

ZedBoard: Zynq-7000

AP SoC Concepts,

Tools. and Techniques

A Hands-On Guide to

Effective Embedded System

Design

ZedBoard (v14.1)



Notice of Disclaimer

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 the Limited Warranties which can be viewed at <http://www.xilinx.com/warranty.htm>; 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 Critical Applications: <http://www.xilinx.com/warranty.htm#critapps>.



© Copyright 2012 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. All other trademarks are the property of their respective owners.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
8/20/2012	14.1	First version

Table of Contents

Contents

Chapter 1	Introduction.....	4
1.1	About this Guide	4
1.1.1	Take a Test Drive!	4
1.1.2	Additional Documentation.....	5
1.1.3	Training Labs	5
1.2	How Zynq AP SoC and Xilinx software Simplify Embedded Processor Design	5
1.3	What You Need to Set Up Before Starting	7
1.3.1	Software Installation Requirements:.....	7
1.3.2	Hardware Requirements for this Guide	8
Chapter 2	Embedded System Design Using the Zynq Processing System	9
2.1	Embedded System Construction	11
2.1.1	Take a Test Drive! Creating a New Embedded Project With a Zynq Processing System	11
2.1.2	Take a Test Drive! Exporting to SDK	18
2.1.3	Take a Test Drive! Running the “Hello World” Application	21
2.1.4	Additional Information	24
Chapter 3	Embedded System Design Using the Zynq Processing System and Programmable Logic.....	25
3.1	Adding soft IP in the PL to interface with the Zynq PS.....	25
3.1.1	Take a Test Drive! Check the Functionality of the IP in the PL.....	27
3.1.2	Take a Test Drive! Working with SDK	33
Chapter 4	Debugging with SDK and ChipScope Pro.....	35
4.1	Take a Test Drive! Debugging with Software, Using SDK.....	35
4.2	Take a Test Drive! Debugging Hardware Using ChipScope Software.....	36
Appendix A	40

Chapter 1

Introduction

1.1 About this Guide

This document provides an introduction to using the Xilinx® ISE® WebPACK software to build a Zynq™-7000 All Programmable SoC (AP SoC) design. The examples target the ZedBoard (<http://www.zedboard.org>) using ISE Design Suite 14.1. The required software is included with the ZedBoard kit.

Note: The Test Drives in this document were created using Windows 7 64-bit operating system. Other versions of Windows might provide varied results.

The Zynq-7000 family is the world's first All Programmable SoC. This innovative class of product combines an industry-standard ARM® dual-core Cortex™-A9 MPCore™ processing system with Xilinx 28 nm unified programmable logic architecture. This processor-centric architecture delivers a complete embedded processing platform that offers developers ASIC levels of performance and power consumption, the flexibility of an FPGA, and the ease of programmability of a microprocessor.

This guide describes the design flow for developing a custom Zynq-7000 AP SoC based embedded processing system using the Xilinx ISE WebPACK software tools. It contains the following four chapters:

- **Chapter 1**, (this chapter) provides a general overview.
- **Chapter 2**, “Embedded System Design Using the Zynq Processing System” describes the tool flow for the Zynq Processing System (PS) to create a simple standalone "Hello World" application.
- **Chapter 3**, “Embedded System Design Using the Zynq Processing System and Programmable Logic” describes how to create a system utilizing both the Zynq PS as well as the Programmable Logic (PL).
- **Chapter 4**, “Debugging with SDK and ChipScope Pro” provides debugging techniques via software (using SDK Debug) and Hardware (using the ChipScope™ software).
- **Appendix A**, Application Software describes details of the application needed for the example design used in this guide.

1.1.1 Take a Test Drive!

The best way to learn a software tool is to use it, so this guide provides opportunities for you to work with the tools under discussion. Procedures for sample projects are given in the Test Drive sections, along with an explanation of what is happening behind the scenes and why you need to do it.

Test Drives are indicated by the car icon, as shown beside the heading above.

1.1.2 Additional Documentation

For further information, refer to:

- <http://www.xilinx.com/support/documentation/zynq-7000.htm>
- <http://www.zedboard.org>
- **Xilinx Design Tools: Installation and Licensing Guide** (UG798):
http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_1/iil.pdf
- **Xilinx Design Tools: Release Notes Guide** (UG631):
http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_1/rn.pdf
- **Xilinx® Documentation:**
<http://www.xilinx.com/support/documentation>
- **Xilinx Glossary:**
http://www.xilinx.com/support/documentation/sw_manuals/glossary.pdf
- **Xilinx Support:** <http://www.xilinx.com/support/>

1.1.3 Training Labs

Some Test Drives have associated training labs that you can use for further practice with the given tasks. When applicable, a description of the lab is provided at the end of the Test Drive.

1.2 How Zynq AP SoC and Xilinx software Simplify Embedded Processor Design

The Zynq-7000 All Programmable SoC (AP SoC) reduces system complexity by offering an dual core ARM Cortex-A9 processing system and hard peripherals coupled with Xilinx series 7 28nm programmable logic all integrated on a single SoC. It is the first of its kind in the market and has tremendous potential as a tightly integrated system.

To simplify the design process, Xilinx offers several sets of tools. The ZedBoard kit includes [ISE WebPACK](#) software, and the appropriate device-locked ChipScope Pro tools. ISE WebPACK includes the “PlanAhead Design and Analysis tools, Embedded Processing” for the Zynq XC7Z020 AP SoC, as well as a limited version of the built-in simulator, ISim. The embedded processing component of the ISE WebPACK tools includes Xilinx Platform Studio (XPS) as well as the Software Development Kit (SDK). The Zynq PS may be used without anything programmed in the Programmable Logic (PL). However, in order to use any soft IP in the PL, or to route PS dedicated peripherals to device pins for the PL, programming of the PL is required.

With this, you have all the Xilinx tools required to work with your ZedBoard. It is a good idea to get to know the basic tool names, project file names, and acronyms for these tools. You can find Xilinx software-specific terms in the Xilinx Glossary: http://www.xilinx.com/support/documentation/sw_manuals/glossary.pdf.

Xilinx ISE WebPACK

ISE® *WebPACK*™ design software is the free, downloadable, fully featured front-to-back FPGA design solution for Linux, Windows XP, and Windows 7, supporting the ZedBoard.

And new in ISE Design Suite 14 – WebPACK now supports embedded processing design for the Zynq™-7000 AP SoC..

The ISE WebPACK tools include PlanAhead, Xilinx Platform Studio and the Software Development Kit, amongst others. A complete description of ISE WebPACK is available: <http://www.xilinx.com/products/design-tools/ise-design-suite/ise-webpack.htm>

PlanAhead Design and Analysis Tools

PlanAhead software provides a central cockpit for design entry in RTL, synthesis and verification. PlanAhead offers integration with XPS for embedded processor design (including access to the Xilinx IP catalog), and SDK to complete the embedded processor software design. Implementation is achieved through integration with the ISE toolflow. The implementation flow of your design may be centrally launched from PlanAhead.

- For more information on the embedded design process as it relates to XPS, see the "Design Process Overview" in the *Embedded System Tools Reference Manual (UG111)*: http://www.xilinx.com/support/documentation/xilinx14_1/est_rm.pdf

Note: For this early version of the Zynq development tools, direct simulation of the Processing System is not available.

Xilinx Platform Studio

XPS is the development environment used for designing the hardware portion of your embedded processor system. Specification of the microprocessor, peripherals, and the interconnection of these components, along with their respective detailed configuration, takes place in XPS. You can run XPS in batch mode or using the GUI, which is demonstrated in this guide.

Software Development Kit

The SDK is an integrated development environment, complementary to XPS, that is used for C/C++ embedded software application creation and verification. SDK is built on the Eclipse open-source framework. For more information about the Eclipse development environment, refer to <http://www.eclipse.org>.

Other Components of ISE WebPACK

Other components include:

- Hardware IP for the Xilinx embedded processors
- Drivers and libraries for the embedded software development
- GNU compiler and debugger for C/C++ software development targeting the ARM Cortex-A9 MPCore in the Zynq Processing System
- Documentation
- Sample projects

1.3 What You Need to Set Up Before Starting

Before discussing the tools in depth, it would be a good idea to make sure they are installed properly and that the environments you set up match those required for the "Test Drive" sections of this guide.

1.3.1 Software Installation Requirements:

1. Xilinx ISE WebPACK software tools

The PlanAhead design tool, and Embedded software tools (including XPS and SDK) as well as ISim (limited) are included in the ISE WebPACK design software. Be sure that the latest version of the software is installed. Apply the Device Pack addition, if it is available.

2. Xilinx ChipScope Pro Tools

A version of the Xilinx ChipScope Pro tools that supports the ZedBoard is included with the kit. ChipScope Pro allows you to probe the internal signals of your design much as you would with a logic analyzer. A license will need to be generated to use the ChipScope Pro tools.

3. Software Licensing

Xilinx software uses FLEXnet licensing. A license is required for ISE WebPACK. A WebPACK license does not require a host ID and, therefore, can work on any computer.

(However, the ChipScope Pro tools do require a Host ID.) To WebPACK license, run the Xilinx License Configuration Manager (XLCM), which is automatically launched when the installation program exits. When XLCM starts, it prompts you to register, then automatically places the WebPACK license in the proper directory.

When the software is first run, it performs a license verification process. If it does not find a valid license, the license wizard guides you through the process of obtaining a license and ensuring that the Xilinx tools can use the license.

4. ZedBoard Board Definition file

The ZedBoard Board Definition File takes the form of **zedboard_rev#_v#.xml**, for example, **zedboard_revC_v1.xml** and should be downloaded from <http://www.zedboard.org>, under the Documentation link. Copy this xml file into: <Xilinx ISE 14.1 installation path>/ISE_DS/ISE/data/zynqconfig/board.

1.3.2 Hardware Requirements for this Guide

The ZedBoard is required to complete the tutorial. A second micro-USB cable is required to connect both the USB-JTAG and USB-UART on-board.

Chapter 2

Embedded System Design Using the Zynq Processing System

Now that you've been introduced to the Xilinx® software tools, you'll begin looking at how to use it to develop an embedded system using the Zynq™ Processing System (PS).

