is selected, then for the Clock Connection, select **Auto**.
3. Click **OK**. This action instantiates an AXI Interconnect IP as well as a Processor System Reset IP, and makes the interconnection between the AXI interface of the GPIO and the Zynq-7000 PS.
4. Select **Run Connection Automation** again. Under **axi gpio #**, select the check-box beside **GPIO**. Under **Select Board Interface**, select **leds 8bits**, as in Figure 14.
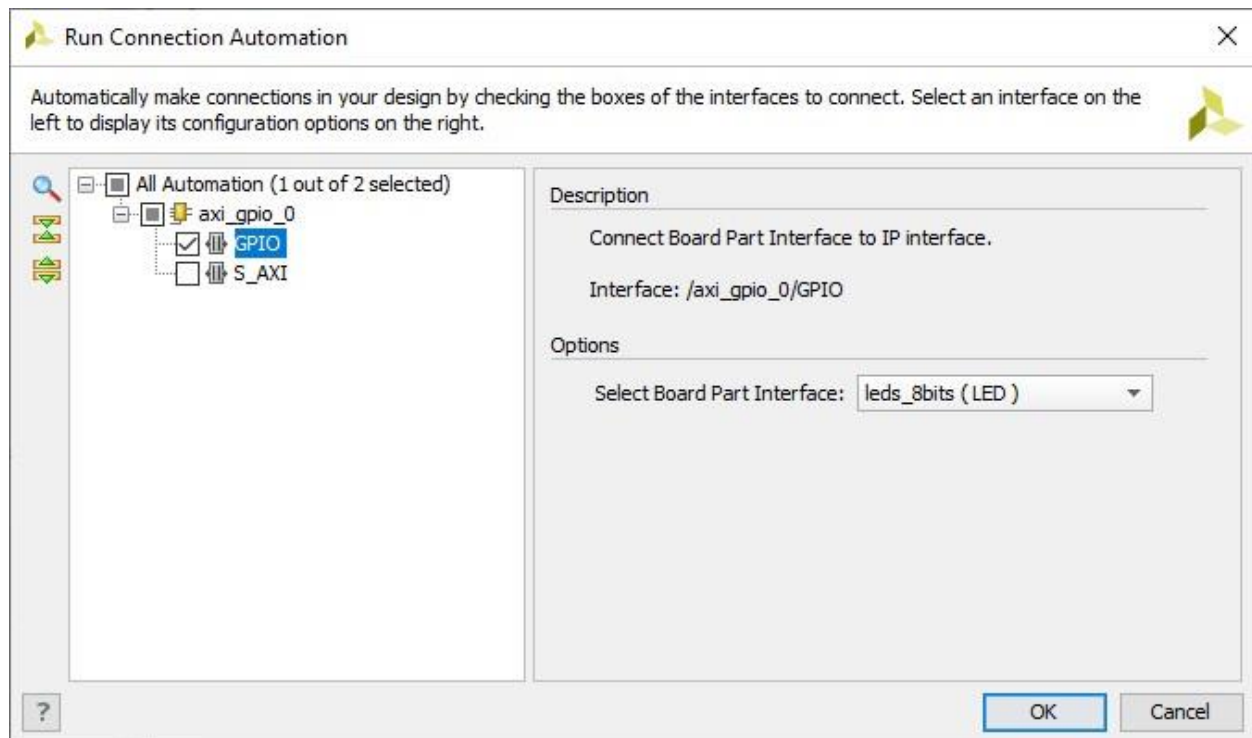
Figure 14: Select Board Interface options.

5. Click OK. This step also configures the IP so that during netlist generation, the IP will create the necessary Xilinx Design Constraints (XDC).

6. Select **Run Connection Automation** one last time, to connect the GPIO to the AXI bus. Again, select the check-box beside **S AXI**, and for **Clock Connection**, select **Auto**.

Editing the Memory Map

This completes the connections between the Zynq7 Processing System and the peripherals. The IP Integrator block diagram should look something like Figure 15, though the positions of the IP can vary.
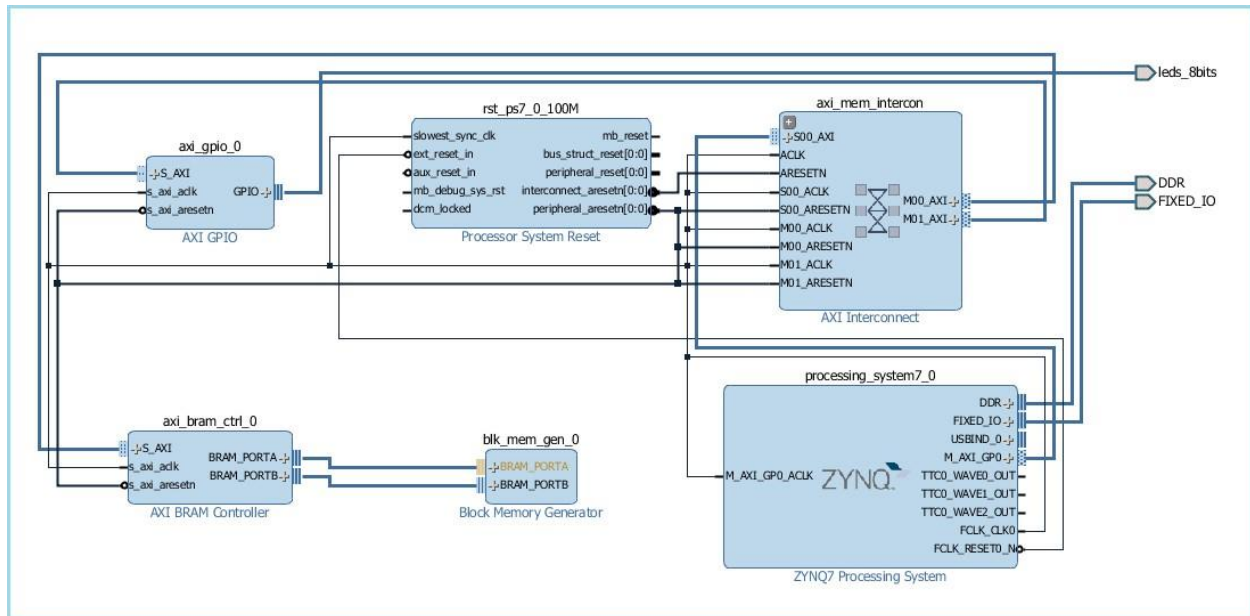
Figure 15: Zynq Processor System, connected to peripherals.

1. Click the Address Editor tab to show the memory map of all the IP in the design. In this case, there are two IP: the AXI GPIO and the AXI BRAM Controller. The IP Integrator assigns the memory maps for these IP automatically, but you can change them if necessary.

2. Change the range of the AXI BRAM Controller to 64K, as shown in Figure 16.

Figure 16: Setting AXI BRAM Controller to 64K range.

3. Save your design with Ctrl-S or by selecting File → Save Block Design.

4. From the toolbar, run Design Rules Check by clicking the **ValidateDesign** button (Figure17). Alternatively, you can select Tools → Validate Design from the menu, or you can right-click in the workspace and select **Validate Design**.
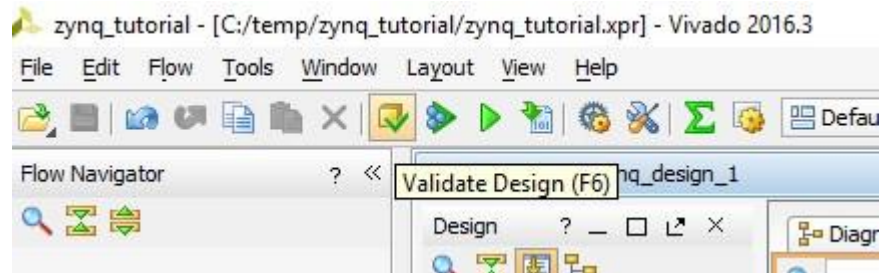

Figure 17: Validating the design.

