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

Shawki Areibi, Matt Saunders

July 3, 2020

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.**
- Check Appendix A for instructions to connect the ZedBoard to the Workstation.

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.

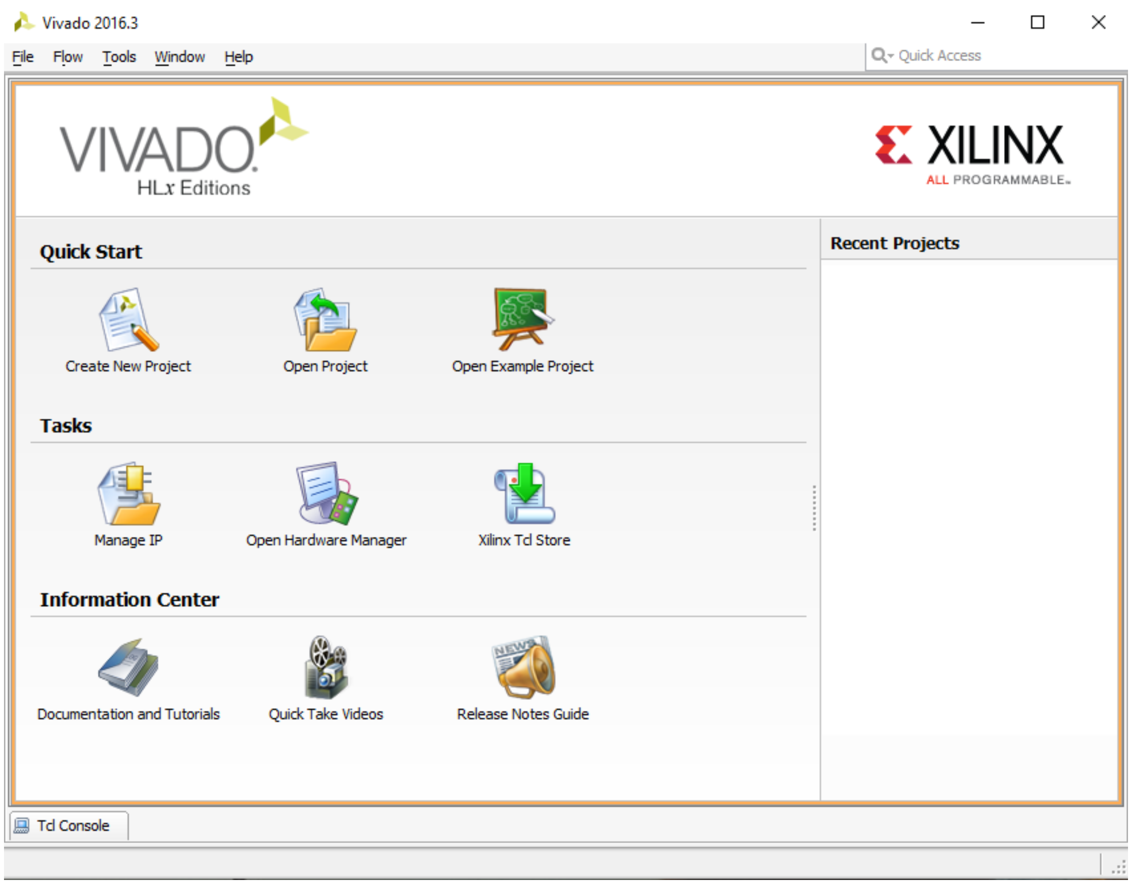
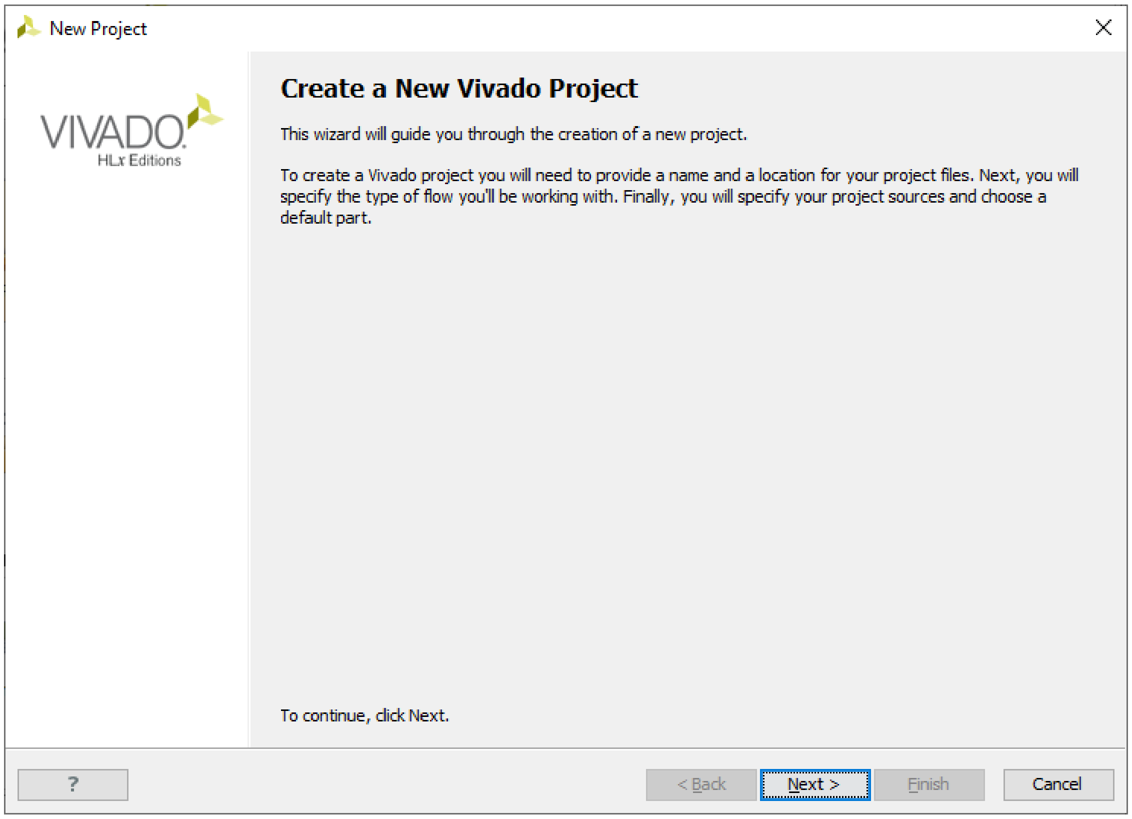


Figure 1: Vivado 2016.3 Getting Started screen.

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



Create a New Vivado Project

This wizard will guide you through the creation of a new project.

To create a Vivado project you will need to provide a name and a location for your project files. Next, you will specify the type of flow you'll be working with. Finally, you will specify your project sources and choose a default part.

To continue, 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).

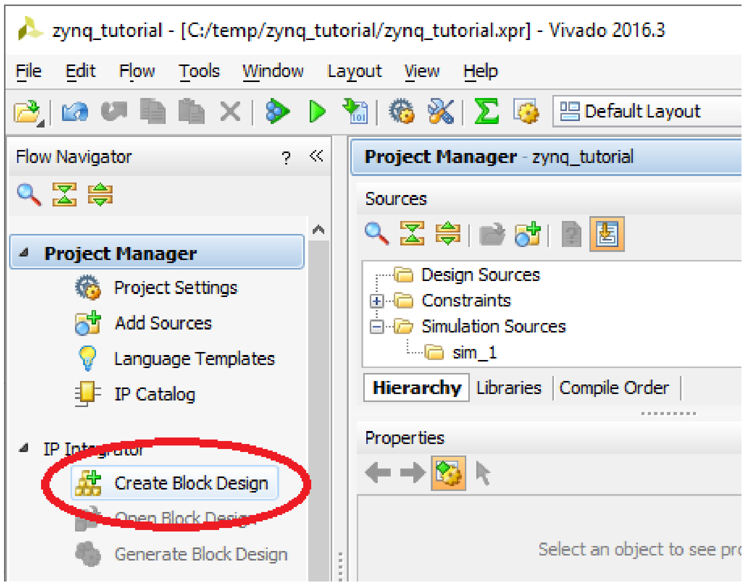
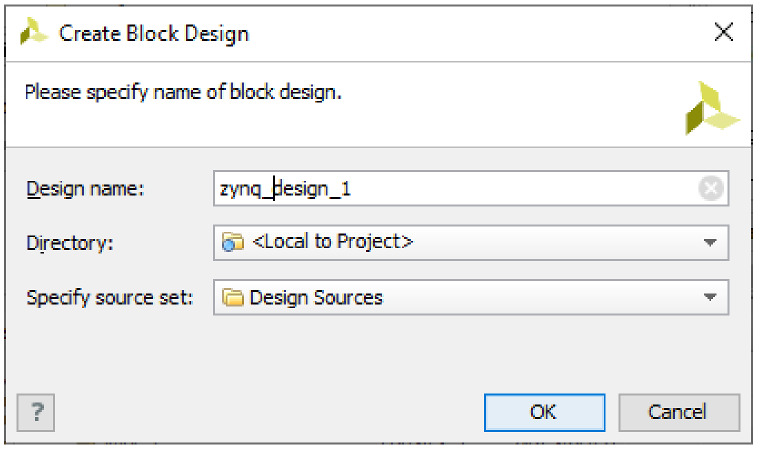


Figure 3: Create Block Design option in Flow Navigator.

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



Please specify name of block design.

Design name: zynq_design_1

Directory: <Local to Project>

Specify source set: Design Sources

Figure 4: Create Block Design dialog box.

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

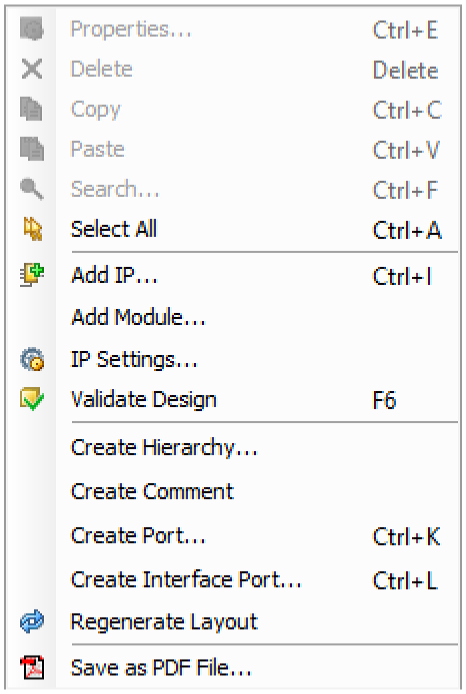


Figure 5: Add IP.

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

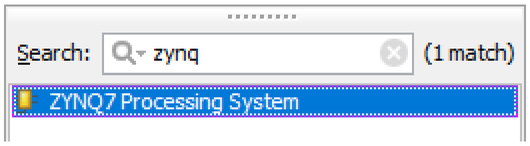


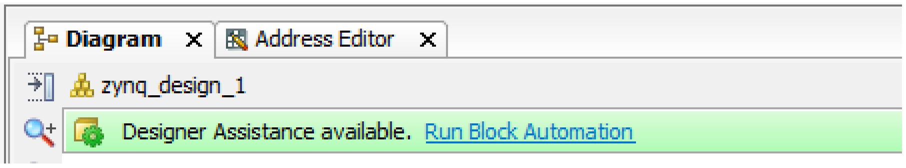
Figure 6: The IP Integrator IP Catalog.

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.