Zynq AP SoC consists of an ARM Cortex A9 MPCore PS which includes various dedicated peripherals as well as a configurable PL. This offering can be used in three ways:

1. The Zynq PS can be used independently of the PL.
2. Soft IP may be added in the PL and connected to extend the functionality of the PS. You can use this PS + PL combination to achieve complex and efficient design of a single System On Chip (SOC).
3. Logic in the PL can be designed to operate independently of the PS. PS or JTAG must be used to program the PL however.

The design flow is described in Figure 2-1: Design Flow for Zynq

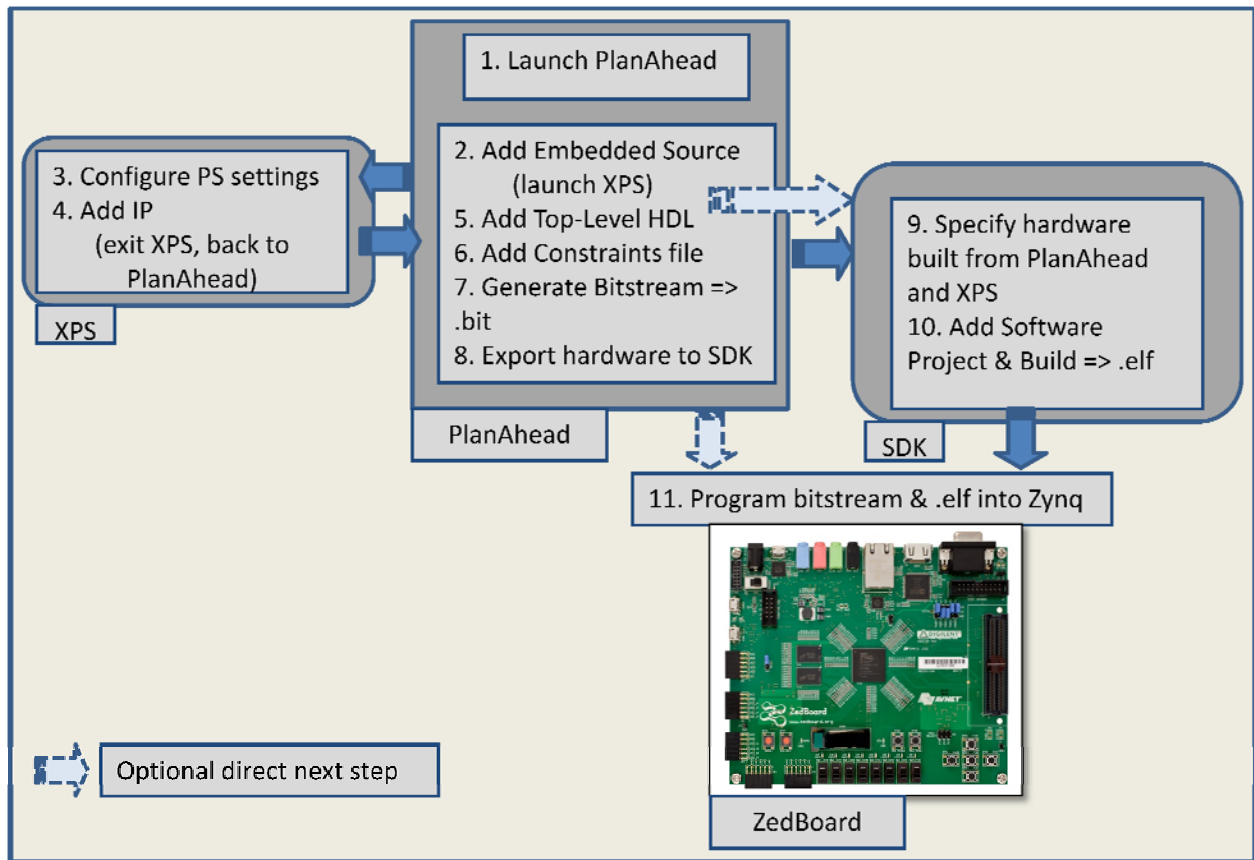


Figure 2-1: Design Flow for Zynq

1. The design and implementation process begins with launching the PlanAhead tools, which is the central cockpit from which design entry through bitstream generation is completed.
2. From PlanAhead, Add an Embedded Source, to include the ARM Cortex A9 Processing System (PS) in the project. XPS is then automatically launched from PlanAhead. Selection of the PS and optional addition of PL peripherals occur within XPS.
3. In XPS, configure settings to select the ZedBoard and make the appropriate design decisions such as selection/de-selection of dedicated PS I/O peripherals, memory configurations, clock speeds, etc.
4. At this point, you may also optionally add soft IP from the IP catalog or create your own customized IP. When finished close XPS to return to PlanAhead.
5. Back in the PlanAhead environment, automatically generate a top-level HDL wrapper for the processing system.
6. Ensure that the appropriate PL related design constraints are defined as required. These constraints would typically be useful to ensure that signals to general purpose I/O such as the switches, LEDs, and Push Buttons on the ZedBoard are routed appropriately. This is done via the creation of a .ucf constraints file in the PlanAhead project.
7. Generate the bitstream for configuring the logic in the PL if soft peripherals or other HDL are included in the design, or if hard peripheral IO was routed through the PL. At this stage, the hardware has been defined in <system.xml>, and if necessary a bitstream <system.bit> has been generated. At this point, the bitstream could be programmed into the FPGA; or it could be done from within SDK.

8. Now that the hardware portion of the embedded system design has been built, export it to SDK to create the software design. (A convenient method to ensure that the hardware for this design is automatically integrated with the software portion is achieved by Exporting the Hardware from PlanAhead to SDK.)
9. From SDK, add a software project to associate with the hardware design exported from PlanAhead.
10. Within SDK, for a standalone application (no operating system) create a Board Support Package (BSP) based on the hardware platform and then develop your user application. Once compiled, a <designname.elf> is generated.
11. The combination of the optional bitstream and the .elf file together programs the hardware and the software functionality into the Zynq device on your ZedBoard.

2.1 Embedded System Construction

Creation of a Zynq system design involves configuring the PS to select appropriate boot devices and peripherals. As long as the selected PS hard peripherals use Multiplexed IO (MIO) connections, and no additional logic or IP is built or routed through the PL, no bitstream is required. This chapter guides you through creating one such design, where only the PS is used.



2.1.1 Take a Test Drive! Creating a New Embedded Project With a Zynq Processing System

For this test drive, you start the ISE® PlanAhead™ design and analysis tool and create a project with an embedded processor system as the top level.

Start the PlanAhead tool.

1. Select **Create New Project** to open the New Project wizard.
2. Use the information in the table below to make your selections in the wizard screens

Wizard Screen	System Property	Setting or Command to Use
Project Name	Project name	Specify the project name.
	Project location	Specify the directory in which to store the project files.
	Create Project Subdirectory	Leave this checked.
Project Type	Specify the type of sources for your design. You can start with RTL or a synthesized EDIF	Use the default selection, RTL Project .
Add Sources	Do not make any changes on this screen.	

Add Existing IP	Do not make any changes on this screen.	
Add Constraints	Do not make any changes on this screen.	
Default Part	Specify	Select Parts .
	Filter	Family: Zynq-7000 Sub-Family: Zynq-7000 Package: CLG 484 Temp Grade: C Speed Grade: -1
	Device	Select xc7z020clg484-1
New Project Summary	Project summary	Review the project summary before clicking Finish to create the project.

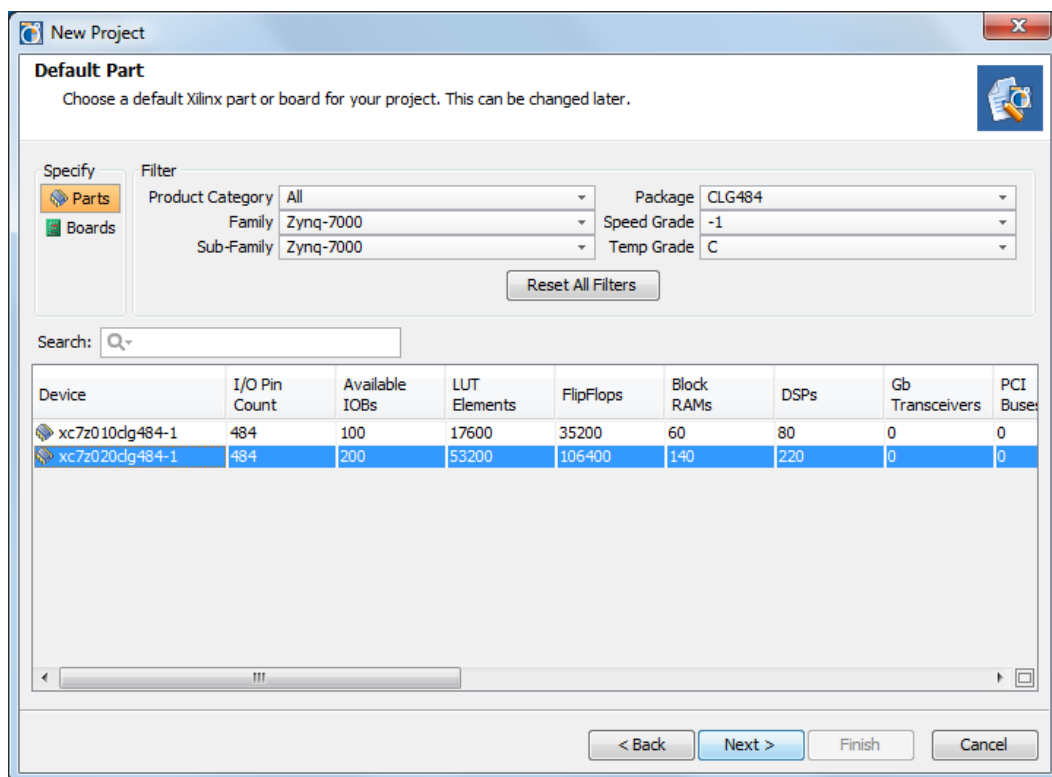


Figure 2-2: New Project Wizard Part selection

When you click **Finish**, the New Project wizard closes and the project you just created opens in the PlanAhead design tool.