5. The Validate Design Successful dialog box should appear. Click OK.

Step 3: Generate HDL Design Files

Now, you will generate the HDL files for the design.

1. In the Sources window, right-click the top-level subsystem design and select **Generate Output Products** (Figure 18). This generates the source files for the IP used in the block diagram and the relevant constraints file.
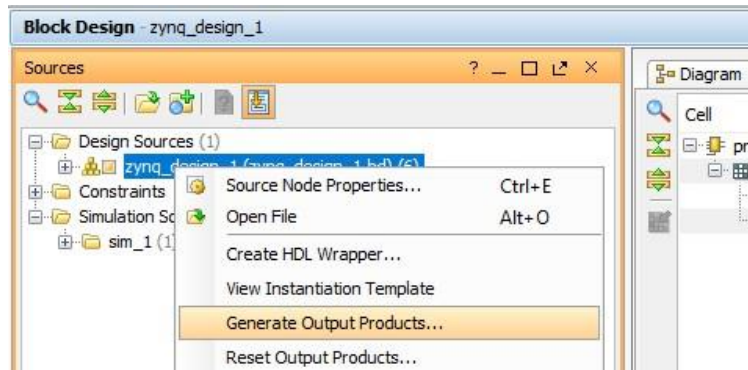

Figure 18: Generating output products.

2. The **Generate Output Products** window opens (Figure 19). Leave the default settings and click **OK**.
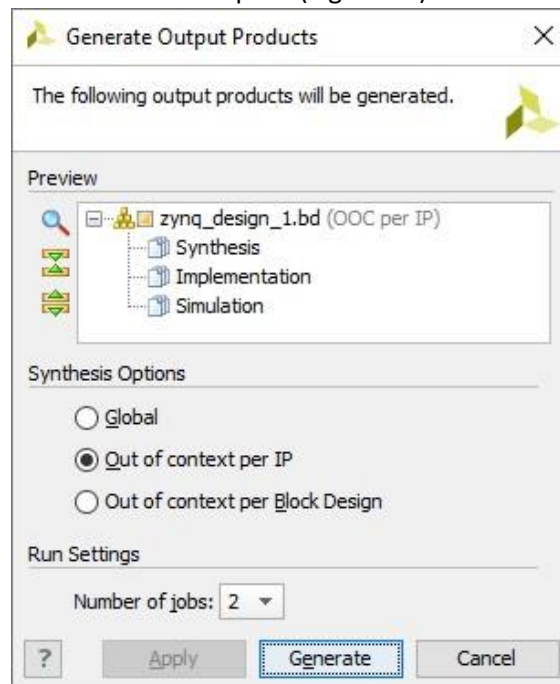

Figure 19: Generating output products.

3. In the Sources window, select the top-level subsystem source, and select **Create HDL Wrapper** to create an example top-level HDL file (Figure 20)). Select **Let Vivado manage wrapper and autoupdate** and click **OK**.
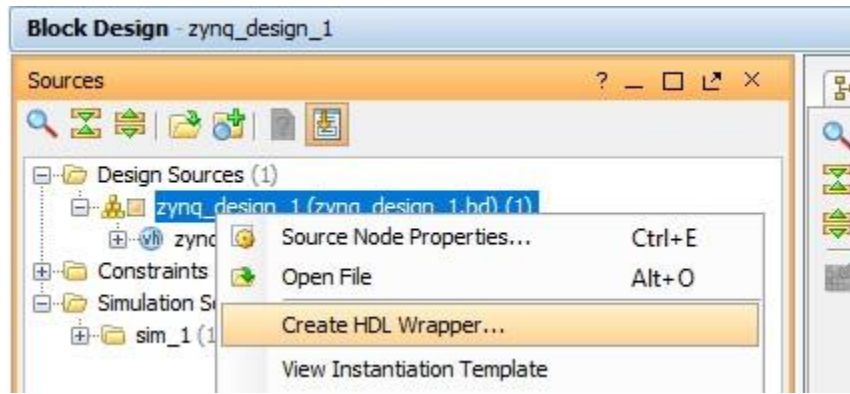

Figure 20: Creating the HDL wrapper.

Step 4: Implement Design and Generate Bitstream

1. In Flow Navigator, click **Generate Bitstream** to implement the design and generate a BIT file. *Note: If the system requests to re-synthesize the design before implementing, click* **No**. The previous step of saving the constraints caused the flow to mark synthesis out-of-date. Ordinarily, you might want to re-synthesize the design if you manually changed the constraints, but for this tutorial, it is safe to ignore this condition.

    You may also see a dialog box stating that no implementation results are available. Click **Yes** to proceed with synthesis, implementation and bitstream generation.

2. After the design implementation, the Bitstream Generation Completed window appears (Figure 21). Select **Open Implemented Design** and then click **OK**. You may get a warning that the implementation is out of date— click **Yes**.
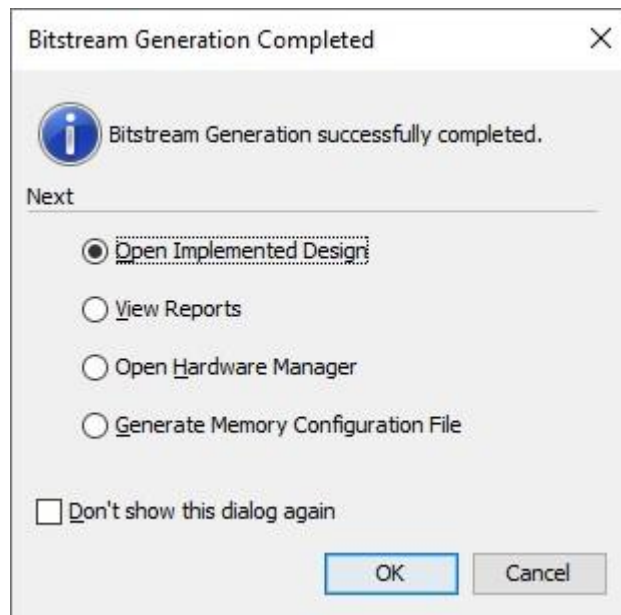

Figure 21: Bitstream generation completed.

Step 5: Export Hardware to SDK

In this step, you export the hardware to description to the Xilinx Software Development Kit (SDK) for use in Part 2. Both the IP Integrator block diagram and the implemented design must be open to export the design to the SDK.

   IMPORTANT: for the Digilent driver to install, you must power on and connect the board to the host PC before launching SDK.

Export to SDK

1. In the Flow Navigator, click **Open Block Design** to invoke the IP Integrator design.

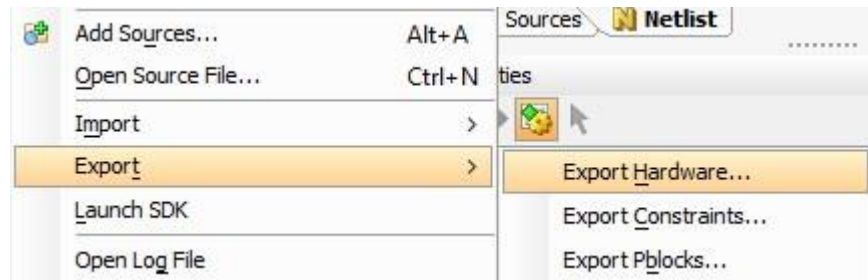2. On the Menu Bar, select File → Export → Export Hardware (Figure 22).

Figure 22: Export Hardware menu option.