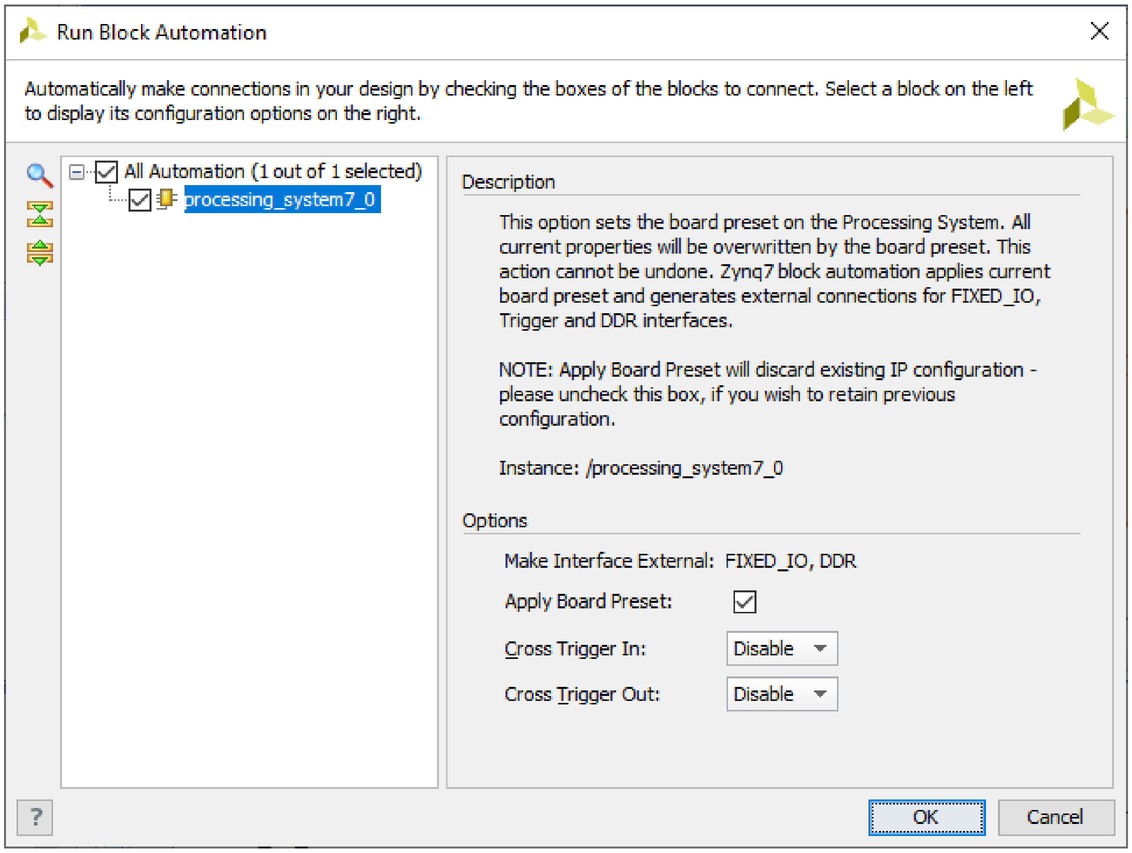
5. In the IP Integrator workspace header, click **Run Block Automation** (Figure 7).



Designer Assistance available. Run Block Automation

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**.



Automatically make connections in your design by checking the boxes of the blocks to connect. Select a block on the left to display its configuration options on the right.

This option sets the board preset on the Processing System. All current properties will be overwritten by the board preset. This action cannot be undone. Zynq7 block automation applies current board preset and generates external connections for FIXED_IO, Trigger and DDR interfaces.

NOTE: Apply Board Preset will discard existing IP configuration - please uncheck this box, if you wish to retain previous configuration.

Instance: /processing_system7_0

Make Interface External: FIXED_IO, DDR

Apply Board Preset: ✓

Cross Trigger In: Disable

Cross Trigger Out: Disable

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).

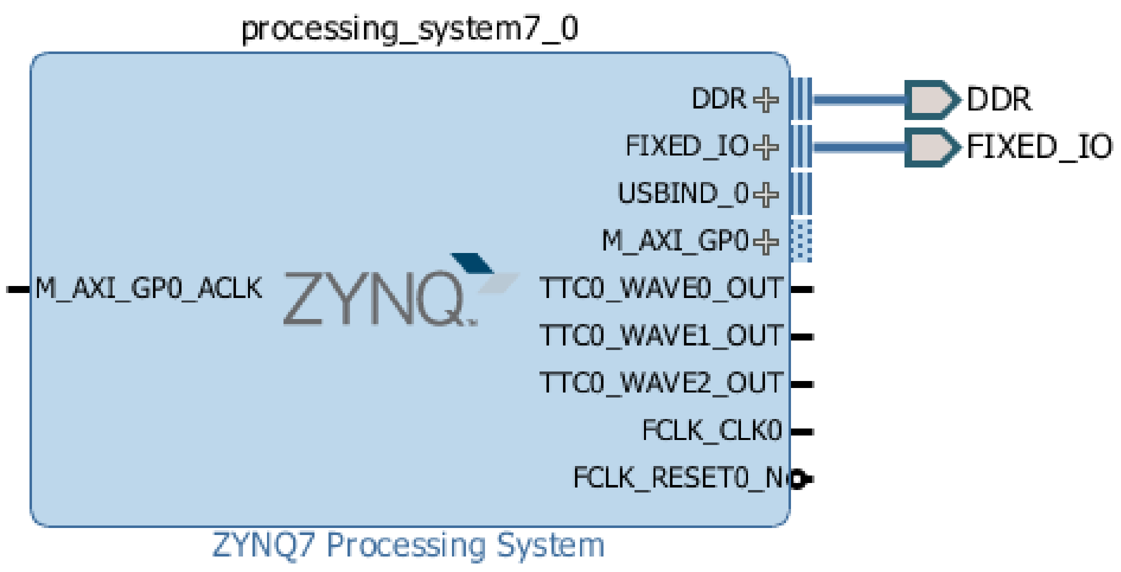


Figure 9: Zynq Processing System after running Block Automation.

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.

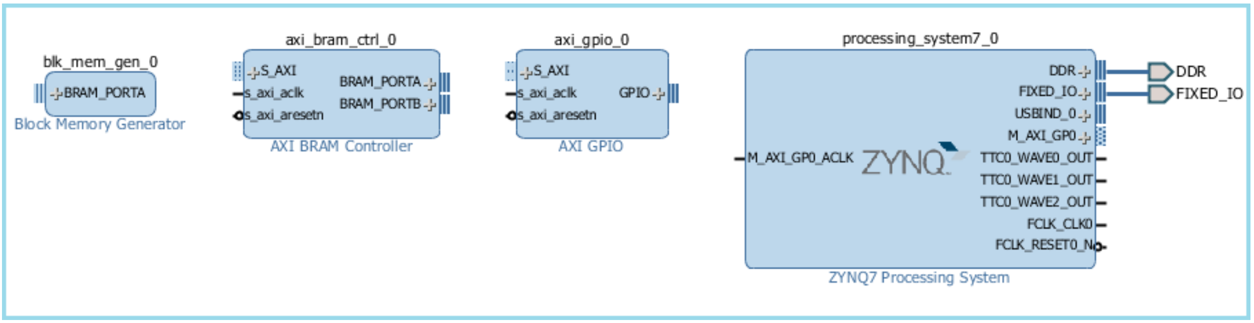
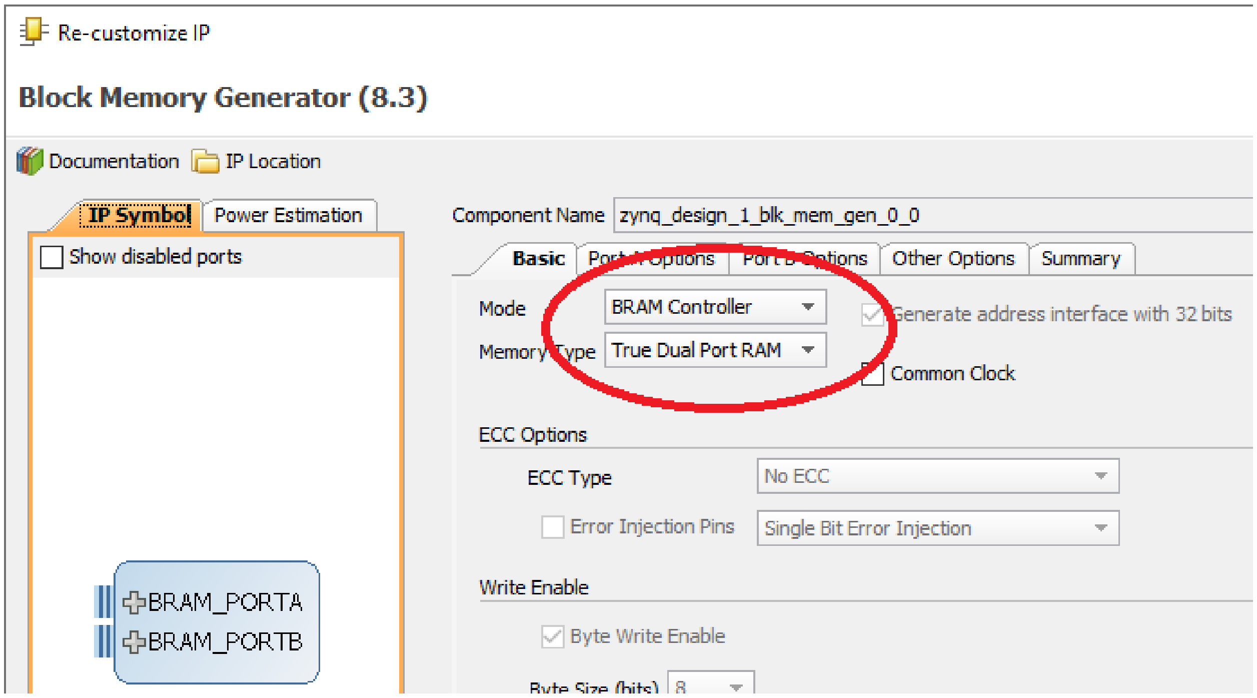


Figure 10: Block Design after instantiating IP.

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**.


Re-customize IP

Block Memory Generator (8.3)

Component Name: zynq_design_1_blk_mem_gen_0_0

Mode: BRAM Controller

Memory Type: True Dual Port RAM

[✓] Generate address interface with 32 bits

[ ] Common Clock

ECC Options

ECC Type: No ECC

[ ] Error Injection Pins: Single Bit Error Injection

Write Enable

[✓] Byte Write Enable

Byte Size (bits): 8

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).

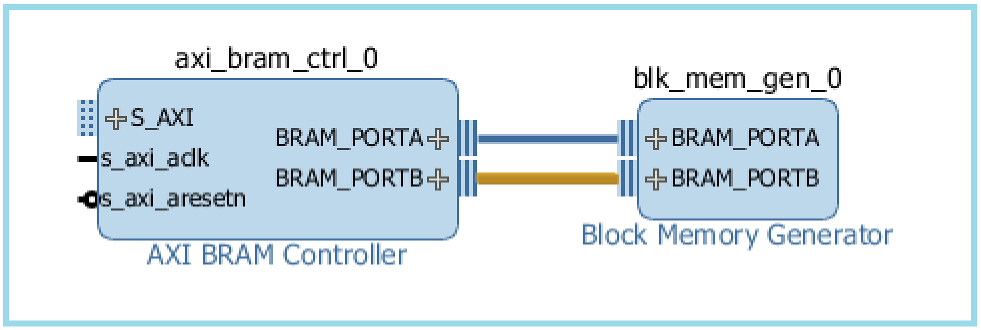


Figure 12: Connected AXI BRAM Controller and Block Memory Generator.

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.