IMPORTANT: The Design Runs module at the bottom of the PlanAhead design tool interface has a Strategy column. Review this column to verify that the values are the PlanAhead Defaults (XST 14) and ISE Defaults (ISE 14). If these do not show the correct values, correct them in the Synthesis Settings and Implementation Settings.

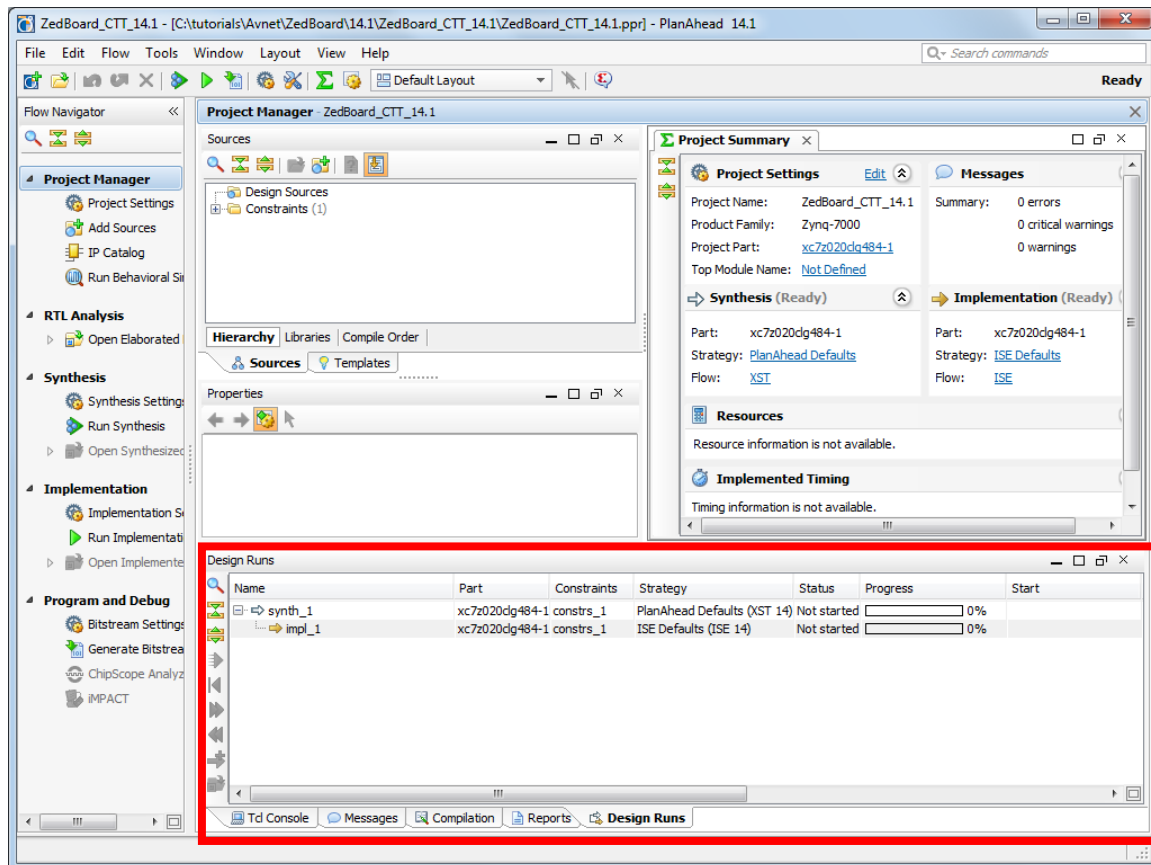


Figure 2-3: PlanAhead GUI

You'll now use the Add Sources wizard to create an embedded processor project.

Click **Add Sources** in the Project Manager.

The Add Sources wizard opens.

1. Select the **Add or Create Embedded Sources** option and click **Next**.
2. In the Add or Create Embedded Source window, click **Create Sub-Design**.
3. Type a name for the module and click **OK**. For this example, use the name **system**.

The module you created displays in the sources list.

Click **Finish**.

XPS opens, and asks if you want to add the Processing System7 to the system.

4. Click **Yes**.

Note: The 14.1 Base System Builder does not support the Processing System.

The XPS System Assembly View opens with the Zynq tab displayed.

Click the Bus Interfaces tab. Notice that `processing_system7` was added as shown in Figure 2-4.

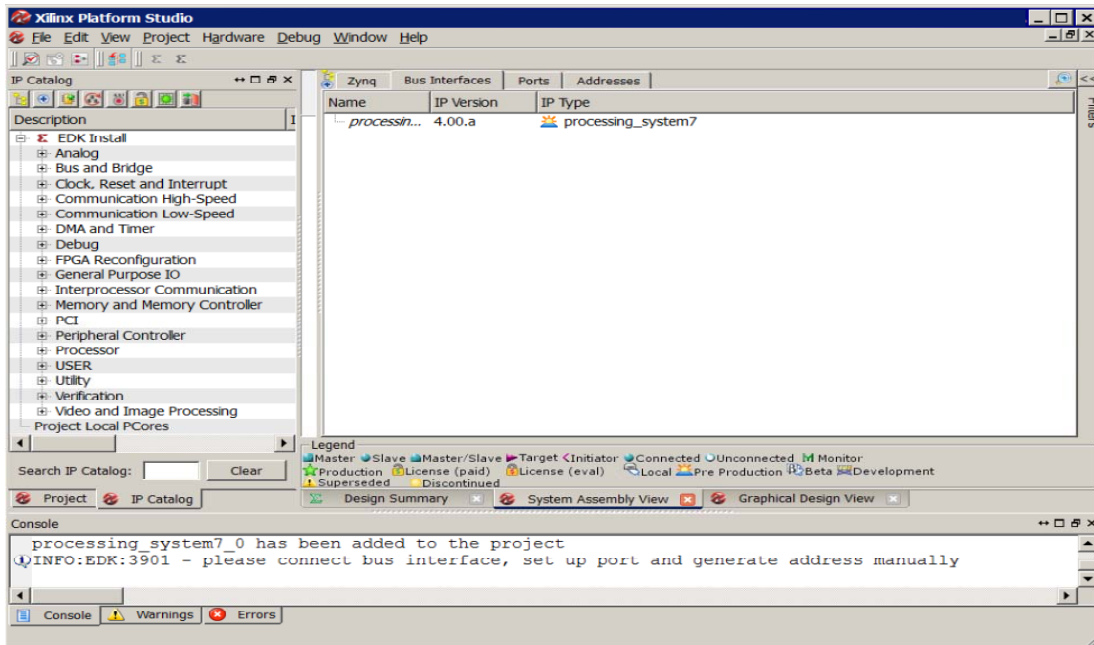


Figure 2-4: XPS System Assembly View

Click the Zynq tab in the System Assembly View to open the Zynq Processing System block diagram

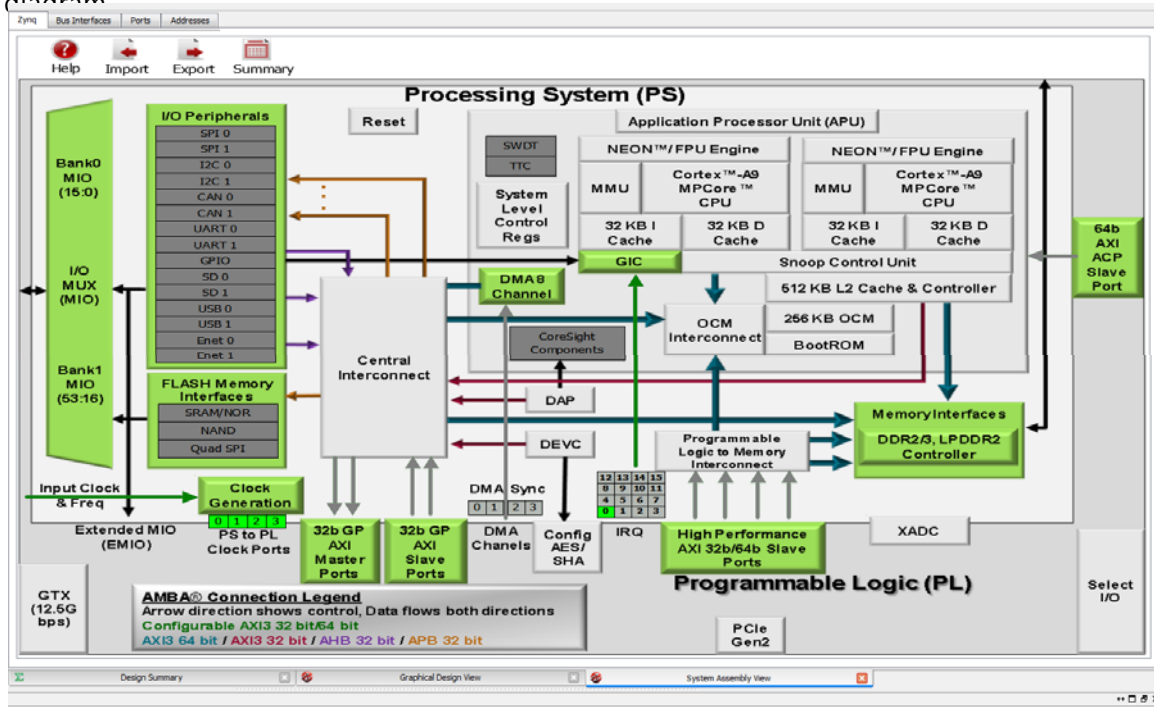


Figure 2-5 : Zynq Processing System

Review the contents of the block diagram. The green colored blocks in the Zynq Processing System diagram are items that are configurable within the Zynq tab. You can click a green block to open the coordinating configuration window.



- Click the **Import Zynq Configurations** button **Import**.

The Import Zynq Configurations dialog box opens.

- Select a configuration template file. The template selected by default is the one in the installation path on your local machine that corresponds to the ZedBoard. To select the Zedboard configuration template, Click on the “+” sign in the User Template section, and navigate to ‘zedboard_revC_v1.xml’ in your installation path. This xml file resides in: <Xilinx ISE 14.1 installation path>/ISE_DS/ISE/data/zynqconfig/board. Select Open to add this template to the User Template section.