3. The Export Hardware dialog box opens. Ensure that **Include bitstream** is selected, then click **OK**.
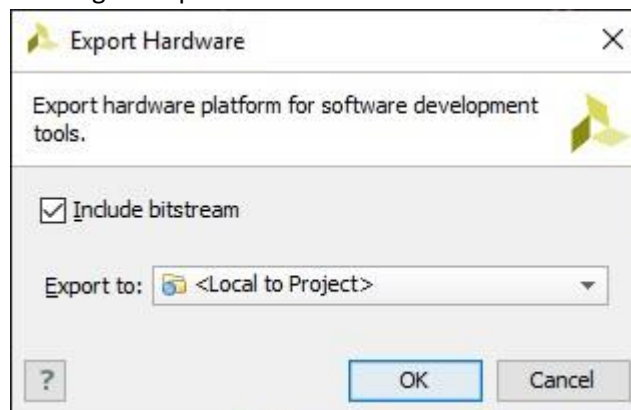

Figure 23: Export Hardware dialog box.

4. On the Menu Bar, select File → Launch SDK. Click **OK** in the dialog box to launch SDK.

Part 2: Build Zynq-7000 Processor Software

In this portion of the tutorial you will built an embedded software project that prints "Hello, World" to the serial port. Connect two micro USB cables and the 12V power adapter to the ZedBoard (see Appendix A for more details).

Step 1: Start SDK and Create a Software Application

1. If you are doing this lab as a continuation of Part 1 then SDK should have launched in a separate window. You can also start SDK from the Windows Start Menu by clicking on ⊞ → **Xilinx Design Tools** → **Xilinx SDK 2016.3**.
2. When launching SDK from the Start Menu, you must select the correct workspace. You can select the workspace by clicking on **File** → **Switch Workspace** → **Other** in SDK. In the Workspace Launcher dialog box, in the Workspace field, point to the **SDK Export** folder where you had exported your

hardware. Usually, this is located at ..\project name\project _name.sdk\SDK\SDK Export. Now you can create a "hello, world" application.
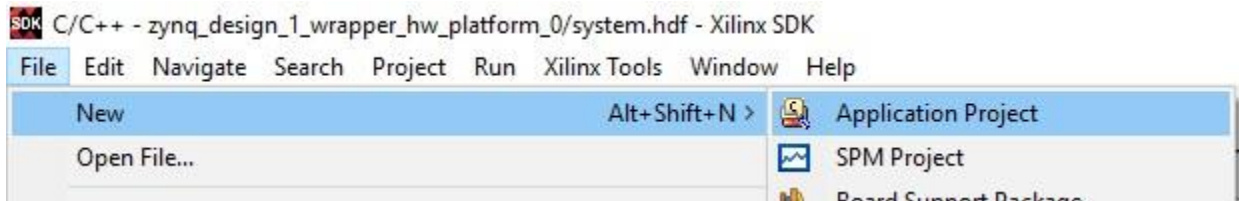
3. Select **File → New → Application Project** (Figure 24).

Figure 24: New Application Project.

4. In the Project Name field, type **Zynq Design**, and click **Next** (**not** Finish), as in Figure 25.

Figure 25: Setting up the new Application Project.

5. From the available templates, select **Hello World** (Figure 26), and click **Finish**.
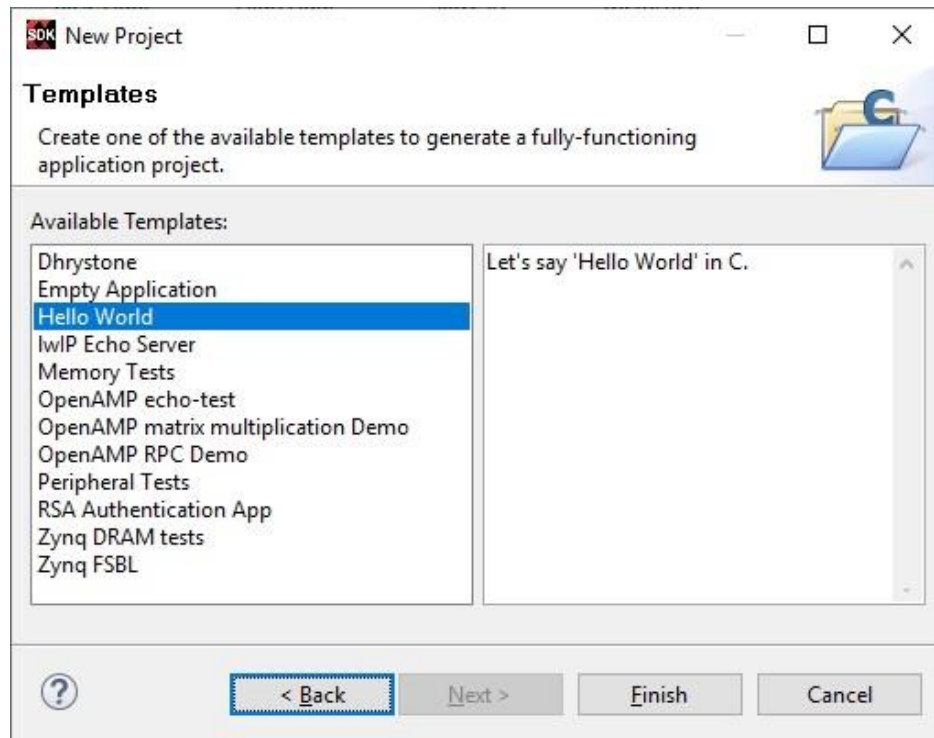
Figure 26: Selecting the new project template.

6. The program will begin compiling in the SDK. When it finishes compiling, you will see the console messages shown in Figure 27.
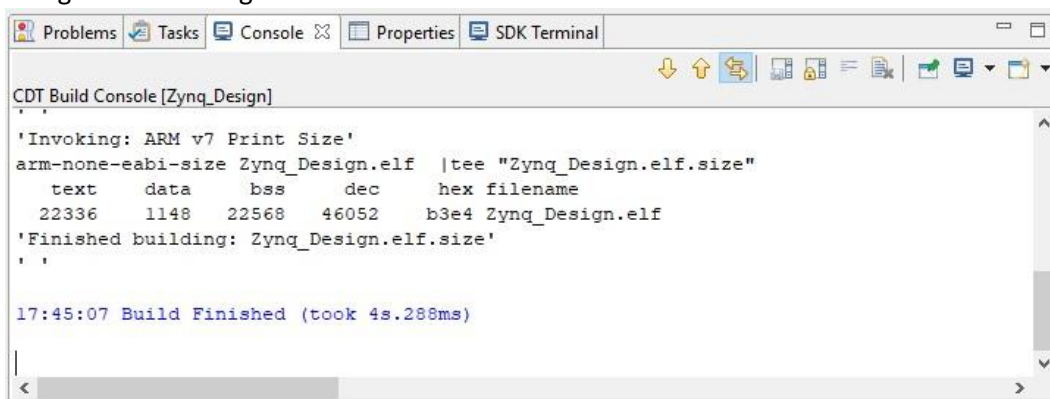

Figure 27: Compilation messages in the SDK Console.

Step 2: Run the Software Application

Now, you must run the "Hello, World" application on the ZedBoard. Make sure that your hardware is powered on and a USB cable is connected to the PROG port of the ZedBoard. Also ensure you have a USB cable connected to the UART port of the ZedBoard. Please check Appendix A for more guidelines.

1. Download the bitstream to the FPGA by selecting **Xilinx Tools → Program FPGA** from the menu bar (Figure 28).
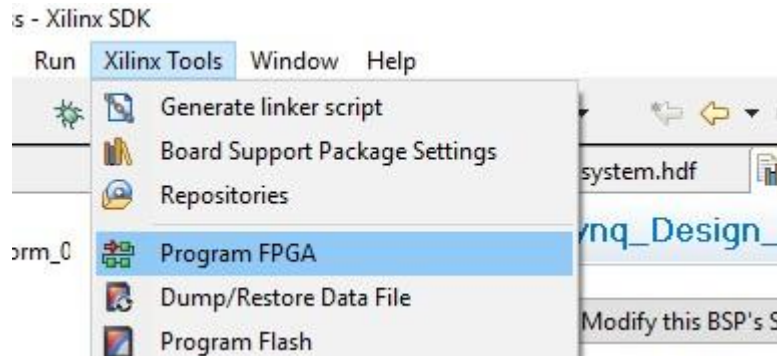

Figure 28: Program FPGA menu option.