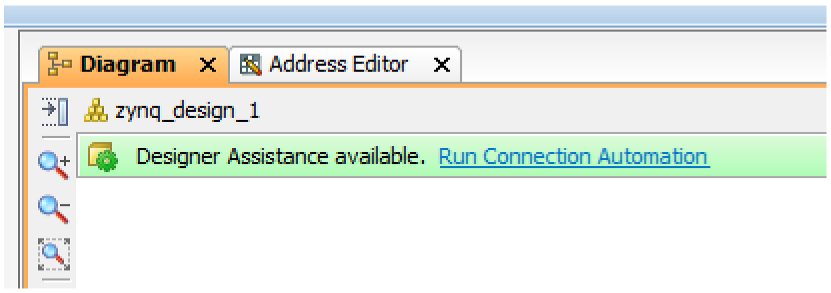
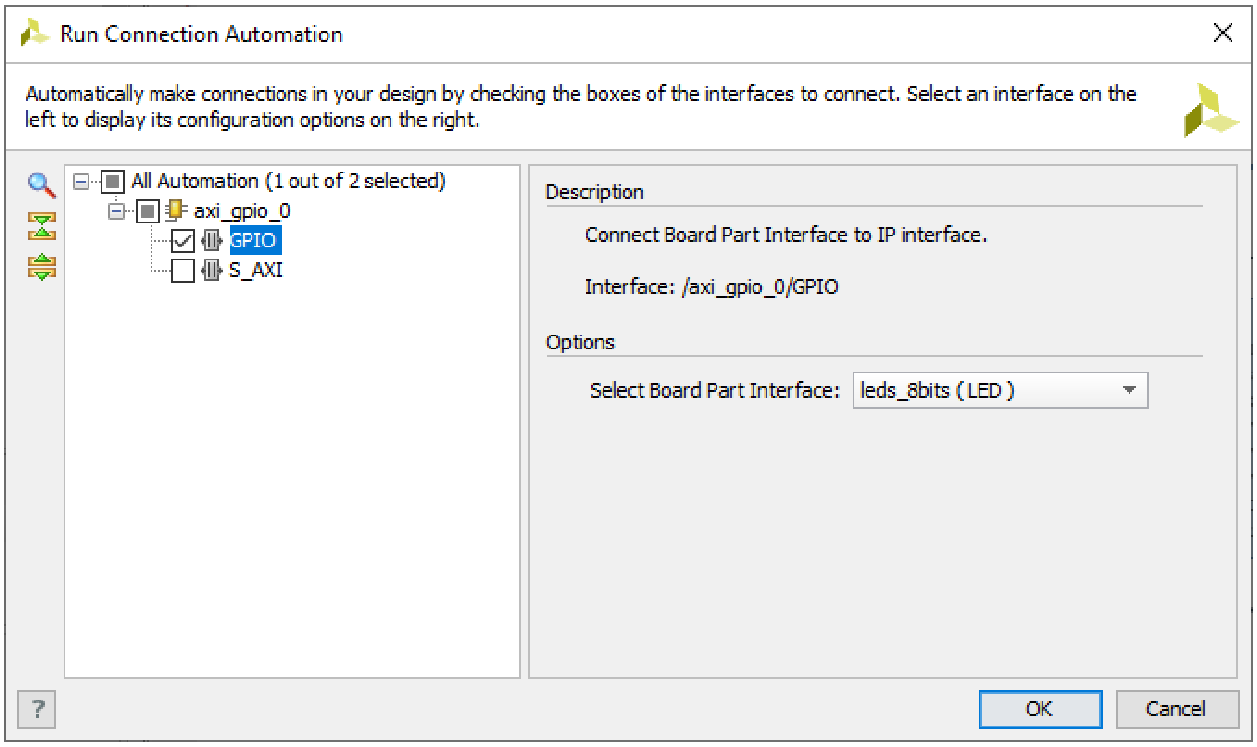


Figure 13: Connected AXI BRAM Controller and Block Memory Generator.

2. In the Run Connection Automation dialog box, under **axi_bram_ctrl_#**, select **S_AXI**. Ensure its check-box 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.


Run Connection Automation

Automatically make connections in your design by checking the boxes of the interfaces to connect. Select an interface on the left to display its configuration options on the right.

Description

Connect Board Part Interface to IP interface.

Interface: /axi_gpio_0/GPIO

Options

Select Board Part Interface: leds_8bits (LED)

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.

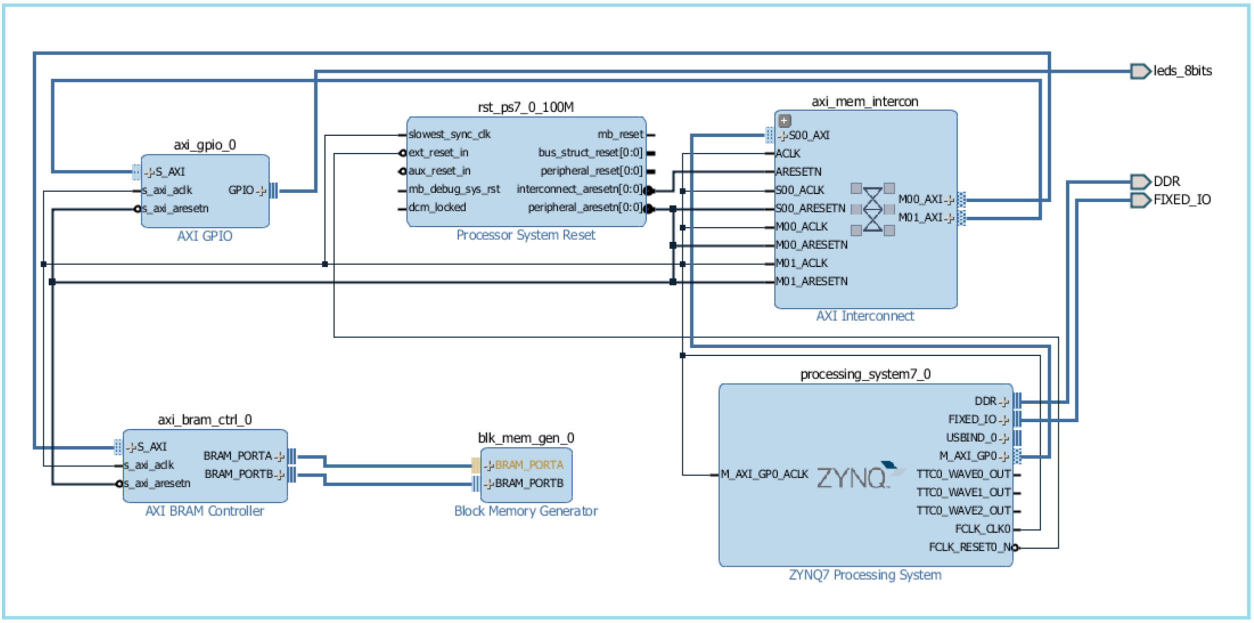
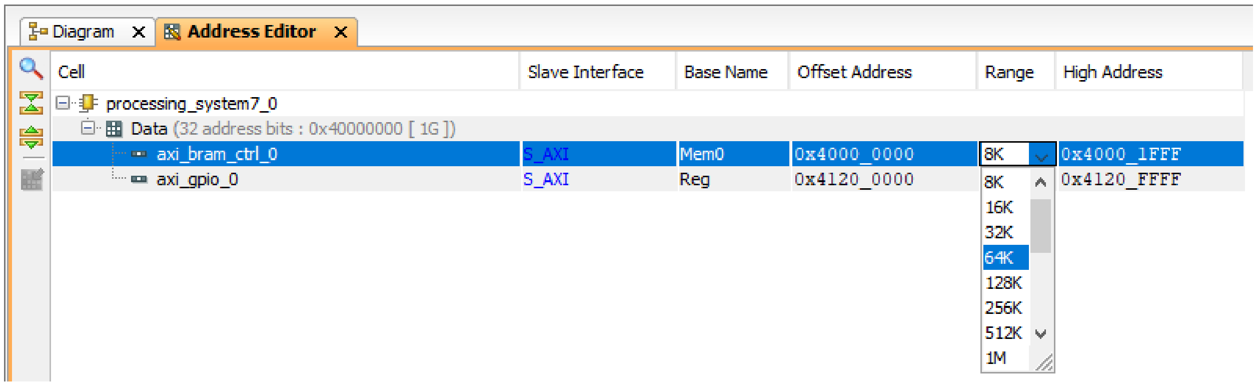


Figure 15: Zynq Processor System, connected to peripherals.

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.



Cell	Slave Interface	Base Name	Offset Address	Range	High Address
processing_system7_0					
Data (32 address bits : 0x40000000 [1G])					
axi_bram_ctrl_0	S_AXI	Mem0	0x4000_0000	8K	0x4000_1FFF
axi_gpio_0	S_AXI	Reg	0x4120_0000		0x4120_FFFF

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 **Validate Design** button (Figure 17). Alternatively, you can select Tools → Validate Design from the menu, or you can right-click in the workspace and select **Validate Design**.

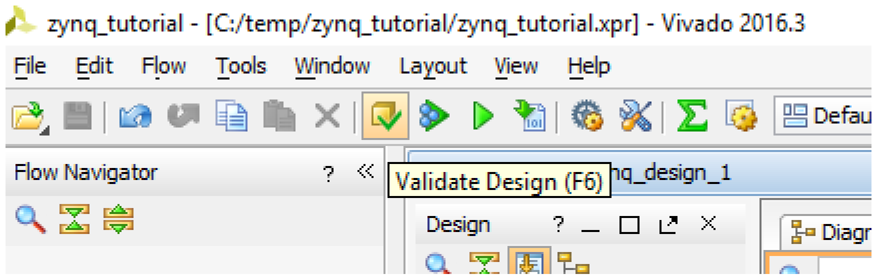


Figure 17: Validating the design.

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.

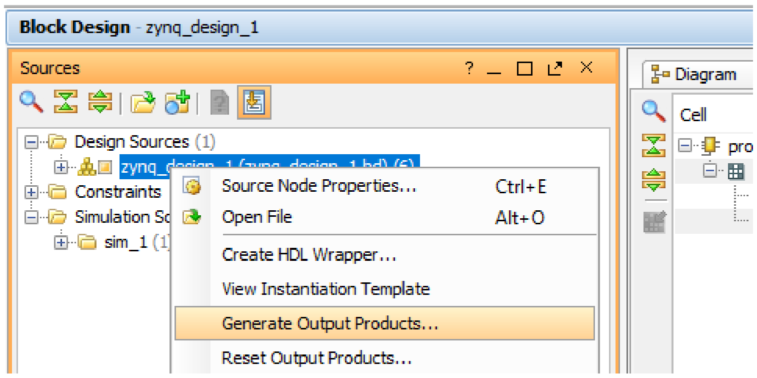

Block Design - zynq_design_1
Sources
Design Sources (1)
zynq_design_1 (zynq_design_1.bd) (6)
Constraints
Simulation Sc
sim_1 (1)
Source Node Properties... Ctrl+E
Open File Alt+O
Create HDL Wrapper...
View Instantiation Template
Generate Output Products...
Reset Output Products...
Diagram
Cell
pro

Figure 18: Generating output products.

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

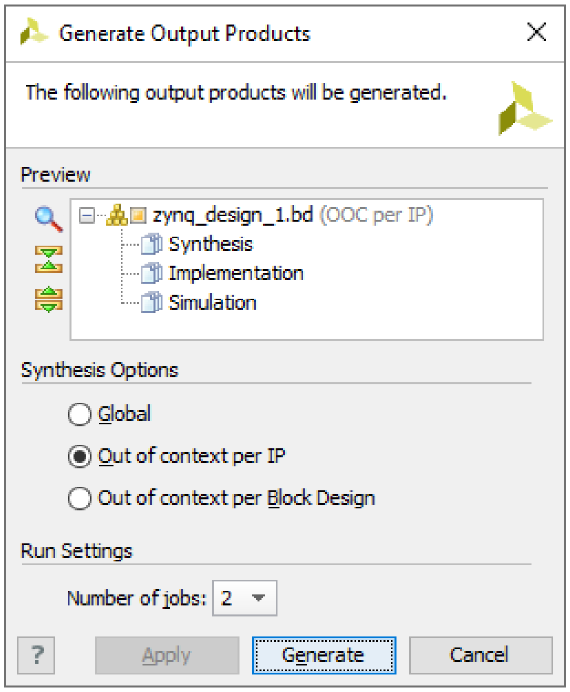


Figure 19: Generating output products.