7. Select **zedboard_RevC_v1.xml** in the User Template section.

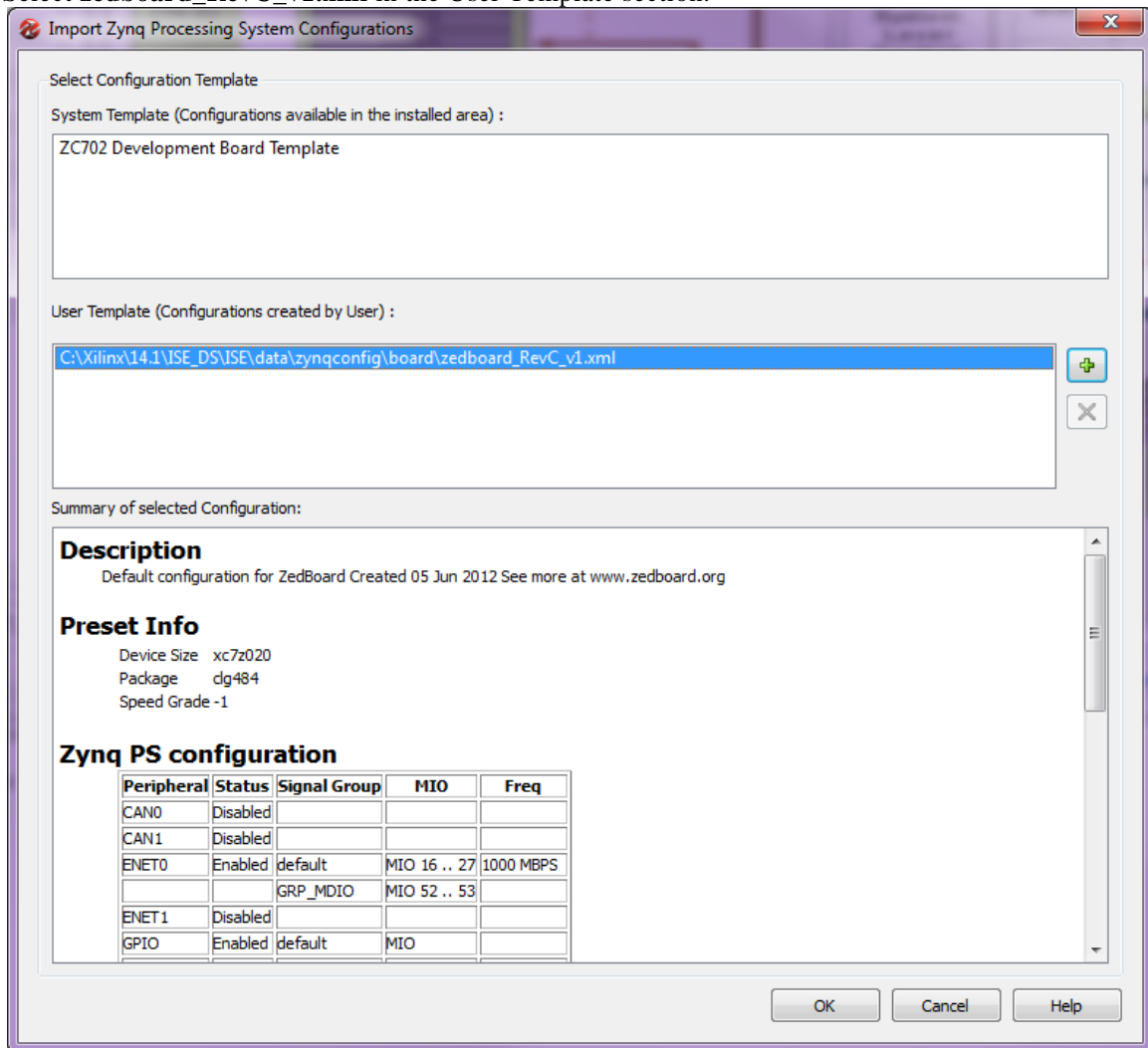


Figure 2-6: Import Zynq Configurations Dialog Box

8. Click **OK**.
9. In the confirmation window that opens to verify that the Zynq MIO Configuration and Design will be updated, click **Yes**.
10. Note the changes to the Zynq block diagram. The I/O Peripherals become active.

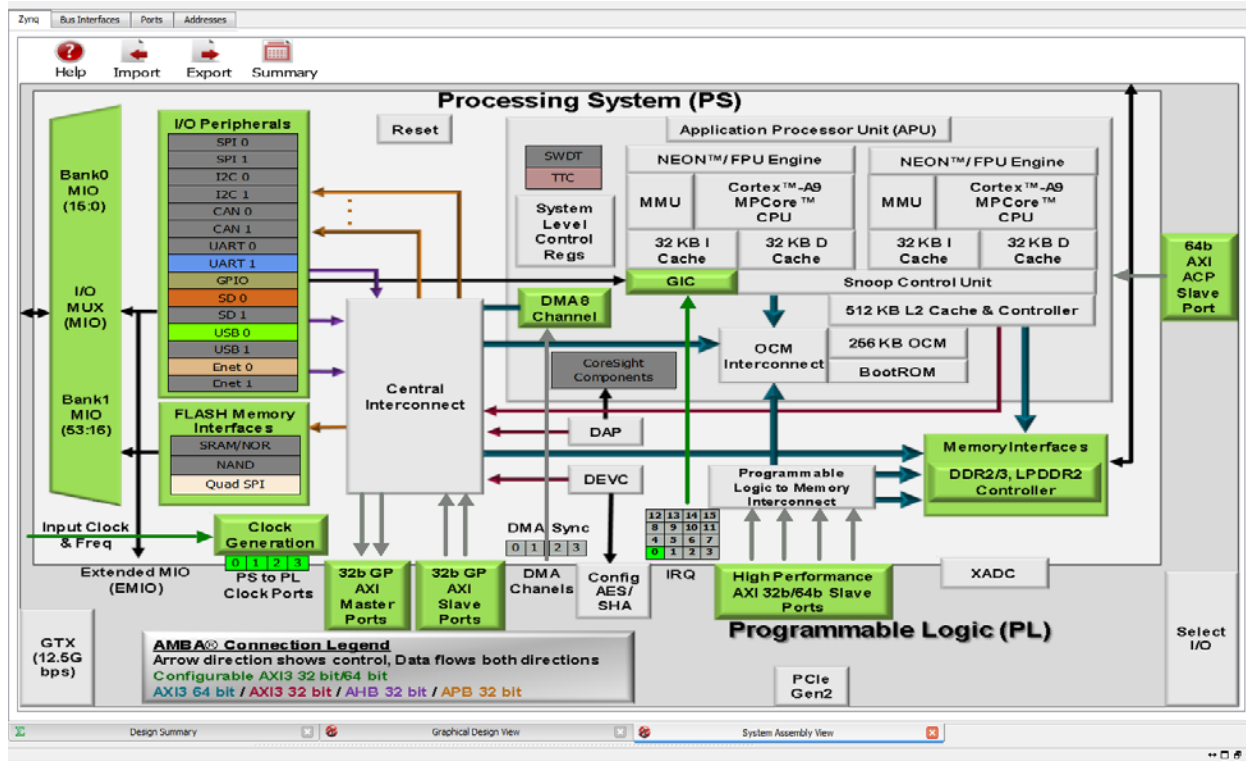


Figure 2-7: Updated Zynq Block Diagram

11. In the block diagram, click the green **I/O Peripherals** box.

As shown in the figure, many peripherals are now enabled in the Processing System with some MIO pins assigned to them as per the board layout of the ZedBoard. For example, UART1 is enabled and UART0 is disabled. This is because UART1 is connected to the USB-UART bridge chip on this board.

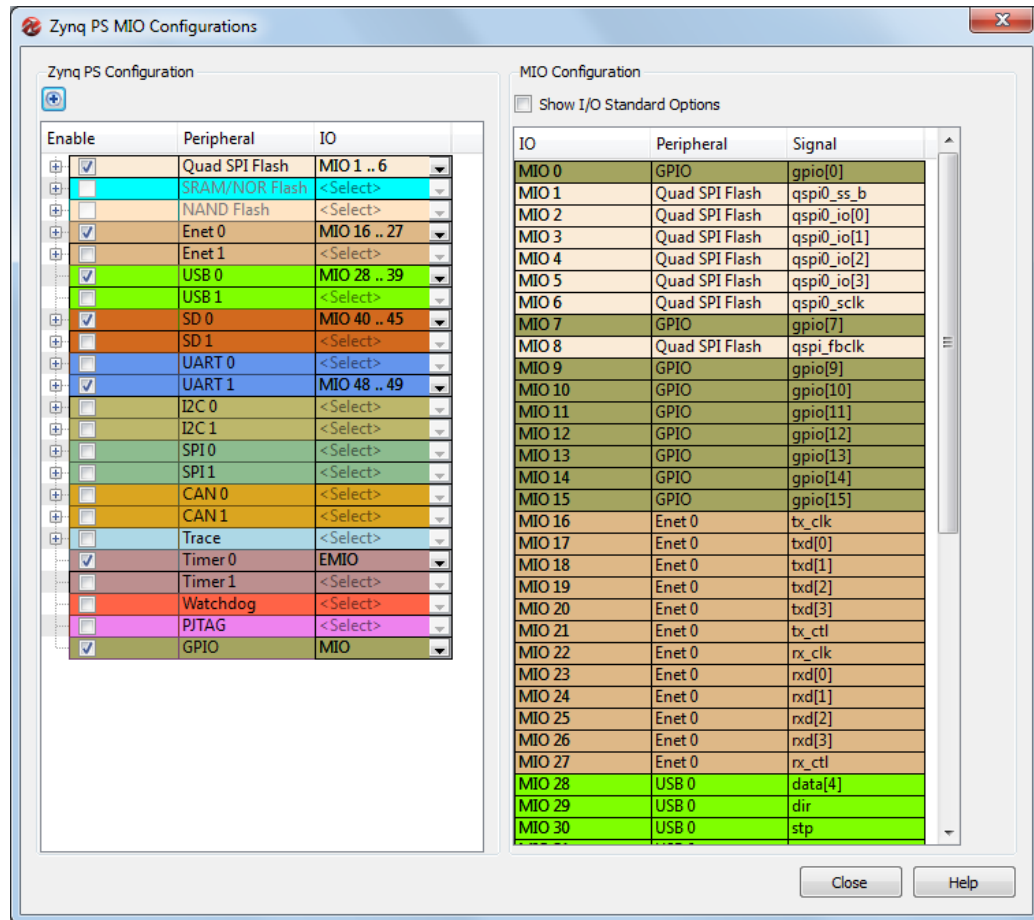


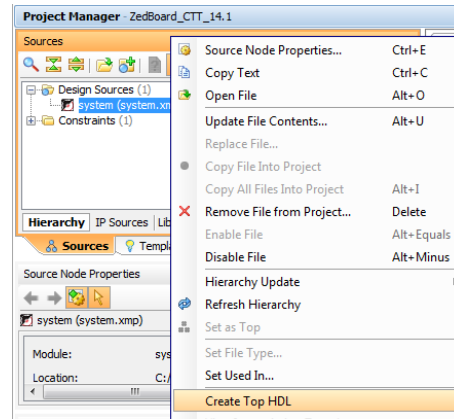
Figure 2-8: Zynq PS MIO Configurations Window

12. Close the Zynq PS MIO Configurations window.
13. Close the XPS window. The active PlanAhead tool session updates with the project settings.