2. Ensure that the path to the bitstream (created in Part 1) is correct, then click **Program**. The DONE LED on the board turns blue if the programming is successful.
3. In the Project Explorer pane, select and right-click the **Zynq Design** application.
4. Select **Debug As → Debug Configurations** (Figure 29).

Project Explorer ⊠

> Zynq_Design
> zy
  .c
  .c
  .c
  .c

New	>
Go Into	
Open in New Window	
Copy	Ctrl+C
Paste	Ctrl+V
Delete	Delete
Source	>
Move...	
Rename...	F2
Import...	
Export...	
Build Project	
Clean Project	
Refresh	F5
Close Project	
Close Unrelated Projects	
Build Configurations	>
Run As	>
Debug As	>
Compare With	>
Restore from Local History...	
C/C++ Build Settings	
Generate Linker Script	
Change Referenced BSP	
Create Boot Image	
Team	>
Properties	Alt+Enter

Debug As submenu:
1 Launch on Hardware (System Debugger)
2 Start Performance Analysis
3 Launch on Hardware (System Debugger on QEMU)
4 Launch on Hardware (GDB)
5 Local C/C++ Application
Debug Configurations...

system.hdf   system.mss ⊠

Zynq_Design_bsp Board Sup

Modify this BSP's Settings   Re-generate

Target Information

This Board Support Package is compiled to

Hardware Specification: C:\temp\zynq_tu
Target Processor: ps7_cortexa9_0

Operating System

Board Support Package OS.

Name: standalone
Version: 6.0
Description: Standalone is a simple, l
environment, such as st
Documentation: standalone v6 0

Peripheral Drivers

Drivers present in the Board Support Packa

axi_bram_ctrl_0 bram
axi_gpio_0 gpio
ps7_afi_0 generic

ps7_dma_s dmaps
ps7_ethernet_0 emacps
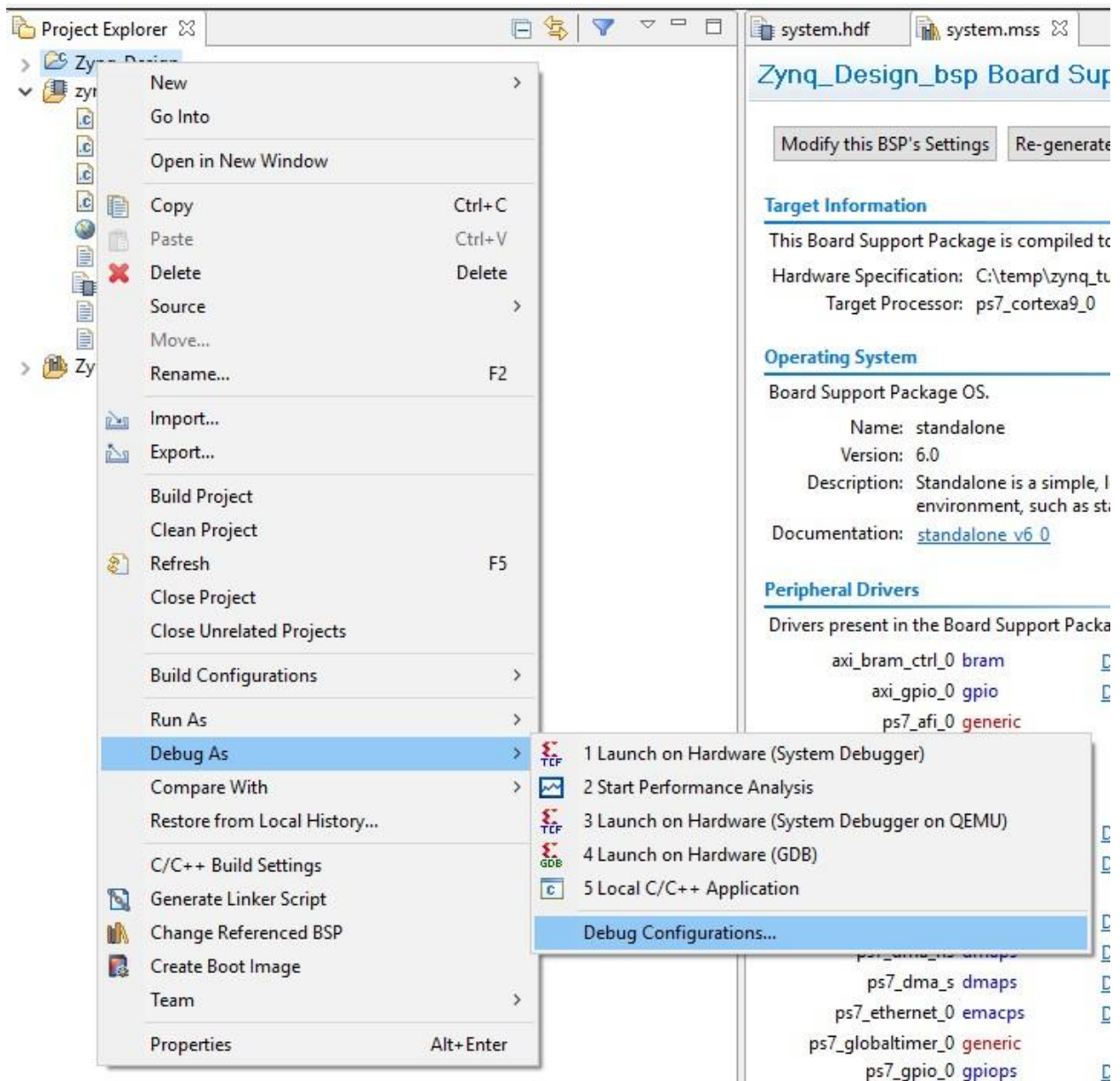ps7_globaltimer_0 generic
ps7_gpio_0 gpiops

Figure 29: Debug Configurations menu option.

5. In the Debug Configurations dialog box, right-click **Xilinx C/C++ application (GDB)** and select **New** (Figure 30).
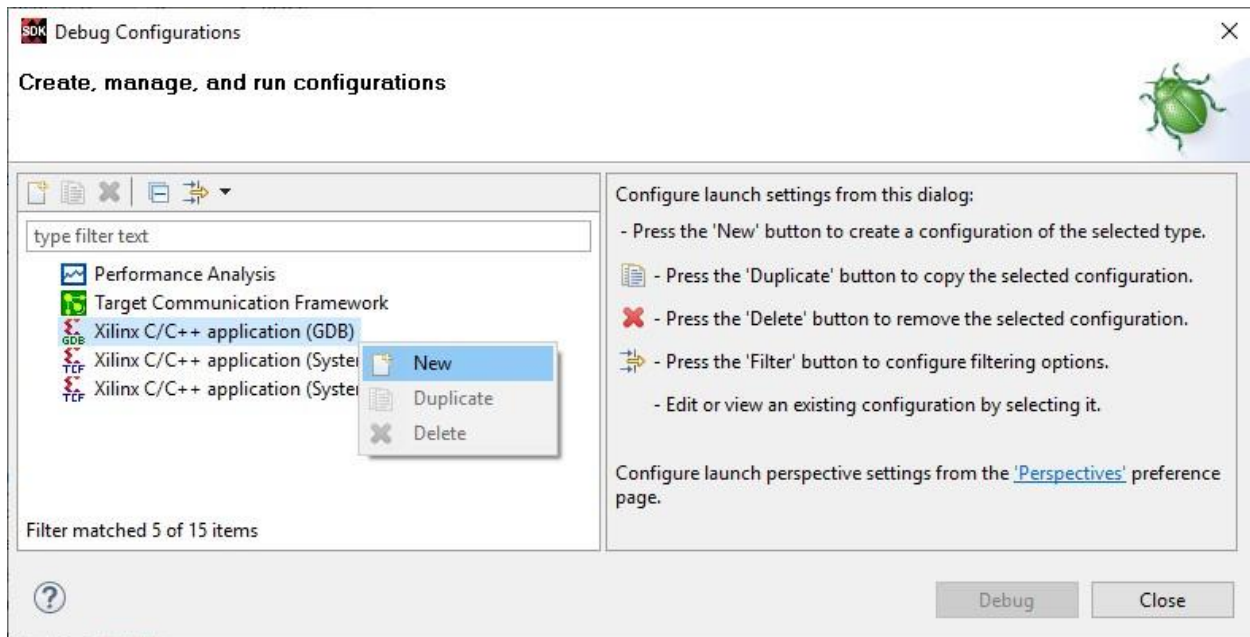
23

Figure 30: Debug Configurations dialog box.