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 auto-update** and click **OK**.

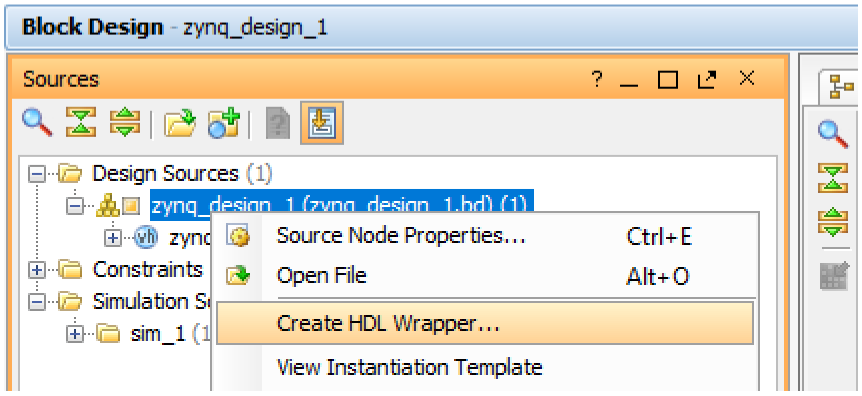
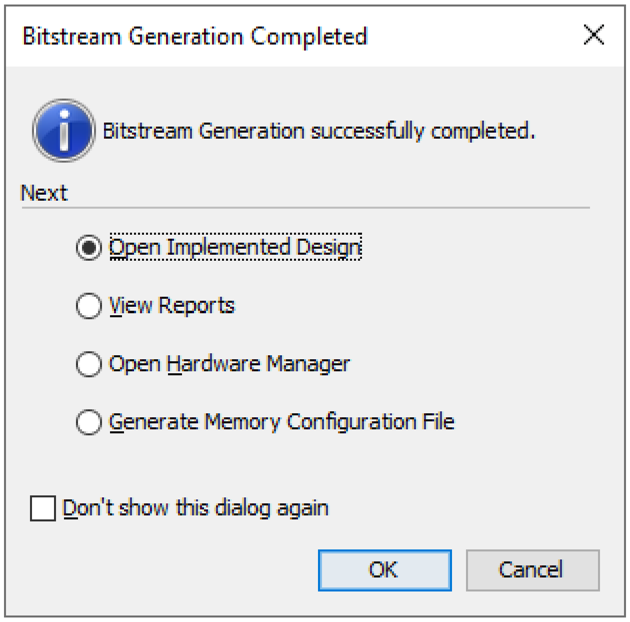

Block Design - zynq_design_1
Sources
Design Sources (1)
zynq_design_1 (zynq_design_1.bd) (1)
zynq
Constraints
Simulation S
sim_1 (1
Source Node Properties... Ctrl+E
Open File Alt+O
Create HDL Wrapper...
View Instantiation Template

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**.



Bitstream Generation Completed

Bitstream Generation successfully completed.

Next

● Open Implemented Design
○ View Reports
○ Open Hardware Manager
○ Generate Memory Configuration File

☐ Don't show this dialog again

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. Refer to Appendix A for necessary cables and jumper connection.

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).

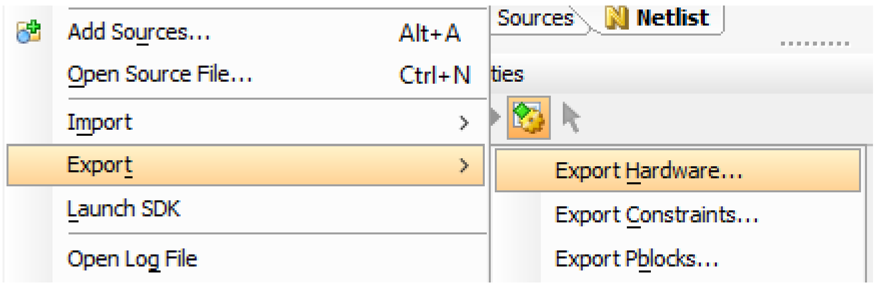
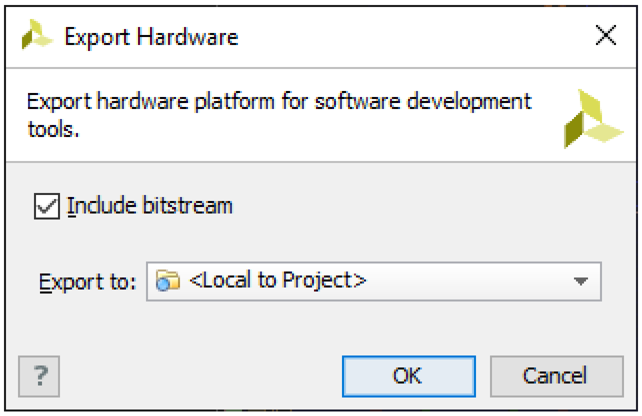


Figure 22: Export Hardware menu option.

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


Export Hardware

Export hardware platform for software development tools.

☑ Include bitstream

Export to: <Local to Project>

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 section of the tutorial you will build 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).

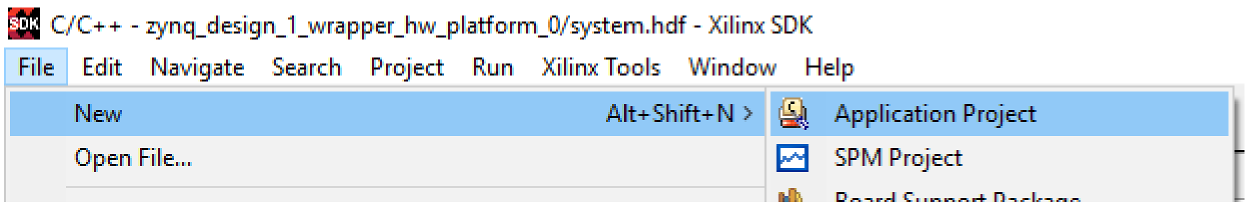
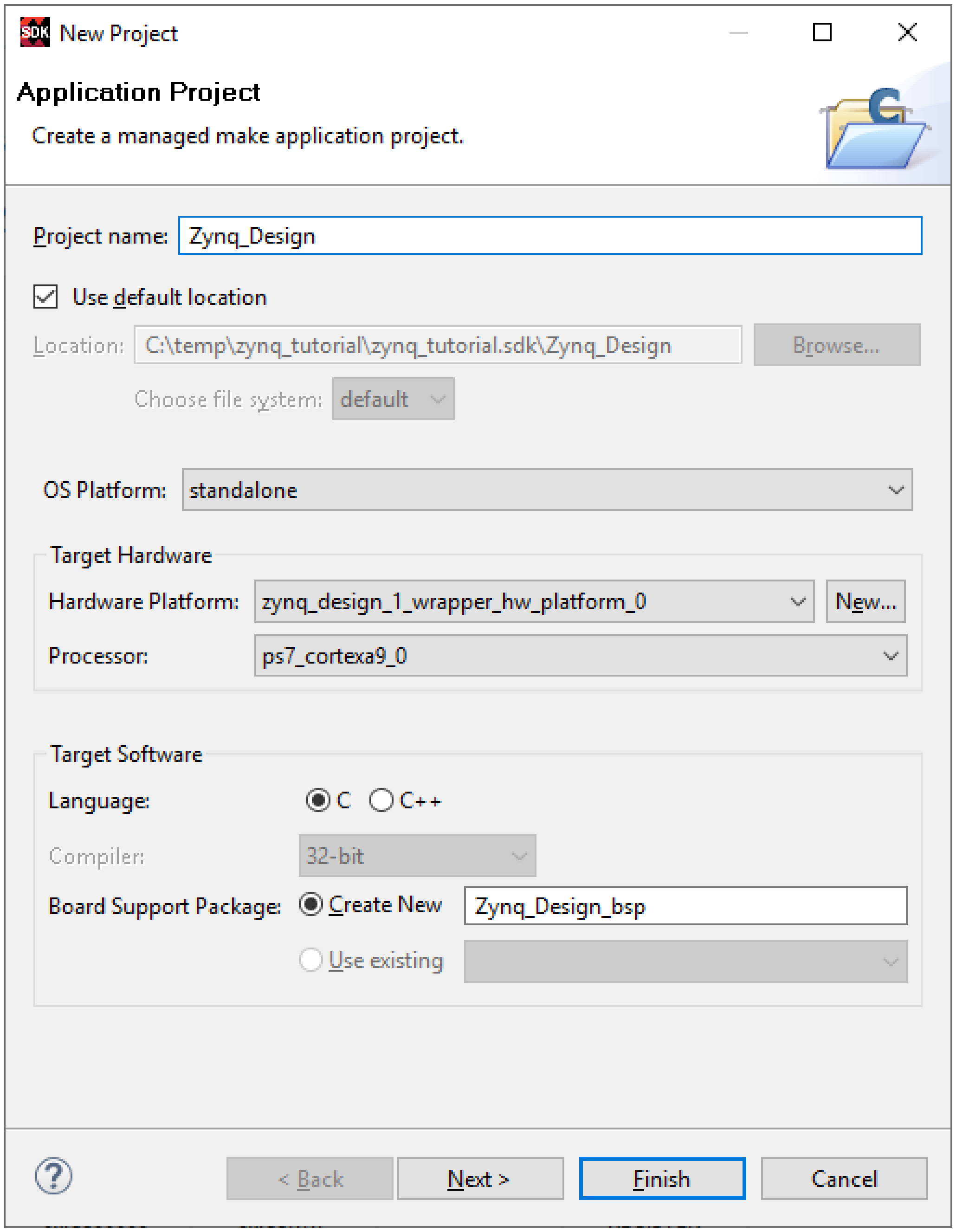


Figure 24: New Application Project.

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



New Project

Application Project

Create a managed make application project.

Project name: Zynq_Design

[✓] Use default location

Location: C:\temp\zynq_tutorial\zynq_tutorial.sdk\Zynq_Design

Choose file system: default

OS Platform: standalone

Target Hardware

Hardware Platform: zynq_design_1_wrapper_hw_platform_0

Processor: ps7_cortexa9_0

Target Software

Language: [✓ C] C++

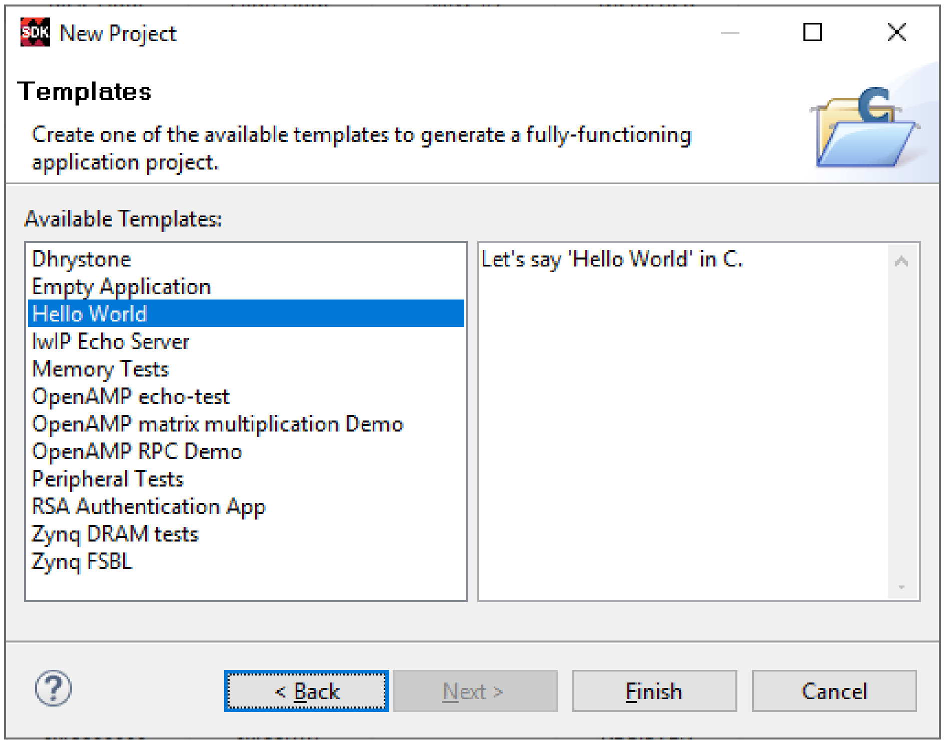
Compiler: 32-bit

Board Support Package: [✓ Create New] Zynq_Design_bsp

Use existing

Figure 25: Setting up the new Application Project.