2.1.2 Take a Test Drive! Exporting to SDK

In this test drive, you will launch SDK from the PlanAhead tool.

1. Under **Design Sources** in the Sources pane, select and right-click **system (system.xmp)** and select **Create Top HDL**.



PlanAhead generates the system_stub.v top-level module for the design.

2. In the PlanAhead tool, Select **File > Export > Export Hardware**.

The Export Hardware dialog box opens. By default, the Export Hardware check box is checked.

3. Check the **Launch SDK** check box.
4. Click **OK**; SDK opens.

Notice that when SDK launches, the hardware description file is automatically read in to create the system_hw_platform. The system.xml tab shows the address map for the entire Processing System.

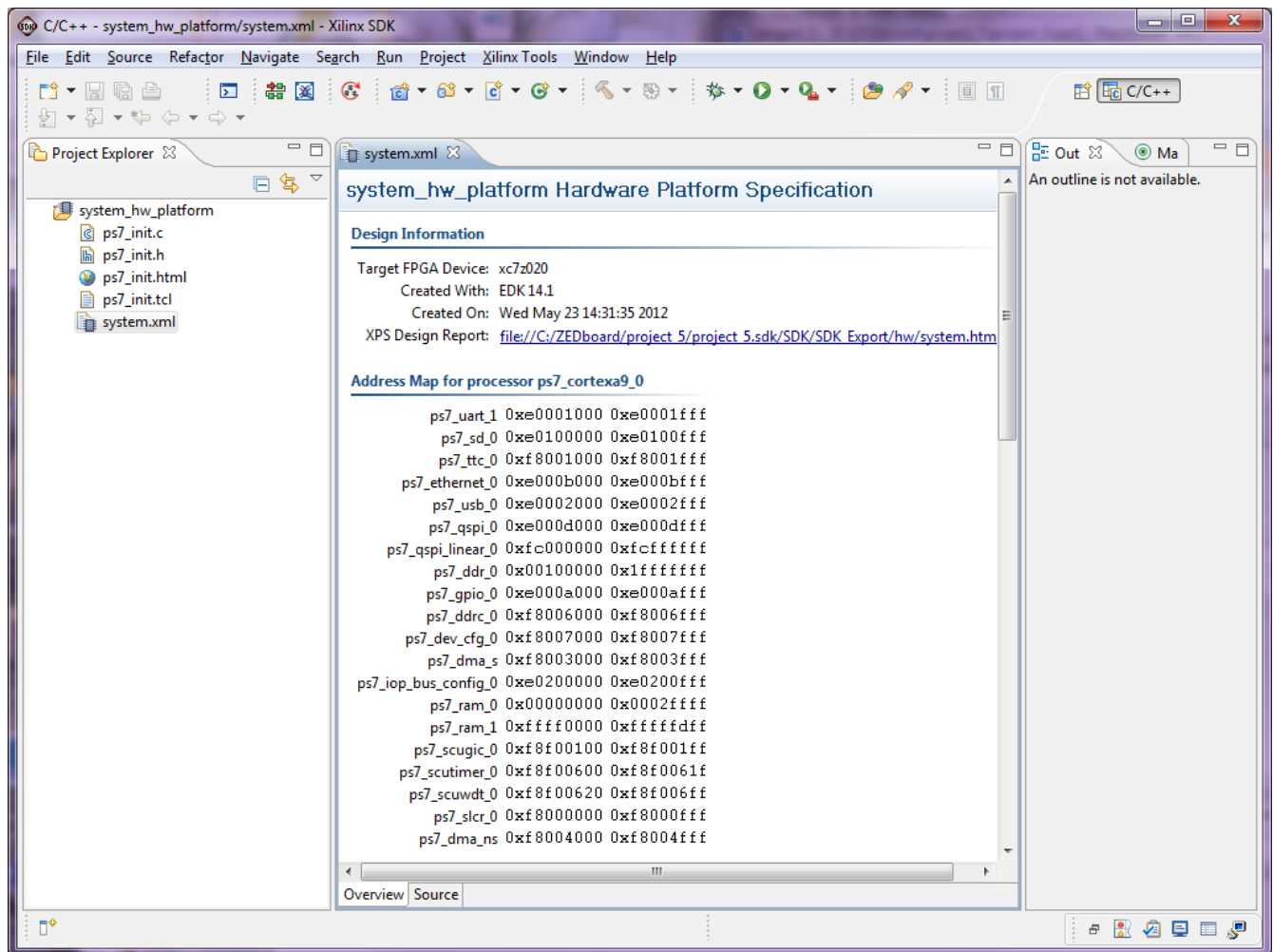


Figure 2-9: Address Map in SDK system.xml Tab

What Just Happened?

The PlanAhead design tool exported the Hardware Platform Specification for your design (system.xml in this example) to SDK. In addition to system.xml, there are four more files relevant to SDK. They are **ps7_init.c**, **ps7_init.h**, **ps7_init.tcl**, and **ps7_init.html**. The .tcl file is used by XMD for configuring the PS when using JTAG.

The system.xml file opens by default when SDK launches. The address map of your system read from this file is shown by default in the SDK window.

The **ps7_init.c** and **ps7_init.h** files contain the initialization code for the Zynq Processing System and initialization settings for DDR, clocks, plls, and MIOs. SDK uses these settings when initializing the processing system so that applications can be run on top of the processing system. There are some settings in the processing system that are fixed for the ZedBoard.

What's Next?

Now you can start developing the software for your project using SDK. The next sections help you create a software application for your hardware platform.

2.1.3 Take a Test Drive! Running the “Hello World” Application

1. Connect the 12V AC/DC converter power cable to the ZedBoard barrel jack.
2. Connect a USB micro cable between the Windows Host machine and the ZedBoard JTAG (J17).
3. Connect a USB micro cable to the USB UART connector (J14) on the ZedBoard with the Windows Host machine. This is used for USB to serial transfer.
4. Power on the board using the switch indicated in Figure 2-7: ZedBoard Power switch and Jumper settings.