6. In the Debug Configurations dialog box, leave the settings as default, and select **Debug**.
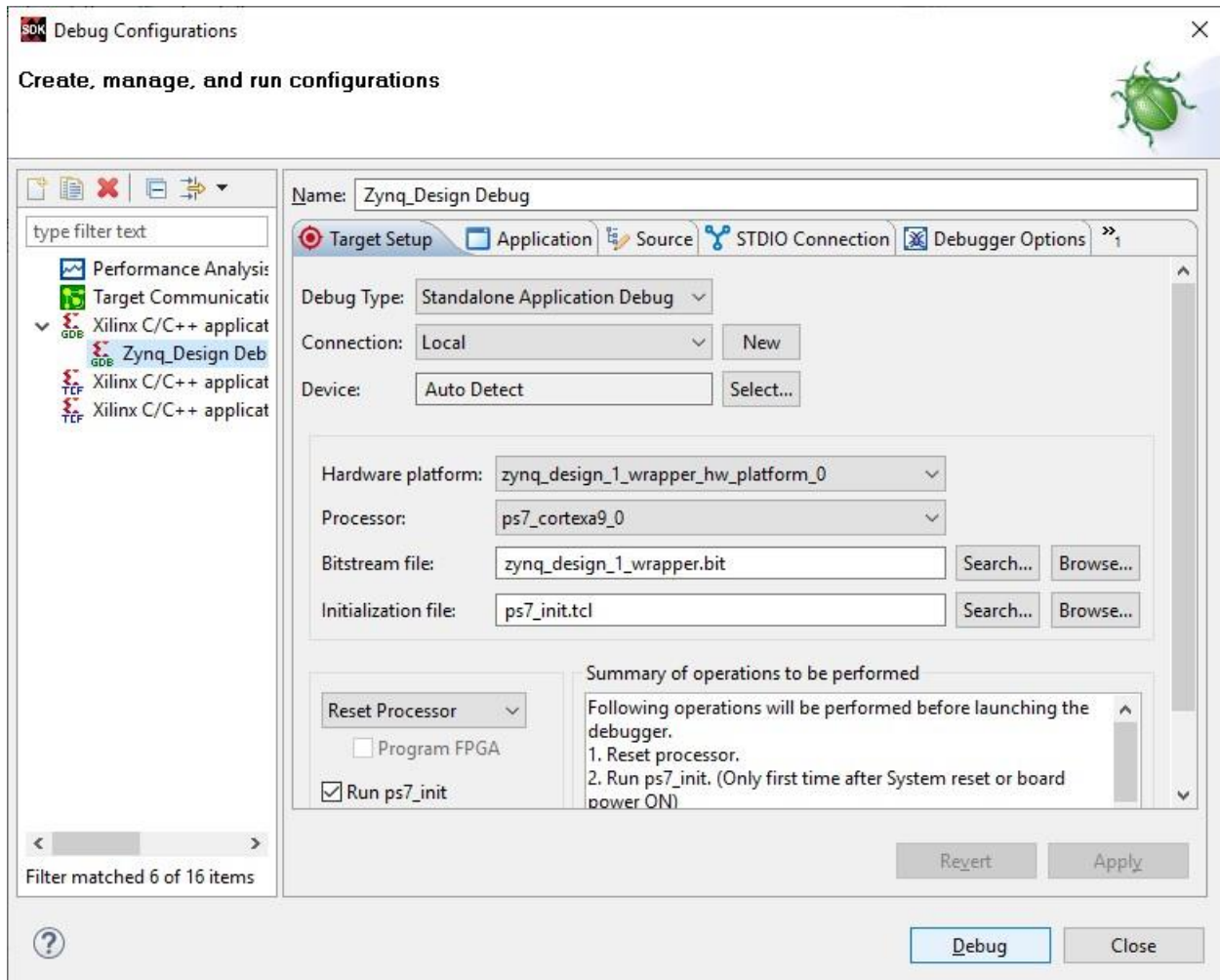
Figure 31: Debug Configurations dialog box.

7. You will be asked to confirm a perspective switch— click **Yes**. The Debug perspective will open.
8. To connect to the ZedBoard serial port, select the **SDK Terminal** tab in the console panel, then click the ✚ to create a new serial connection, as in Figure 32.
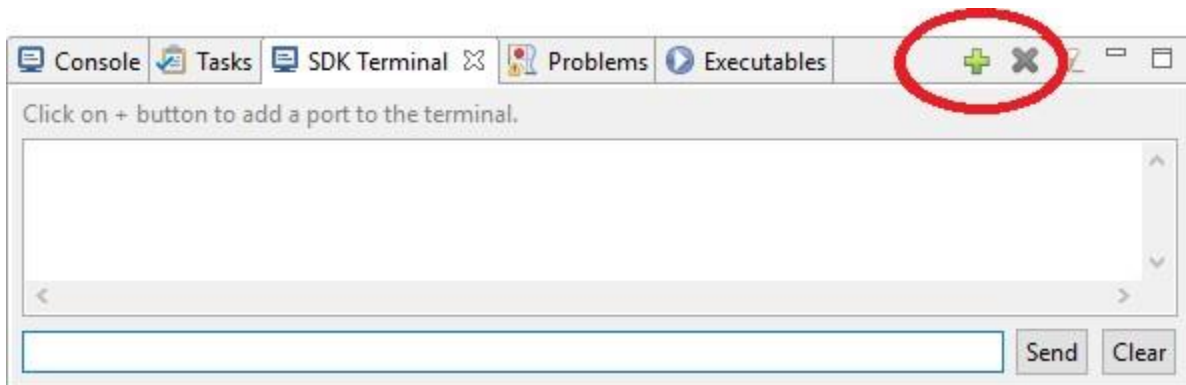
Figure 32: Creating a new serial connection.

9. Use the settings in Figure 33 to set up the serial port. Check the Windows Device Manager on your workstation for the **USB Serial Port** to determine which COM port is connected to the ZedBoard. Once the settings are correct, click **OK**.
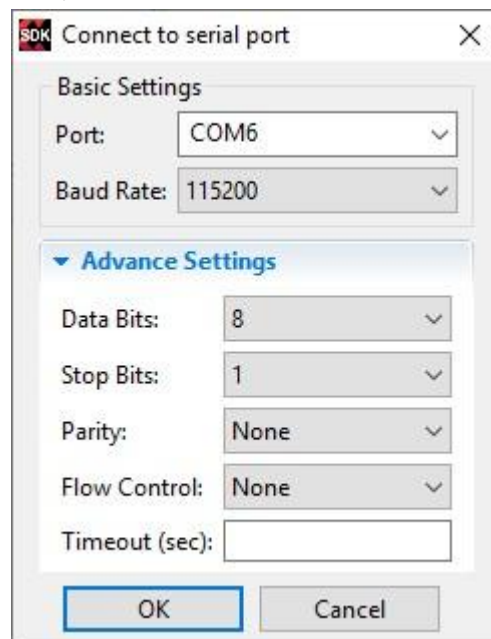

Figure 33: Setting up the serial port.