5. From the available templates, select **Hello World** (Figure 26), and click **Finish**.


New Project

Templates

Create one of the available templates to generate a fully-functioning application project.

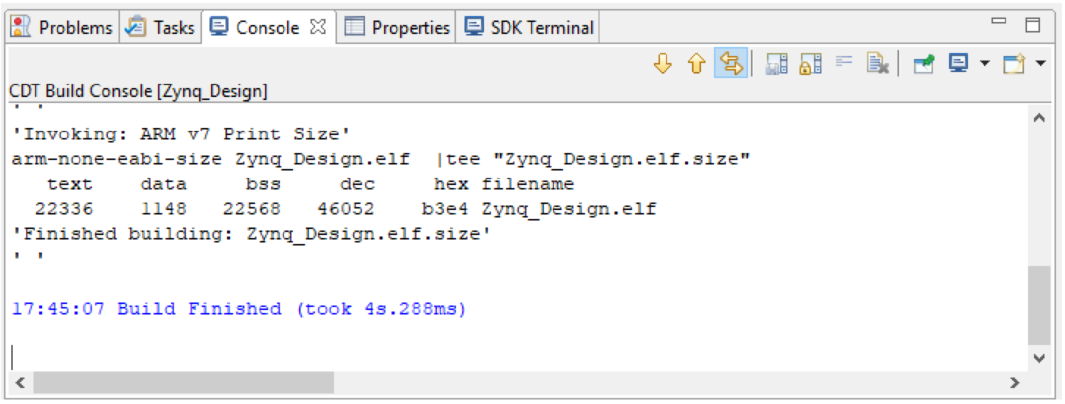
Available Templates:

- Dhrystone
- Empty Application
- Hello World
- lwIP Echo Server
- Memory Tests
- OpenAMP echo-test
- OpenAMP matrix multiplication Demo
- OpenAMP RPC Demo
- Peripheral Tests
- RSA Authentication App
- Zynq DRAM tests
- Zynq FSBL

Let's say 'Hello World' in C.

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.



CDT Build Console [Zynq_Design]

```
' '
'Invoking: ARM v7 Print Size'
arm-none-eabi-size Zynq_Design.elf  |tee "Zynq_Design.elf.size"
   text    data     bss     dec     hex filename
  22336    1148   22568   46052    b3e4 Zynq_Design.elf
'Finished building: Zynq_Design.elf.size'
' '

17:45:07 Build Finished (took 4s.288ms)
```

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).

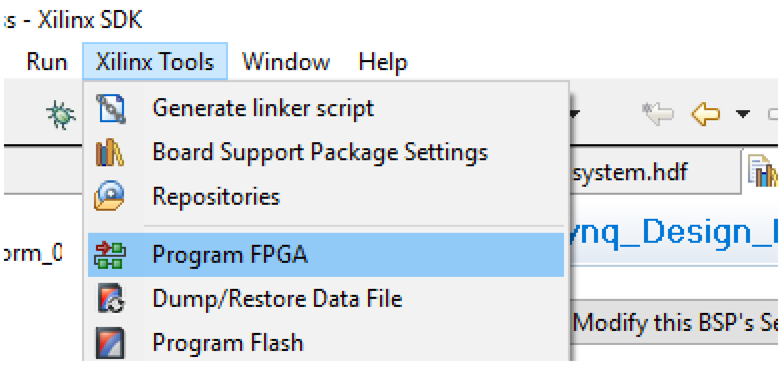


Figure 28: Program FPGA menu option.

2. Ensure that the path to the bitstream (created in Part 1) is correct, then click **Program**.
3. NOTE: The bit file is usually found in:
   - A directory with runs extension i.e., project_name.runs in impl_1 subdirectory, or
   - A directory with sdk extension i.e., project_name.sdk in wrapper_hw_platform
4. The DONE LED on the board turns blue if the programming is successful.

5. In the Project Explorer pane, select and right-click the **Zynq_Design** application.
6. Select **Debug As** → **Debug Configurations** (Figure 29).

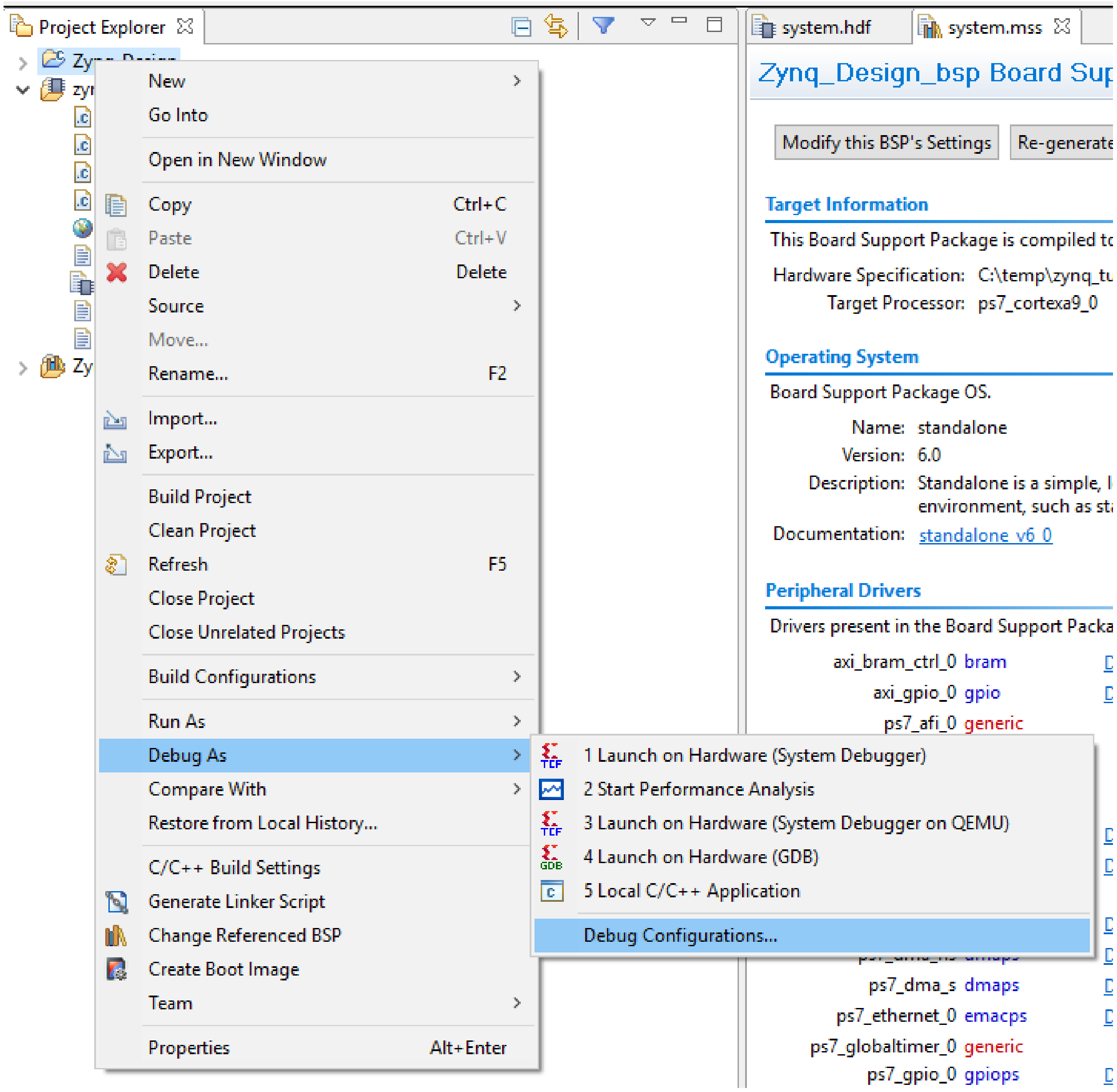
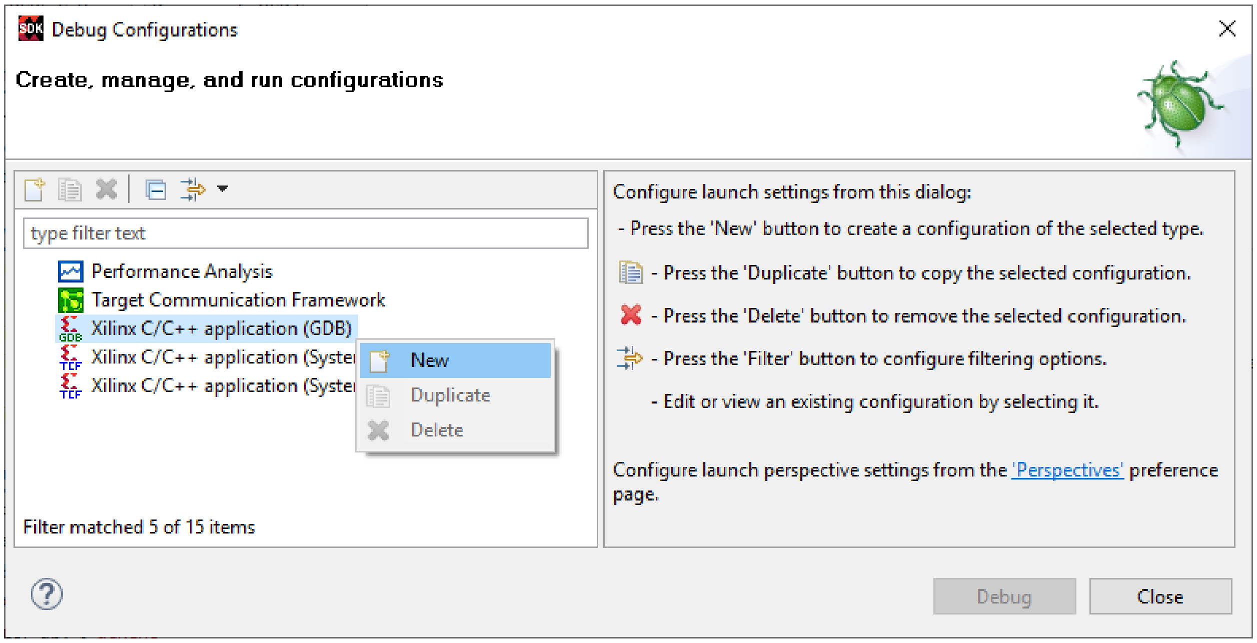


Figure 29: Debug Configurations menu option.