IMPORTANT: *Ensure that jumpers are set as shown in the figure.*

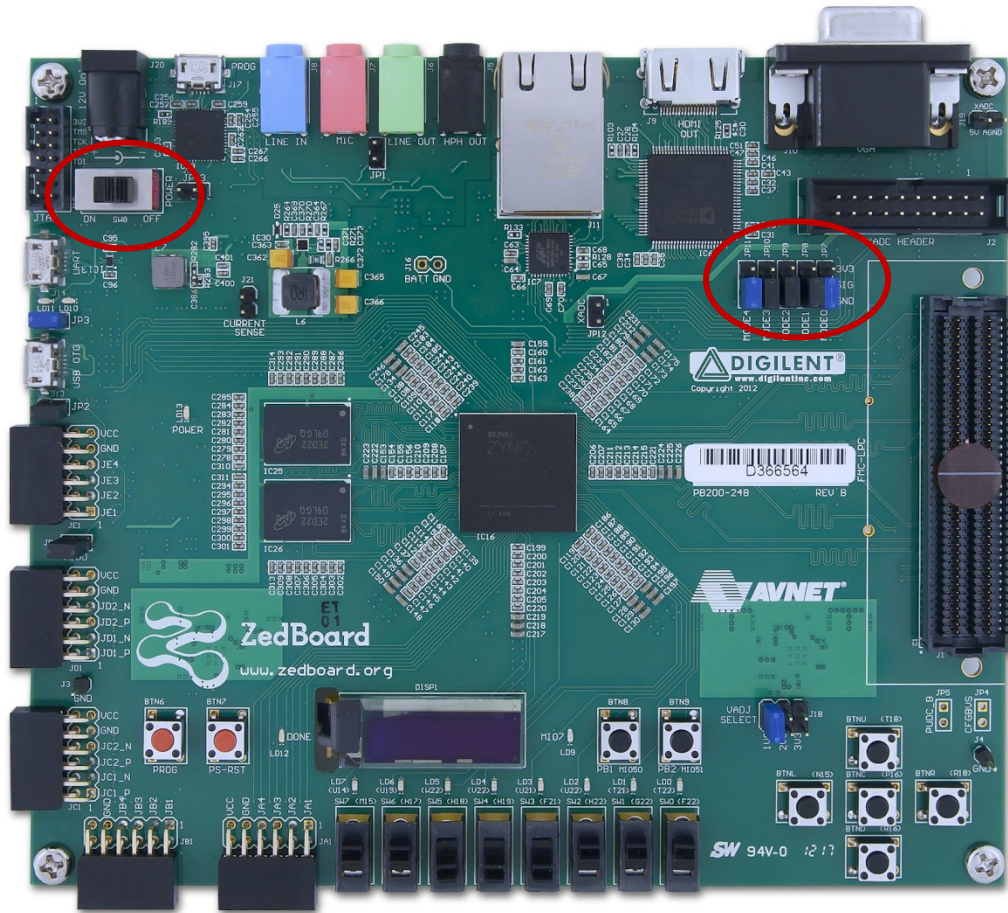



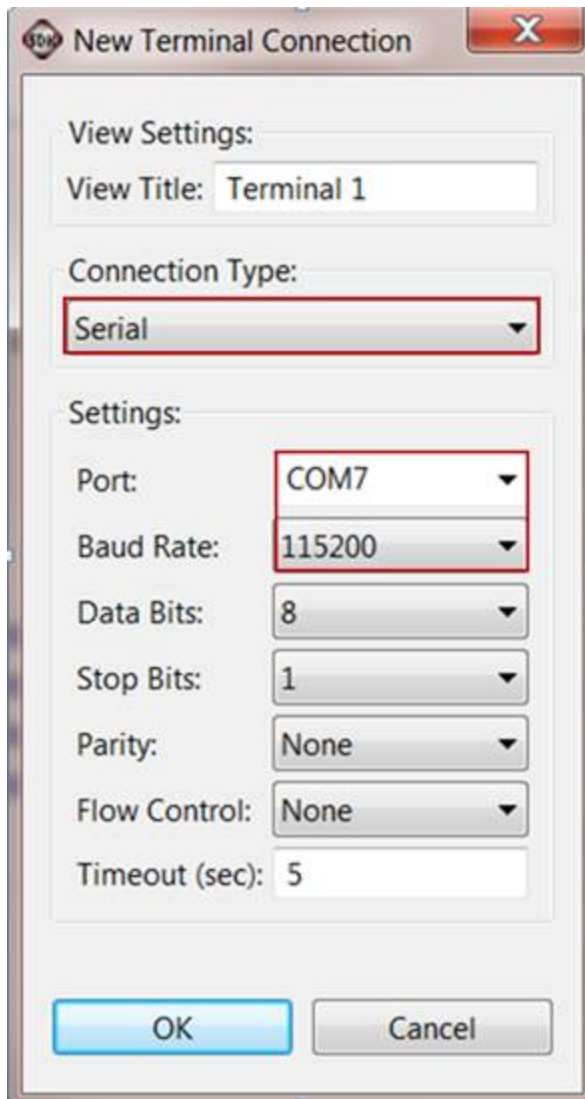
Figure 2-10: ZedBoard Power switch and Jumper settings

5. Open a serial communication utility for the COM port assigned on your system.

Note: The default configuration for Zynq Processing System is: Baud rate 115200; 8 bit; Parity: none; Stop: 1 bit; Flow control: none.

To open a serial communication terminal in SDK:

Select **Window > Show view > Terminal** and click  in the console view area. Configure it with the parameters as shown below (replacing COM7 with the appropriate COM port number, verify using Control Panel > Device Manager).



6. Select **File > New > Xilinx C Project**.
7. Select **Hello World** in the template list and keep the remaining default options. The location of your project, hardware platform used, and processor are visible in this window. For now the processor used is ps7_cortexa9_0.
8. Click **Next**.
9. On the next page, the BSP for this project is selected. Click **Finish** to generate the BSP for the Hello World application. Wait for the process to complete; when you see the 'hello_world_0' project with its 4 sub-directories: Binaries, Includes, Debug and src, it is complete.
10. The Hello World application and its BSP are both compiled and the .elf file is generated.
11. Right-click **hello_world_0** and select **Run as > Run Configurations**.
12. Right-click **Xilinx C/C++ ELF** and click **New**.

13. The new run configuration is created named hello_world_0 Debug.
14. The configurations associated with the application are pre-populated in the Main tab of the launch configurations.
15. Click the **Device Initialization** tab in the launch configurations and check the settings here.
16. Notice that there is a configuration path to the initialization TCL file. The path of ps7_init.tcl is mentioned here. This is the file that was generated when you imported your design into SDK; it contains the initialization information for the processing system when using JTAG.
17. The STDIO Connection tab is available in the launch configurations settings. You can use this to have your STDIO connected to the console. We will not use this now because we have already launched a serial communication utility. There are more options in launch configurations but we will focus on them later.
18. Click **Run**.
19. "Hello World" appears on the serial communication utility.
20. Close SDK.

Note: There was no bitstream download required for the above software application to be executed on the ZedBoard. The ARM Cortex A9 dual core is already present on the board. Basic initialization of this system to run a simple application is done by the device initialization TCL script.

2.1.4 Additional Information

Board Support Package

The BSP is the support code for a given hardware platform or board that helps in basic initialization at power up and helps software applications to be run on top of it. It can be specific to some operating systems with bootloader and device drivers.

Standalone Application

Standalone applications do not utilize an Operating System (OS). They are sometimes also referred to as bare-metal applications. Standalone applications have access to basic processor features such as caches, interrupts, and exceptions, as well as the basic processor features. These basic features include standard input/output, profiling, abort, and exit. It is a single threaded semi-hosted environment.

The application you ran in this chapter was created on top of a standalone BSP.

Chapter 3

Embedded System Design Using the Zynq Processing System and Programmable Logic

One of the unique features of using the Zynq™ AP SoC as an embedded design platform is in using the Zynq PS for its ARM Cortex A9 MPCore processing system as well as the available PL.

In this chapter we will be creating a design with:

- PL-based AXI GPIO and AXI Timer with interrupt from PL to PS section
- ChipScope™ IP instantiated in the PL
- Zynq PS GPIO pin connected through the PL pins routed via the Extended MIO (EMIO) interface

The flow of this chapter is similar to that in **Chapter 2**. If you have skipped that chapter, you might want to look at it because we will refer to it many times in this chapter.

3.1 Adding soft IP in the PL to interface with the Zynq PS

Complex soft peripherals can be added into the PL to be tightly coupled with the Zynq PS. This section covers a simple example with AXI GPIO, AXI Timer with interrupt, PS section GPIO pin connected to PL side pin via EMIO interface, and ChipScope instantiation for verification.

In this section, you'll create a design to check the functionality of the AXI GPIO, AXI Timer with interrupt instantiated in PL, and PS section GPIO with EMIO interface. The block diagram for the system is as shown in Figure 3-1: Block Diagram.

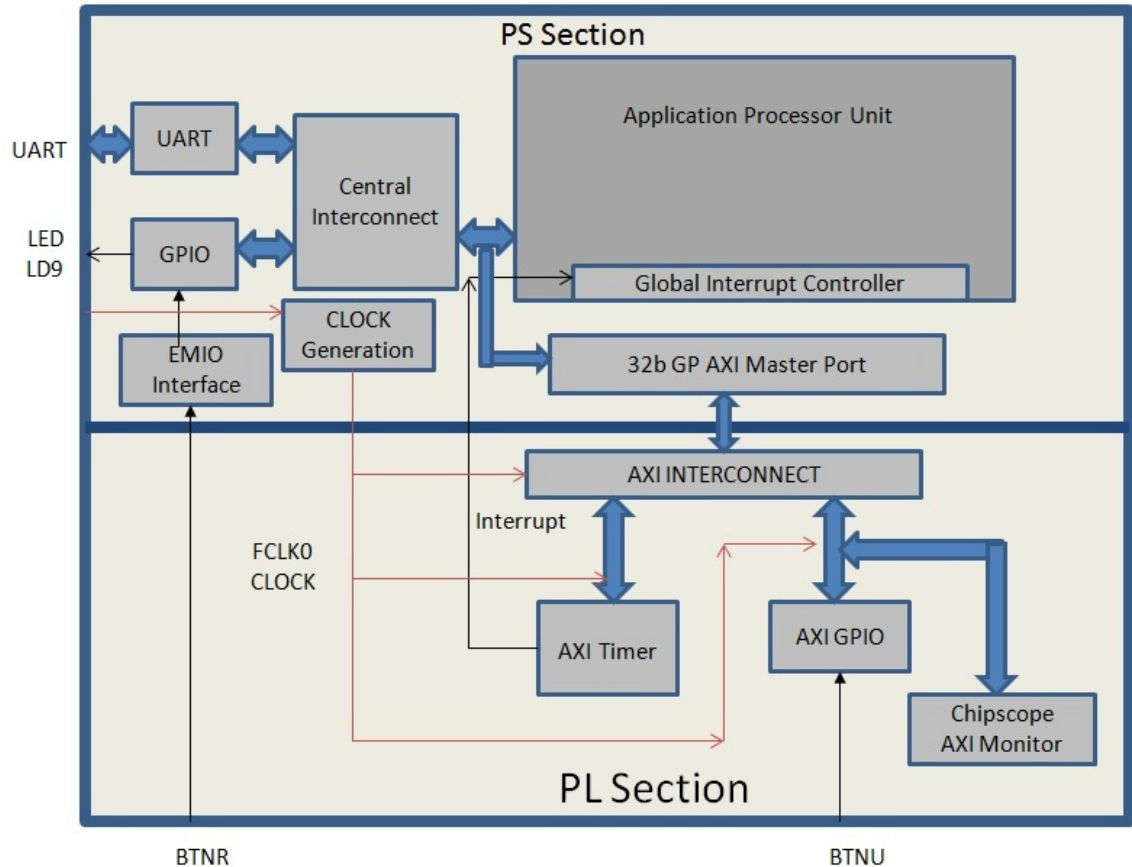


Figure 3-1: Block Diagram

This system covers the following connections:

- The PL-side AXI GPIO has only 1 bit channel width and it is connected to the push-button switch 'BTNU' on the ZedBoard.
- The PS section GPIO also has a 1 bit interface routed to PL pin via EMIO interface and connected to the push-button switch 'BTNR' on the board.
- In the PS section another 1 bit GPIO is connected to the LED 'LD9' on board which is on the MIO.
- An AXI timer interrupt is connected from PL to PS section interrupt controller. The timer starts when the user presses any of the selected push buttons on board and toggles the LED 'LD9' on board

You will write application software, which takes input from the user to select the push button switch on the board and waits for the user to press that particular push button. When the push button is pressed, the timer starts automatically, switches OFF the LED and waits for the timer interrupt to happen. After the Timer Interrupt, the LED switches ON and execution starts again, and it waits for a valid selection from the user.

You will add the ChipScope Integrated Controller (ICON) and AXI Monitor IPs to the design so that in a later section you can learn how to debug hardware using the AXI monitor.

The sections of **Chapter 2** are valid for this design flow also. You'll use the system created in that chapter and pick up the procedure following **2.1.1 Take a Test Drive! Creating a New Embedded Project With a Zynq Processing System.**

3.1.1 **Take a Test Drive! Check the Functionality of the IP in the PL**

In this test drive, you'll check the functionality of the AXI GPIO, AXI Timer with interrupt instantiated in PL and EMIO interface.