10. Verify the Terminal connection by checking the status at the top of the tab (Figure 34).
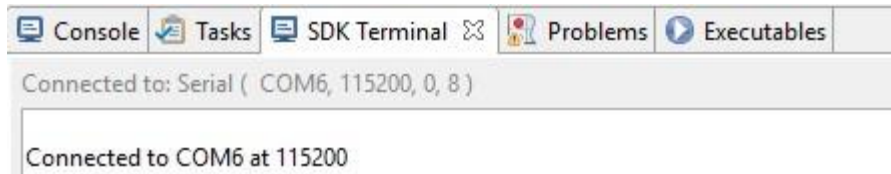
Figure 34: Terminal connection verification.

11. In the **Debug** tab, expand the tree, and select the processor core on which the program is to be run (Figure 35).
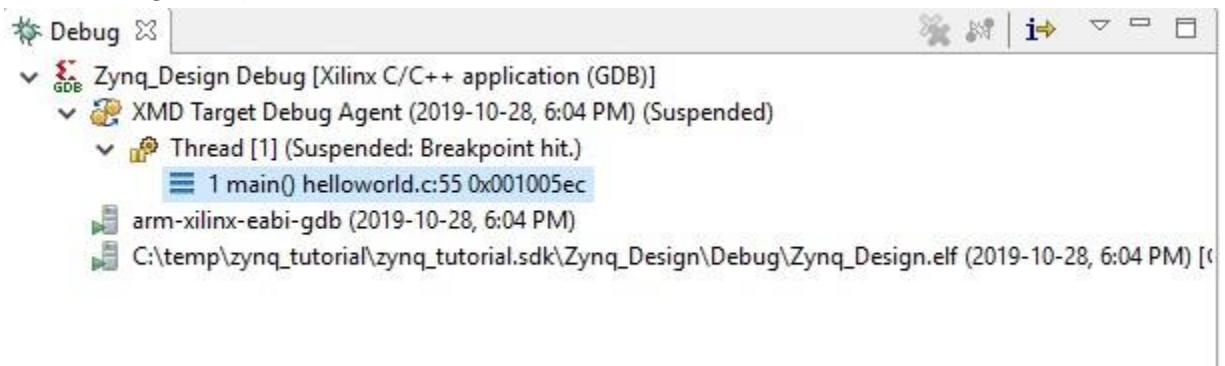

Figure 35: Terminal connection verification.

12. If not already open, select ../src/helloworld.c, and double-click to open the source file.

Add a Break Point

Next, you will add a break point at the line which prints "Hello World."

1. Select **Navigate → Go to Line**. To go to line 57, type **57**.
2. Double-click to the left of Line 57, which adds a break point on that line. (Figure 36)

```
#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"


int main()
{
    init_platform();

    print("Hello World\n\r");

    cleanup_platform();
    return 0;
}
```

Figure 36: Adding a break point.

Step 3: Executing the Software

This step will take you through executing the code up to and past the break point.

1. Click the **Resume** button, or press **F8**.
2. Click the **Step Over** button, or press **F6**.
3. If everything worked correctly, you should see "Hello World" printed in the terminal. (Figure 37)

```
Console  Tasks  SDK Terminal ⊠  Problems  Executables
Connected to: Serial ( COM6, 115200, 0, 8 )

Connected to COM6 at 115200
Hello World
```

Figure 37: Terminal output.

Part 3: Profiling an Application

Step 1: Export the design to the SDK

In this part, you will use the same hardware configuration and bitstream to profile a matrix multiplication application.

1. Follow Part 1 of the tutorial to create a bitstream and export it to SDK.

2. If SDK is still open from the previous step, return to the C/C++ view using the selector in the top right corner of the program.

3. In SDK, select **File → New → Board Support Package**.

4. Leave the default settings; notice the new board support package is named standalone bsp _0. Click **Finish**.

5. The Board Support Package Settings window will appear. Select **Overview → standalone**, click on the drop-down arrow over the enable sw intrusive-profiling *Value* field, and select **true**, as in Figure 38).
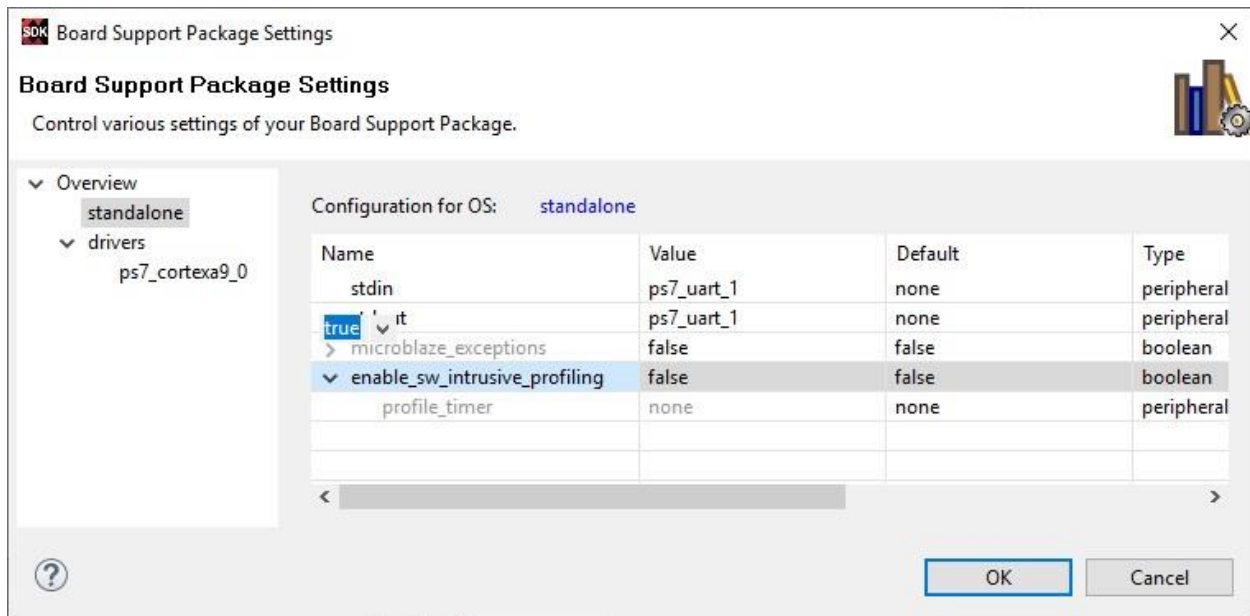
Figure 38: Board Support Package Settings window.

6. Select **Overview → drivers → ps7 cortexa9** and in the extra compiler _flags field, add -g -pg to the front of the other flags, as in Figure 39. The complete field should read:

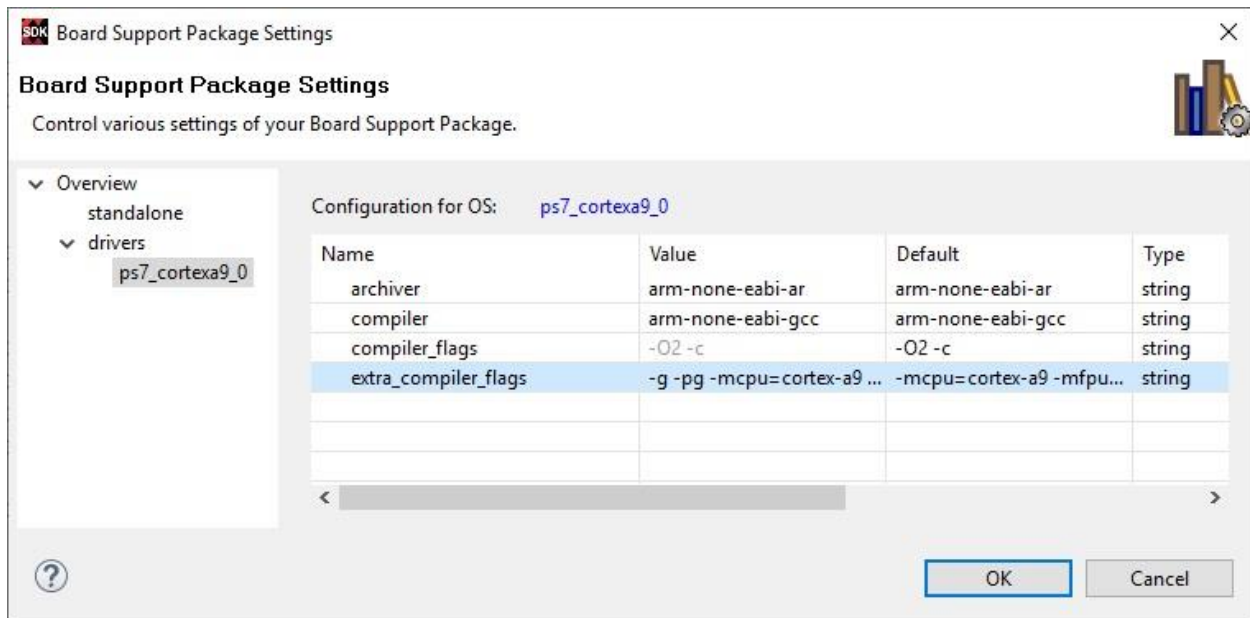-g -pg -mcpu=cortex-a9 -mfpu=vfpv3 -mfloat-abi=hard -nostartfiles