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


Debug Configurations

Create, manage, and run configurations

type filter text
Performance Analysis
Target Communication Framework
Xilinx C/C++ application (GDB)
Xilinx C/C++ application (Syste
Xilinx C/C++ application (Syste
New
Duplicate
Delete
Filter matched 5 of 15 items
Debug
Close

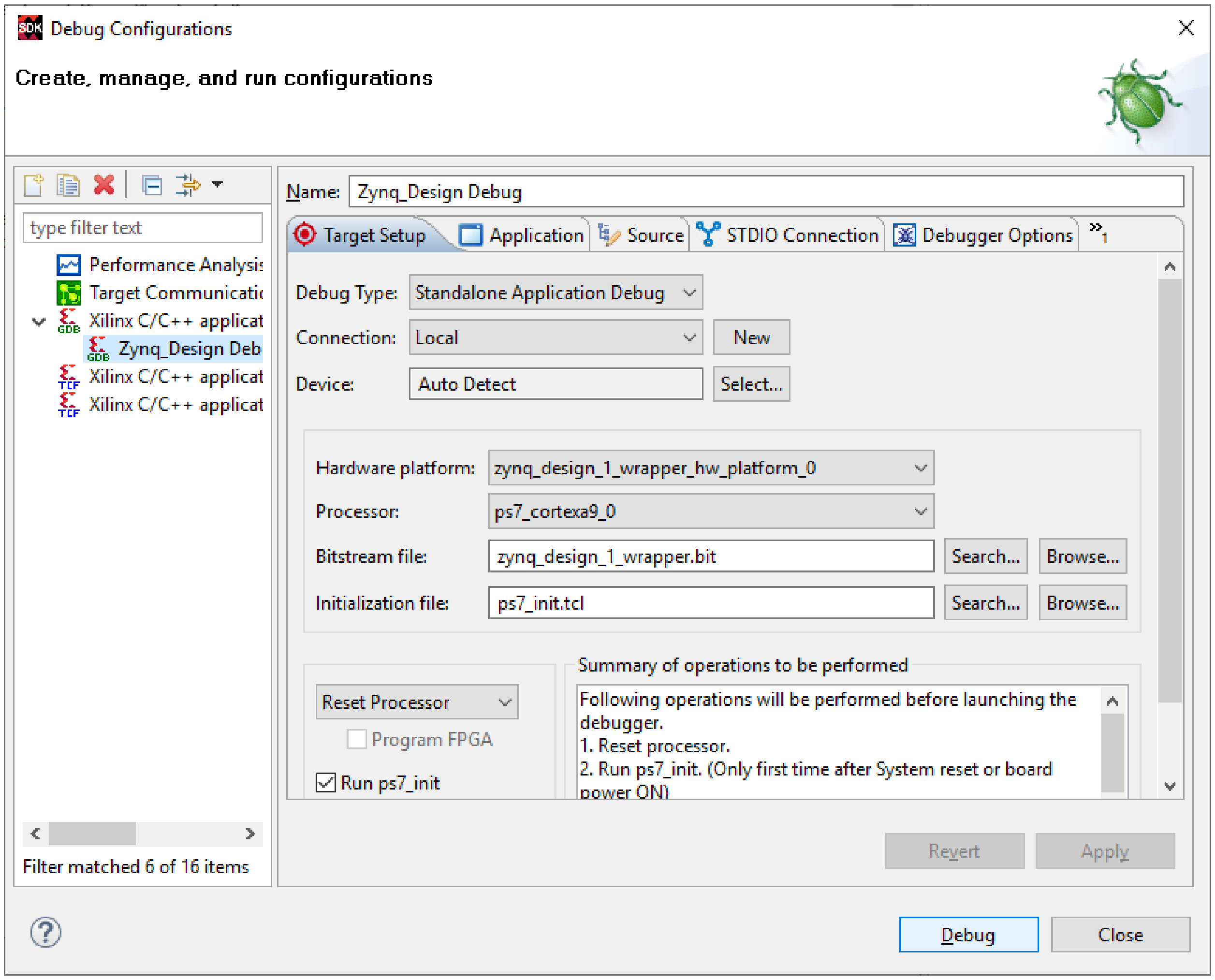
Configure launch settings from this dialog:

- Press the 'New' button to create a configuration of the selected type.
- Press the 'Duplicate' button to copy the selected configuration.
- Press the 'Delete' button to remove the selected configuration.
- Press the 'Filter' button to configure filtering options.
- Edit or view an existing configuration by selecting it.

Configure launch perspective settings from the 'Perspectives' preference page.

Figure 30: Debug Configurations dialog box.

8. In the Debug Configurations dialog box, leave the settings as default, and select **Debug**.


Debug Configurations

Create, manage, and run configurations

Name: Zynq_Design Debug

Debug Type: Standalone Application Debug

Connection: Local

Device: Auto Detect

Hardware platform: zynq_design_1_wrapper_hw_platform_0

Processor: ps7_cortexa9_0

Bitstream file: zynq_design_1_wrapper.bit

Initialization file: ps7_init.tcl

Reset Processor

☐ Program FPGA

☑ Run ps7_init

Summary of operations to be performed

Following operations will be performed before launching the debugger.
1. Reset processor.
2. Run ps7_init. (Only first time after System reset or board power ON)

Figure 31: Debug Configurations dialog box.

9. NOTE: If the **Bitsream file:** and/or the **Initialization file:** are not populated (i.e. grey) press search and populate them with the required values as seen in the figure above.
10. NOTE: Check if the Project Name and Application (elf file) in "Application" part are populated. If not search and populate them.
11. You will be asked to confirm a perspective switch— click **Yes**. The Debug perspective will open.

12. 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.

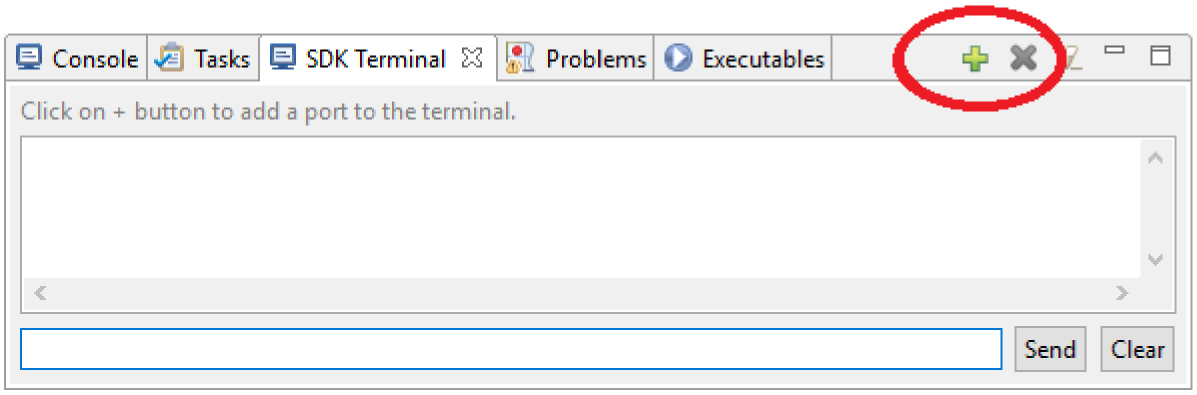
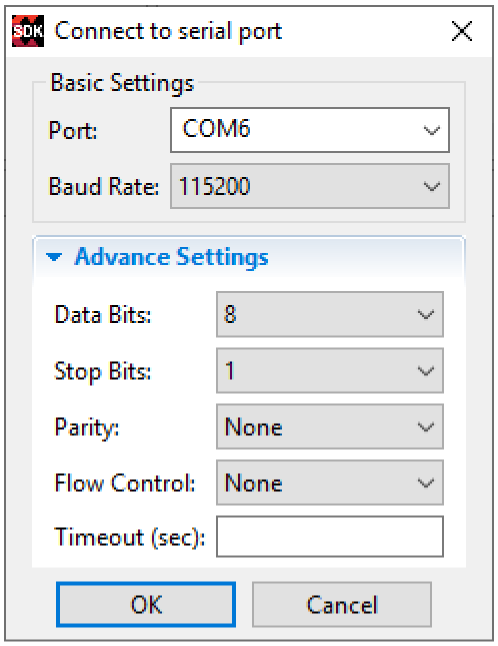


Figure 32: Creating a new serial connection.

13. 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**.



Connect to serial port

Basic Settings

Port: COM6

Baud Rate: 115200

Advance Settings

Data Bits: 8

Stop Bits: 1

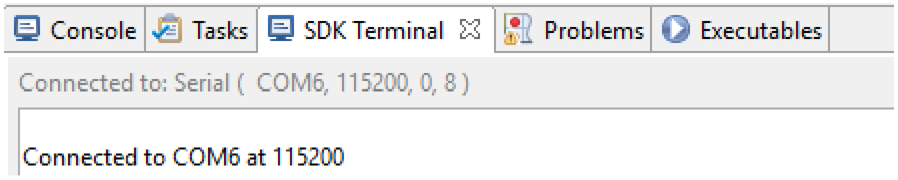
Parity: None

Flow Control: None

Timeout (sec):

Figure 33: Setting up the serial port.

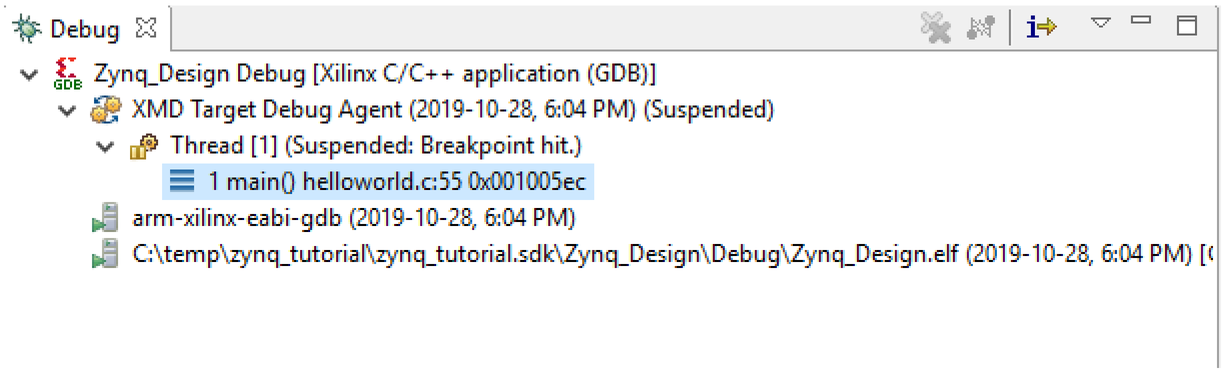
14. Verify the Terminal connection by checking the status at the top of the tab (Figure 34).



```
Connected to: Serial (  COM6, 115200, 0, 8 )

Connected to COM6 at 115200
```

Figure 34: Terminal connection verification.

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



```
Zynq_Design Debug [Xilinx C/C++ application (GDB)]
  XMD Target Debug Agent (2019-10-28, 6:04 PM) (Suspended)
    Thread [1] (Suspended: Breakpoint hit.)
      1 main() helloworld.c:55 0x001005ec
  arm-xilinx-eabi-gdb (2019-10-28, 6:04 PM)
  C:\temp\zynq_tutorial\zynq_tutorial.sdk\Zynq_Design\Debug\Zynq_Design.elf (2019-10-28, 6:04 PM) [
```

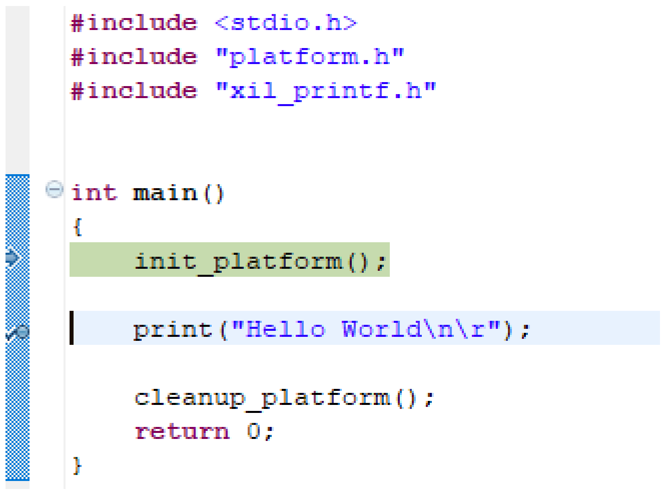
Figure 35: Terminal connection verification.