1. In the PlanAhead tool Sources pane, invoke XPS by double-clicking system_i-system(system.xmp).

This is the embedded source you created in **Take a Test Drive! Creating a New Embedded Project With a Zynq Processing System, page 10.**

2. In the XPS System Assembly View, click the **Bus Interfaces** tab.
3. From the IP catalog, expand **General Purpose IO** and double-click **AXI General Purpose IO** to add it.

A message appears asking if you want to add the axi_gpio 1.01.b IP instance to your design.

4. Click **Yes**.

The configuration window for GPIO opens.

5. Expand Channel 1 to view configuration parameters for channel 1.
6. Notice GPIO Data Channel Width with value 32. Change it to 1 as your design needs only one bit of input to work. Leave all other parameters as they are.
7. Click **OK**.

A message window opens with the message "axi_gpio IP with version number 1.01.b is instantiated with name axi_gpio_0". It will ask you to determine to which processor to connect. Remember you are designing with a dual core ARM processor. The message also says XPS will make the Bus Interface Connection, assign the address, and make IO ports external.

The default choice of processor is "processing_system7_0". Do not change this.

8. Click **OK**.

There are a few connections that are not done automatically and must be done manually.

NOTE: The AXI interconnect automatically gets instantiated between the PL IPs and the PS Section Interconnect. In this example, AXI GPIO is connected to PS through AXI interconnect.

9. In the IP Catalog, expand **DMA and Timer** and double-click the **AXI Timer/Counter** IP to add it.

A dialog box appears asking if you want to add the axi_timer_1.03.a IP instance to your design.

10. Click **Yes**.

The configuration window for TIMER opens. Leave all other parameters as they are.

11. Click **OK**.

A message window opens with the message "axi_timer IP with version number 1.03.a is instantiated with name axi_timer_0." It will ask you to determine to which processor to connect. Remember you are designing with a dual core ARM processor. The message also says XPS will make the Bus Interface Connection, assign the address, and make IO ports external.

The default choice of processor is "processing_system7_0". Do not change this.

12. Click **OK**.

You'll connect the AXI timer Interrupt to the PS section interrupt manually later in this section.

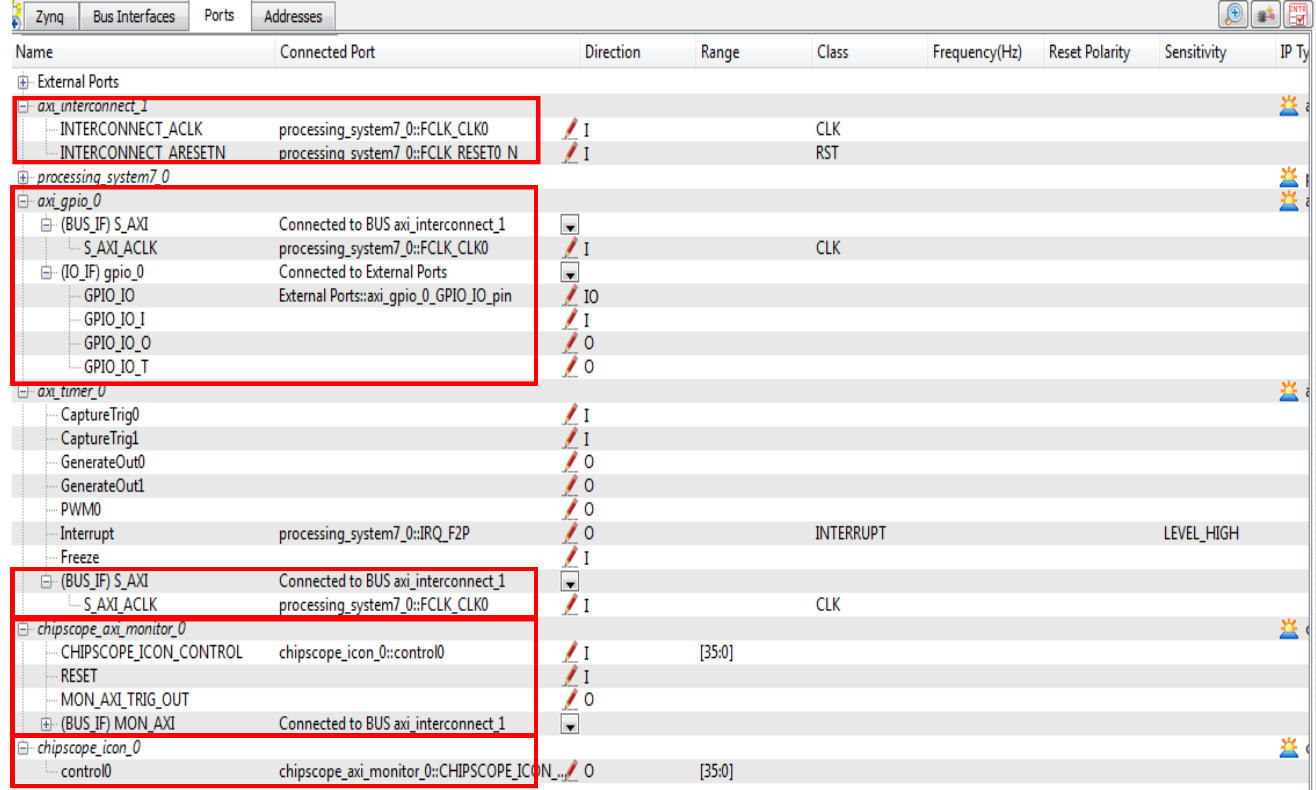
13. In the IP Catalog, expand **Debug** and add two IPs to the design: **ChipScope AXI Monitor** and **ChipScope Integrated Controller**. Do not make changes to the configuration of either IP.

14. Click the **Ports** tab, which lists the IPs and their ports. Expand axi_interconnect_1, axi_gpio_0, axi_timer_0, chipscope_axi_monitor_0, and chipscope_icon_0.

15. Review the following IP connections. If any of these aren't already connected, connect them now

IP	Port	Connection
axi_interconnect_1	INTERCONNECT_ACLK	Processing_ps7_0 : FCLK_CLK0
	INTERCONNECT_ARESETN	Processing_ps7_0::FCLK_RESET0_N
axi_gpio_0	(BUS_IF) S_AXI::S_AXI_ACLK	Processing_ps7_0 : FCLK_CLK0
	(IO_IF) gpio_0::GPIO_IO	External Port ::axi_gpio_0_GPIO_IO_pin
axi_timer_0	(BUS_IF) S_AXI::S_AXI_ACLK	Processing_ps7_0 : FCLK_CLK0
Chipscope_axi_monitor_0	CHIPSCOPE_ICON_CONTROL	Chipscope_icon_0 ::control0
	(BUS_IF) MON_AXI::MON_AXI_ACLK	Processing_ps7_0 : FCLK_CLK0
Chipscope_icon_0	Control0	Chipscope_axi_monitor0::CHIPSCOPE_ICON_CONTROL

Your Ports tab should be similar to Figure 3-2: Completed Port Connections



Name	Connected Port	Direction	Range	Class	Frequency(Hz)	Reset Polarity	Sensitivity	IP Ty
External Ports								
axi_interconnect_1								
INTERCONNECT_ACLK	processing_system7_0::FCLK_CLK0	I		CLK				
INTERCONNECT_ARESETN	processing_system7_0::FCLK_RESET0_N	I		RST				
processing_system7_0								
axi_gpio_0								
(BUS_IF) S_AXI	Connected to BUS axi_interconnect_1							
S_AXI_ACLK	processing_system7_0::FCLK_CLK0	I		CLK				
(IO_IF) gpio_0	Connected to External Ports							
GPIO_IO	External Ports::axi_gpio_0_GPIO_IO_pin	IO						
GPIO_IO_I		I						
GPIO_IO_O		O						
GPIO_IO_T		O						
axi_timer_0								
CaptureTrig0		I						
CaptureTrig1		I						
GenerateOut0		O						
GenerateOut1		O						
PWM0		O						
Interrupt	processing_system7_0::IRQ_F2P	O		INTERRUPT			LEVEL_HIGH	
Freeze		I						
(BUS_IF) S_AXI	Connected to BUS axi_interconnect_1							
S_AXI_ACLK	processing_system7_0::FCLK_CLK0	I		CLK				
chipscope_axi_monitor_0								
CHIPSCOPE_ICON_CONTROL	chipscope_icon_0::control0	I	[35:0]					
RESET		I						
MON_AXI_TRIG_OUT		O						
(BUS_IF) MON_AXI	Connected to BUS axi_interconnect_1							
chipscope_icon_0								
control0	chipscope_axi_monitor_0::CHIPSCOPE_ICON_0	O	[35:0]					

Figure 3-2: Completed Port Connections

16. Collapse all IPs and expand processing_system7_0. If the following port connection is not made, do it now. It should look like Figure 3-3: Ports Tab with processing_system7_0 expanded and M_AXI_GPO_ACLK connected

IP	Port	Connection
Processing_system7_0	(BUS_IF) M_AXI_GPO:: M_AXI_GPO_ACLK	processing_system7_0 :: FCLK_CLK0