Figure 39: Adding compiler flags for profiler.

Step 2: Create the application

1. Select **File → New → Application Project**.
2. Enter **tutorial-profile** as the project name, select the **Use existing *standalone bsp 0*** option, and click **Next** (*not* Finish).
3. Select **Hello World** from the Available Templates window, and click **Finish**.
4. Replace the Hello World C program with the C code (matrixoperations.c) found on the course website.
5. Save the program and it should compile successfully and generate the tutorial-profile.elf file.

Step 3: Run the Application and Profile

1. Place the ZedBoard in JTAG boot-up mode. See Appendix A for ZedBoard connection details.
2. Power on the board.
3. In SDK, select **Xilinx Tools → Program FPGA** and click on **Program**.
4. In the Project Explorer, right-click on the tutorial-profile directory, and select **C/C++ Build Settings**.
5. Under the **ARM gcc compiler** group, select the **Profiling** subgroup, then check the **Enable Profiling** box, and click **OK** (Figure 40).
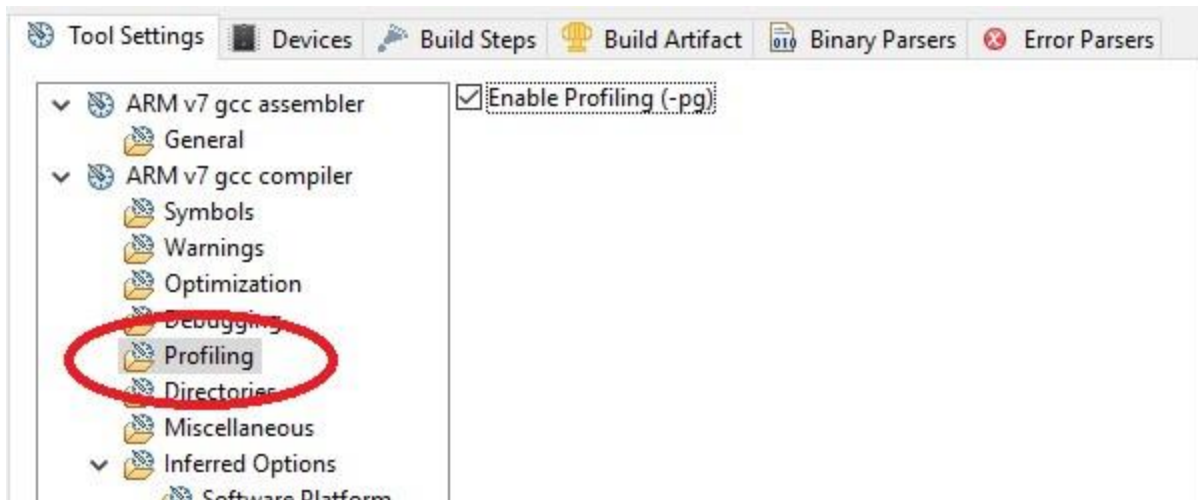
Figure 40: Compiler setting to enable profiling.

6. From the menu bar, select **Run → Run Configurations…** and double-click on **Xilinx C/C++ application** to create a new configuration.
   For SDK 2019.1, select **Applications → Advanced Options → Edit…,** as in Figure [a].
7. Click on the **Enable Profiling** check-box. Enter **100000** (100 kHz) in the Sampling Frequency field, enter **0x10000000** in the scratch memory address field, and click **Apply**, as in Figure 41 or in Figure [b].
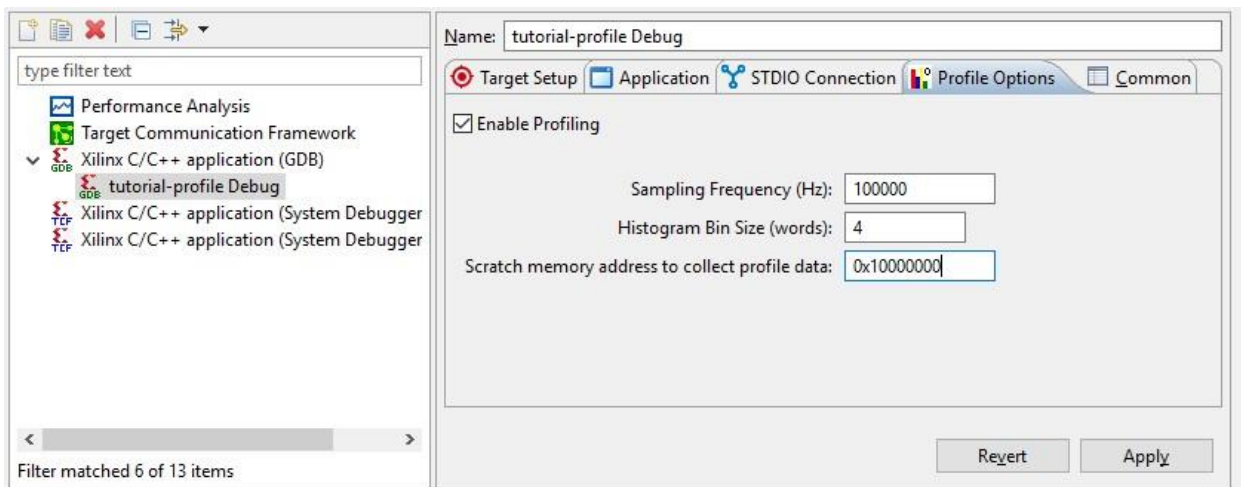