16. 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. If the line numbers are not displayed point the mouse on the far left side of the source code and right click to enable displaying line numbers.
2. Select **Navigate** → **Go to Line**. To go to line 57, type **57**.
3. Double-click to the left of Line 57, which adds a break point on that line. (Figure 36)



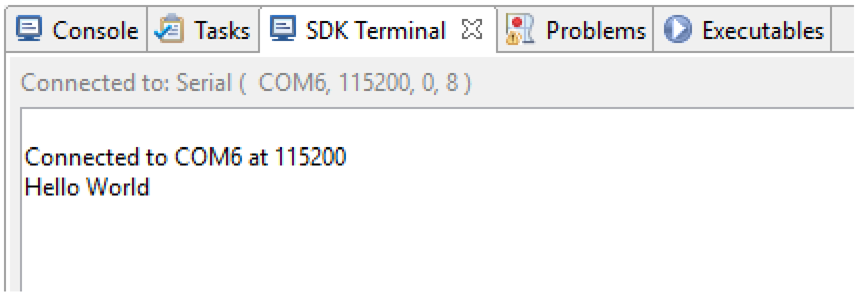
```
#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"


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)



```
Connected to: Serial ( COM6, 115200, 0, 8 )

Connected to COM6 at 115200
Hello World
```

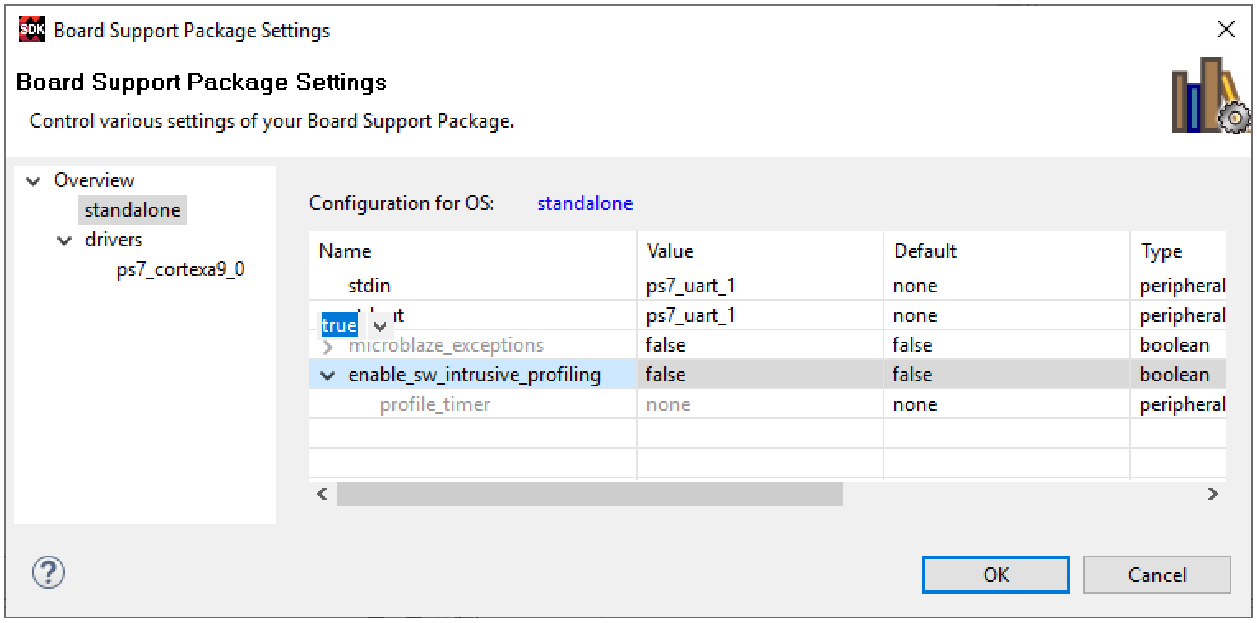
Figure 37: Terminal output.

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 shown in Figure 38.

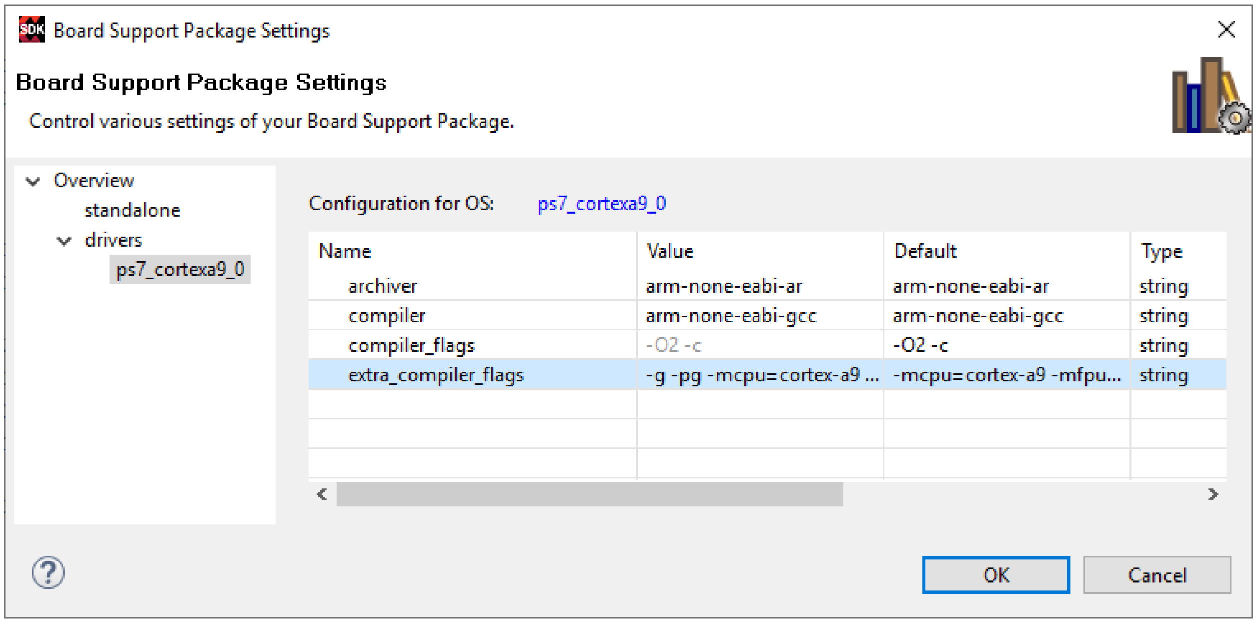

Board Support Package Settings
Board Support Package Settings
Control various settings of your Board Support Package.
Overview
standalone
drivers
ps7_cortexa9_0
OK
Cancel

Configuration for OS: standalone

Name	Value	Default	Type
stdin	ps7_uart_1	none	peripheral
true	ps7_uart_1	none	peripheral
microblaze_exceptions	false	false	boolean
enable_sw_intrusive_profiling	false	false	boolean
profile_timer	none	none	peripheral

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`
7. Press OK.


Board Support Package Settings
Board Support Package Settings
Control various settings of your Board Support Package.
Overview
standalone
drivers
ps7_cortexa9_0
OK
Cancel

Configuration for OS: ps7_cortexa9_0

Name	Value	Default	Type
archiver	arm-none-eabi-ar	arm-none-eabi-ar	string
compiler	arm-none-eabi-gcc	arm-none-eabi-gcc	string
compiler_flags	-O2 -c	-O2 -c	string
extra_compiler_flags	-g -pg -mcpu=cortex-a9 ...	-mcpu=cortex-a9 -mfpu...	string

Figure 39: Adding Compiler Flags for Profiler.

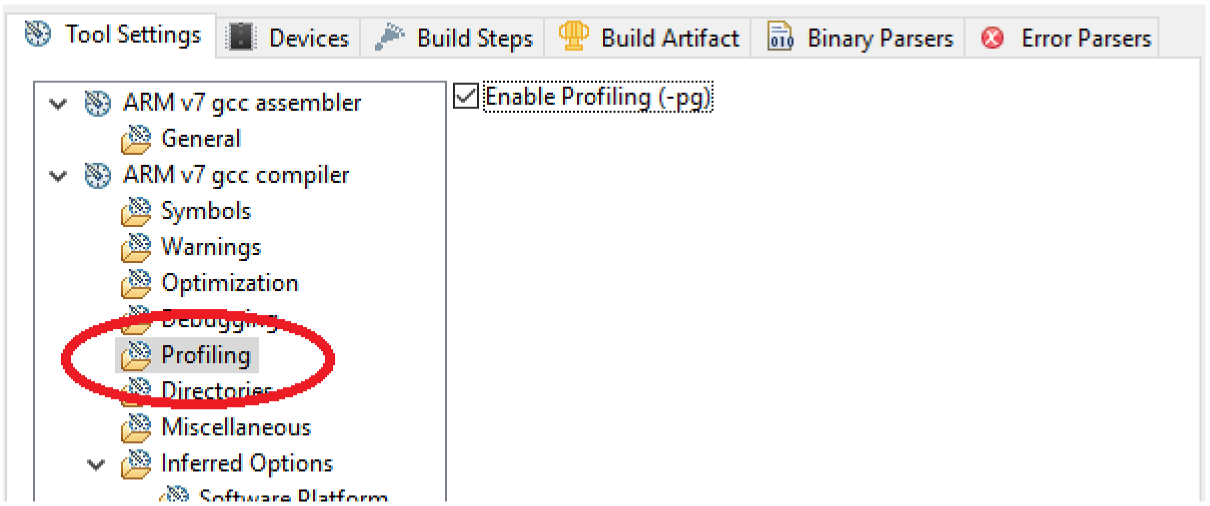
Step 2: Create the application

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 (make sure you save it as a C program and not a text file!!).
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**.

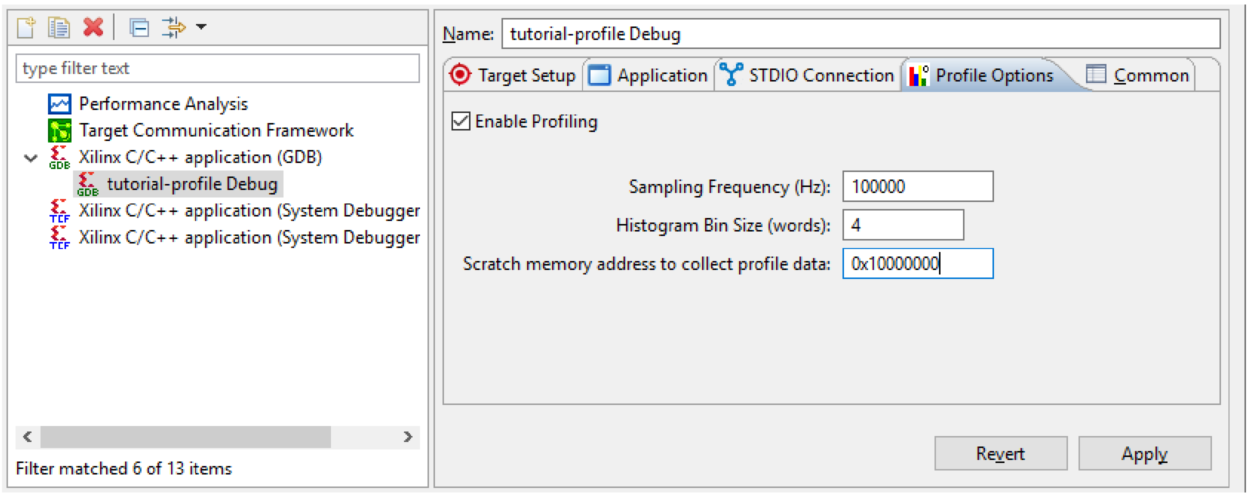
5. Under the **ARM gcc compiler** group, select the **Profiling** subgroup, then check the **Enable Profiling** box, and click **OK** (Figure 40).



☑ Enable Profiling (-pg)

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.
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.


type filter text
Performance Analysis
Target Communication Framework
Xilinx C/C++ application (GDB)
tutorial-profile Debug
Xilinx C/C++ application (System Debugger
Xilinx C/C++ application (System Debugger
Filter matched 6 of 13 items
Target Setup
Application
STDIO Connection
Profile Options
Common
Revert
Apply

Name: tutorial-profile Debug

☑ Enable Profiling

Sampling Frequency (Hz): 100000
Histogram Bin Size (words): 4
Scratch memory address to collect profile data: 0x10000000

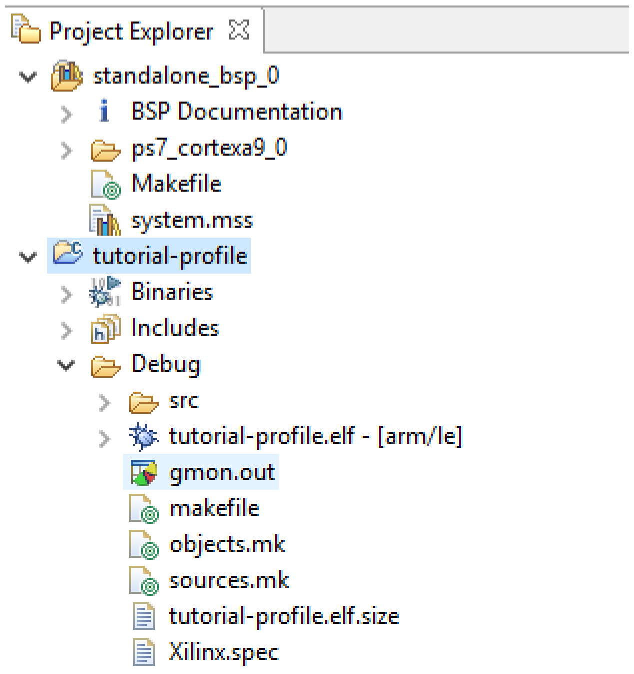
Figure 41: Profiling options.

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.

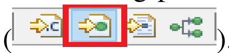
Step 4: Invoke gprof and analyze the results

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

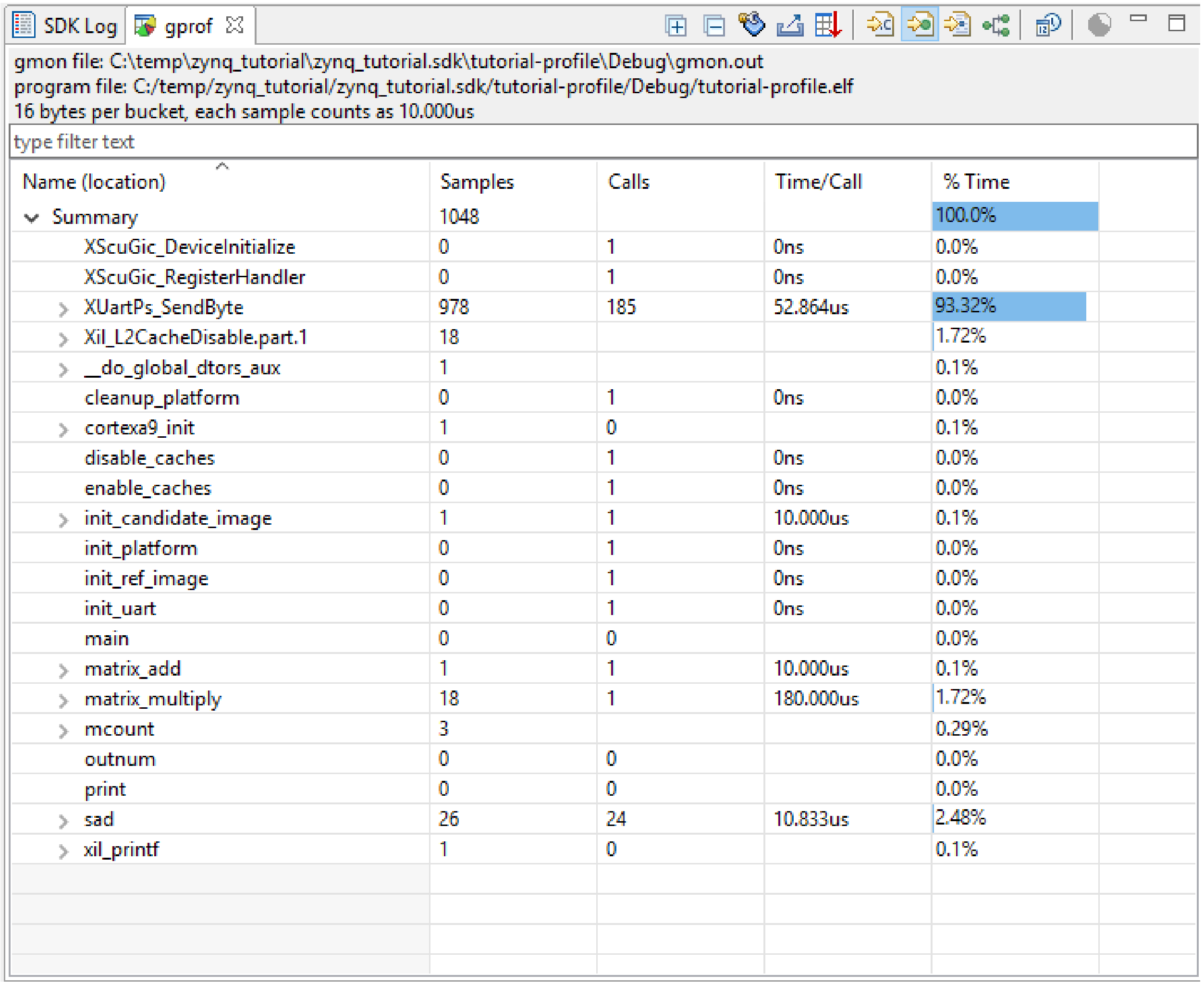