Figure 41: Profiling options
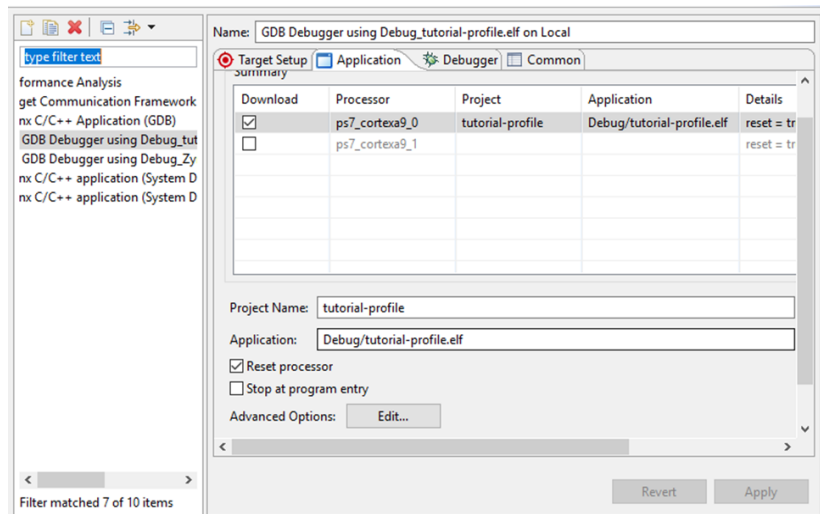
Figure **[a]** Profiling options in Vivado 2019.1
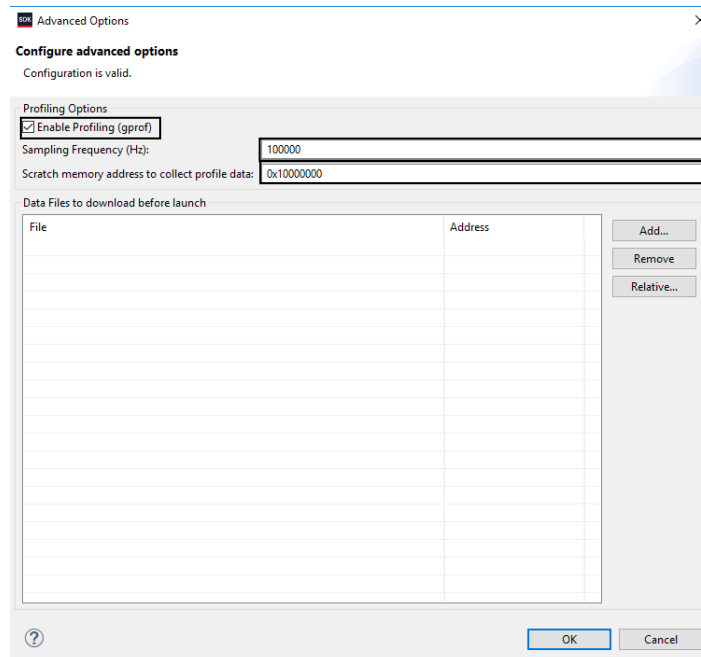
Figure **[b]** Inside the Advance Options

8. Click the **Run** button to download the application and execute it.

   The program will run, and when execution has completed, a message will be displayed indicating that the profiling results are being saved in the gmon.out file at the tutorial-profile\Debug directory.

Step 4: Invoke gprof and analyze the results

1. Expand the **Debug** folder under the tutorial-profile project in the Project Explorer view, and doubleclick on the **gmon.out** entry (Figure 42).
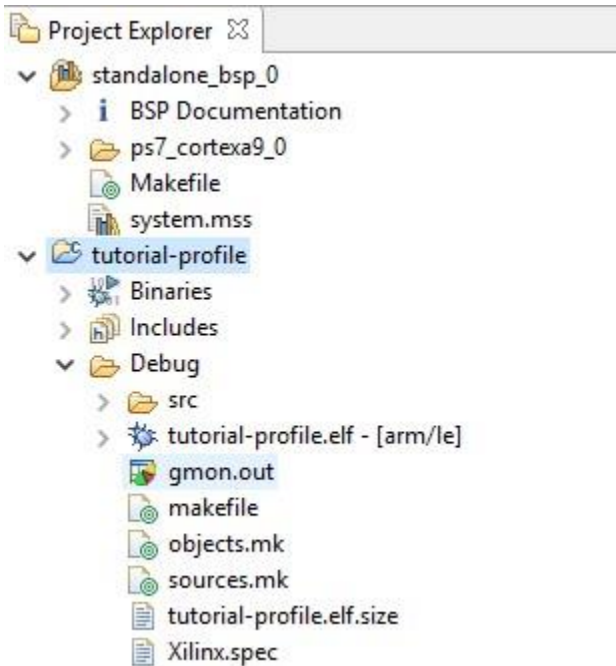
Figure 42: Invoking gprof on gmon.out.

2. The **Gmon File Viewer** dialog box will appear showing tutorial-profile.elf as the corresponding binary file. Click **OK**. The gprof viewer will load in the log panel.
3. Click on the **Sort samples per function** button ( ).
4. Click in the **%Time** column to sort in the descending order (Figure 43).

Figure 43: Sorting results.

5. Go back to the *Run Configuration*, and change the sampling frequency to **1000000** (1 MHz) and profile the application again.
6. Invoke **gprof**, select the **Sort samples per function** output, and sort the **%Time** column. Notice that the output has better resolution and reports more functions and more samples per function calls.
7. Close the SDK and Vivado programs by selecting the **File → Exit** in each program.
8. Turn OFF the power on the board.

Conclusion

This tutorial led you through the process of using SDK and gprof to profile a software application run on a custom hardware configuration.

Appendix A: ZedBoard Connection

The ZedBoard must be connected with two micro USB cables and a power supply, as shown in Figure **??** below. Additionally, the jumpers JP7, JP8, JP9, JP10 and JP11 must be connected to ground as shown in the figure.
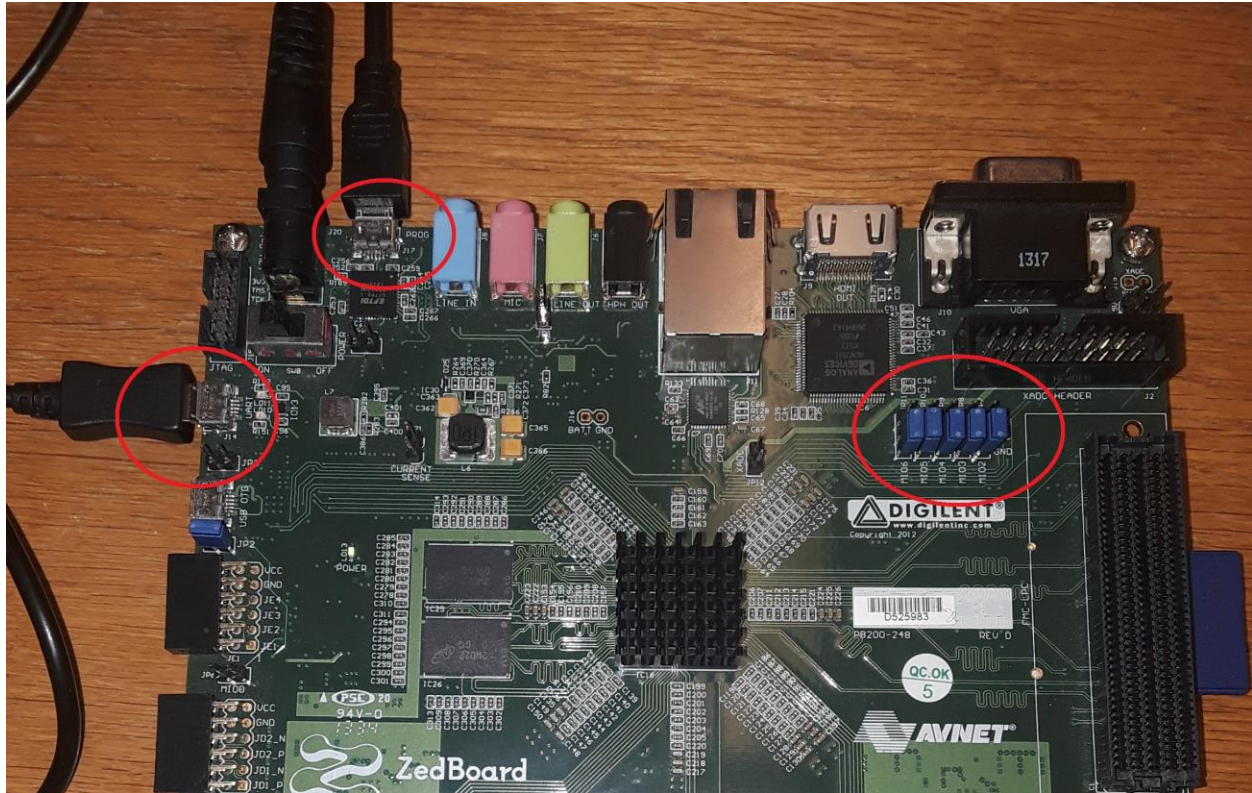

Figure 44: ZedBoard hardware configuration.