```
Project Explorer
standalone_bsp_0
    BSP Documentation
    ps7_cortexa9_0
    Makefile
    system.mss
tutorial-profile
    Binaries
    Includes
    Debug
        src
        tutorial-profile.elf - [arm/le]
        gmon.out
        makefile
        objects.mk
        sources.mk
        tutorial-profile.elf.size
        Xilinx.spec
```

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).


SDK Log | gprof

gmon file: C:\temp\zynq_tutorial\zynq_tutorial.sdk\tutorial-profile\Debug\gmon.out
program file: C:/temp/zynq_tutorial/zynq_tutorial.sdk/tutorial-profile/Debug/tutorial-profile.elf
16 bytes per bucket, each sample counts as 10.000us

type filter text

Name (location)	Samples	Calls	Time/Call	% Time
Summary	1048			100.0%
XScuGic_DeviceInitialize	0	1	0ns	0.0%
XScuGic_RegisterHandler	0	1	0ns	0.0%
XUartPs_SendByte	978	185	52.864us	93.32%
Xil_L2CacheDisable.part.1	18			1.72%
__do_global_dtors_aux	1			0.1%
cleanup_platform	0	1	0ns	0.0%
cortexa9_init	1	0		0.1%
disable_caches	0	1	0ns	0.0%
enable_caches	0	1	0ns	0.0%
init_candidate_image	1	1	10.000us	0.1%
init_platform	0	1	0ns	0.0%
init_ref_image	0	1	0ns	0.0%
init_uart	0	1	0ns	0.0%
main	0	0		0.0%
matrix_add	1	1	10.000us	0.1%
matrix_multiply	18	1	180.000us	1.72%
mcount	3			0.29%
outnum	0	0		0.0%
print	0	0		0.0%
sad	26	24	10.833us	2.48%
xil_printf	1	0		0.1%

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 44 below. Additionally, the jumpers JP7, JP8, JP9, JP10 and JP11 must be connected to ground as shown in the figure.

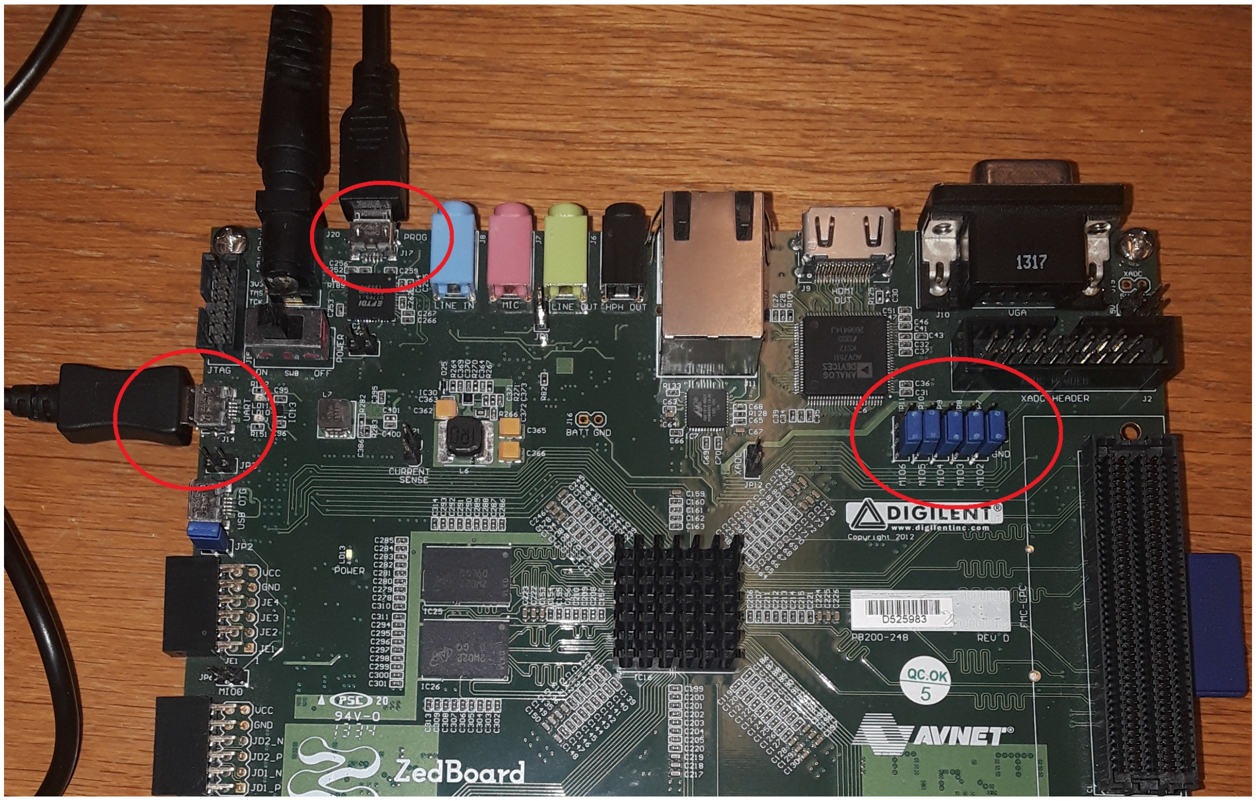


Figure 44: ZedBoard hardware configuration.