Name	Connected Port	Direction	Range	Class	Frequency(Hz)
processing_system7_0					
- M_AXI_GP0_ARESETN		0		RST	
- FCLK_CLK3		0		CLK	
- FCLK_CLK2		0		CLK	
- FCLK_CLK1		0		CLK	
- FCLK_CLK0	processing_system7_0::[M_AXI_GP0]::M_AXI_GP0_ACLK axi_gpio_0::[S_AXI]::S_AXI_ACLK axi_interconnect_1::[S_AXI_CTRL]::INTERCONNECT_ACLK axi_timer_0::[S_AXI]::S_AXI_ACLK	0		CLK	
- FCLK_CLKTRIG3_N		1			
- FCLK_CLKTRIG2_N		1			
- FCLK_CLKTRIG1_N		1			
- FCLK_CLKTRIG0_N		1			
- FCLK_RESET3_N		0		RST	
- FCLK_RESET2_N		0		RST	
- FCLK_RESET1_N		0		RST	
- FCLK_RESET0_N	axi_interconnect_1::INTERCONNECT_ARESETN	0		RST	
- IRQ_F2P	L to H: No Connection	1		INTERRUPT	
- Core0_nFIQ		1		INTERRUPT	
- Core0_nIRQ		1		INTERRUPT	
- Core1_nFIQ		1		INTERRUPT	
- Core1_nIRQ		1		INTERRUPT	
- IRQ_P2F_QSPI		0		INTERRUPT	
- IRQ_P2F_GPIO		0		INTERRUPT	
- IRQ_P2F_USB0		0		INTERRUPT	
- IRQ_P2F_ENET0		0		INTERRUPT	
- IRQ_P2F_ENET_WAKE0		0		INTERRUPT	
- IRQ_P2F_SDIO0		0		INTERRUPT	
- IRQ_P2F_I2C0		0		INTERRUPT	
- IRQ_P2F_CAN0		0		INTERRUPT	
- IRQ_P2F_UART1		0		INTERRUPT	
⊖ (BUS_IF) M_AXI_GP0	Connected to BUS axi_interconnect_1				
- M_AXI_GP0_ACLK	processing_system7_0::FCLK_CLK0	1		CLK	
⊖ (IO_IF) MEMORY_0	Connected to External Ports				
⊖ (IO_IF) PS_REQUIRED_EXTERNAL...	Connected to External Ports				

Figure 3-3: Ports Tab with processing_system7_0 expanded and M_AXI_GP0_ACLK connected

17. Connect the Timer interrupt on the PL side to the PS side interrupt controller by doing the following:

- In the Connected Port column of Processing_System7_0 for IRQ_F2P, click **L to H: No Connection**.

The Interrupt Connection dialog box opens as shown in Figure 3-4.

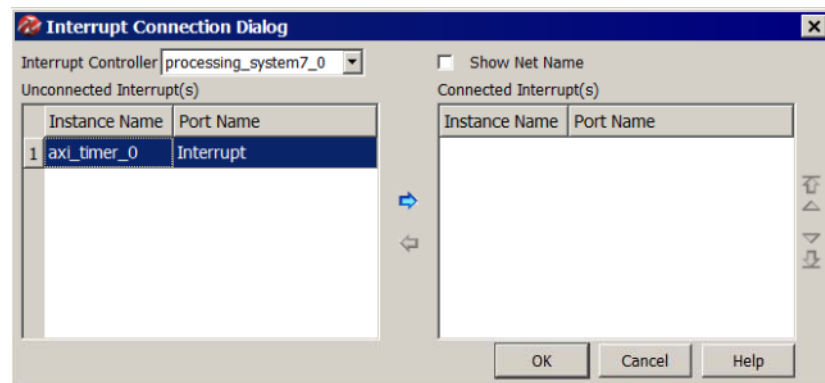


Figure 3-4: Interrupt Connection Dialog Box

- In the Unconnected Interrupts list, select axi_timer_0 and click the right arrow button to move it to the Connected Interrupts list.

Figure 3-5: Interrupt Connection Dialog Box with Connected Interrupt displays the `axi_timer_0` interrupt instance connected with Interrupt ID 91.

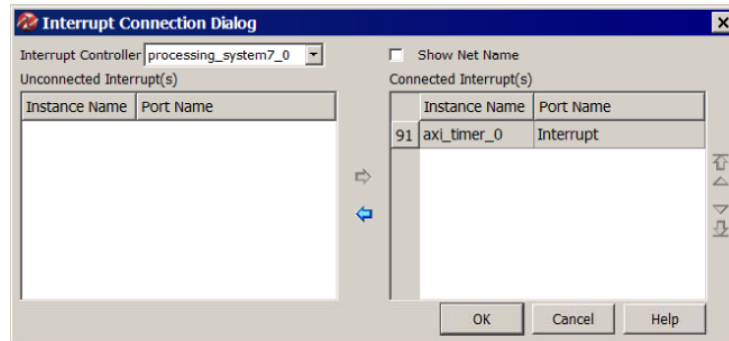


Figure 3-5: Interrupt Connection Dialog Box with Connected Interrupt

- c. Click **OK**.

XPS connects the timer interrupt on the PL side to the PS section interrupt controller.

FCLK_RESET1_N		RST
FCLK_RESET2_N		RST
FCLK_RESET1_N		RST
FCLK_RESET0_N	axi_interconnect_1::INTERCONNECT_ARESETN	RST
IRQ_F2P	L to H: axi_timer_0_interrupt	INTERRUPT
Core0_nFIQ		INTERRUPT
Core0_nIRQ		INTERRUPT
Core1_nFIQ		INTERRUPT
Core1_nIRQ		INTERRUPT

Figure 3-6: Timer Interrupt Connected on the PL Side

18. Click the **Bus Interfaces** tab and expand `chipscope_axi_monitor_0`.
19. In the **Bus Name** column, click **No Connection**. Using the drop-down list that appears, connect `chipscope_axi_monitor` to `axi_gpio_0.S_AXI`.

By making this connection, you can monitor any type of AXI-related transactions on the `axi_gpio_0` slave AXI bus using ChipScope Analyzer.

Name	IP Version	Bus Name	IP Type
axi_interconnect_1	1.06.a		axi_interconnect
processing_system7_0	4.00.a		processing_system7
axi_gpio_0	1.01.b		axi_gpio
axi_timer_0	1.03.a		axi_timer
chipscope_axi_monitor_0	3.03.a		chipscope_axi_monitor
MON_AXI		axi_gpio_0.S_AXI	
chipscope_con_0	1.06.a		chipscope_console

Figure 3-7: Connected chipscope_axi_monitor

20. Route the PS section GPIO to the PL side pad using the EMIO interface by doing the following:
 - d. In the XPS System Assembly View, click the **Zynq** tab.
 - e. Click the green **32b GP AXI Master Ports** button to open the XPS Core Config dialog box.

- f. In the **User** tab, expand the **General** item.
- g. Click to select the **Enable GPIO on EMIO Interface** check box.

If you cannot see the check boxes for the items in the XPS Core Config dialog box, click and drag the right side of the window to expand it.

The **Width of GPIO on EMIO interface** setting is enabled on the next row. The default setting is 64.

- h. Change the GPIO width to **1** and click **OK**.
- i. In the System Assembly View, click the **Ports** tab and expand processing_system7_0. You can see that the GPIO port is not connected to an external port.

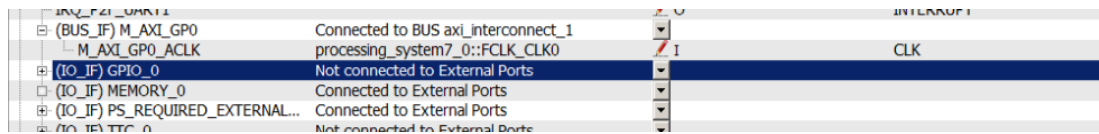


Figure 3-8: GPIO Port Not Connected to External Ports

21. Click the drop-down arrow in the **Connected Port** column and select **Make Ports External**.

Making this connection allows you to assign the PL section pin location to PS GPIO in the user constraint file (UCF) later in this chapter.

22. Run **Project > Design Rule Check**. Review the messages in the Warnings tab.

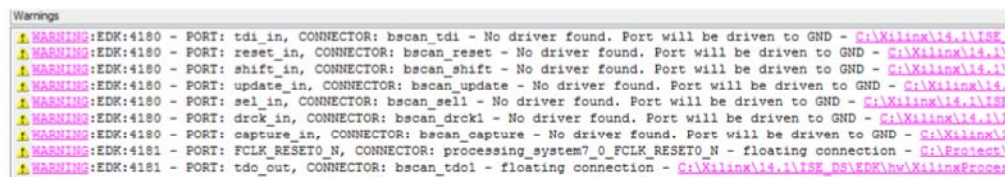


Figure 3-9: Design Rule Check Warnings

23. Close XPS. The PlanAhead™ design tool window becomes active again.
24. In Design Sources, click on your embedded source and then right-click it and select **Create Top HDL**. The PlanAhead tool generates the system_stub.v file.
25. In the Project Manager list of the Flow Navigator, click **Add Sources**.
26. In the dialog box that opens, select **Add or Create Constraints**, then click **Next**.
27. Click **Create File**. In the Create Constraints File dialog box that opens, name the file **system** and click **OK**.
28. Click **Finish**.
29. Expand the **Constraints** folder in the Sources window. Notice that the blank file system.ucf was added inside constrs_1. Double-click system.ucf to open it in the editor.

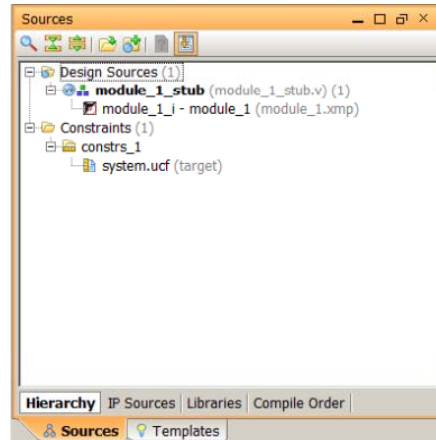


Figure 3-10: system.ucf File Added

30. Type the following text in the UCF file:

```
# Connect to Push Button "BTNU"
```

```
NET axi_gpio_0_GPIO_IO_pin IOSTANDARD=LVC MOS25 | LOC=T18;
```

```
# Connect to Push Button "BTNR"
```

```
NET processing_system7_0_GPIO_pin IOSTANDARD=LVC MOS25 | LOC=R18;
```

The following settings are made:

- The LOC constraint for NET “axi_gpio_0_IO_pin” connects the AXI GPIO pin to the T18 pin of the PL section and physically connects it to the BTNU push button on the board.
- The LOC constraint for NET “processing_system7_0 GPIO pin” connects the PS section GPIO to the FR18 pin of the PL section and physically connects it to the BTNR push button on the board.
- The IOSTANDARD=LVC MOS25 constraint sets both pins to LVC MOS 2.5V I/O standard.

31. Save the UCF.

32. In the Program and Debug list in the Flow Navigator, click **Generate Bitstream**. Ignore any critical warnings that appear.

33. After the Bitstream generation completes, export the hardware and Launch SDK as described in **Chapter 2**. For this design, since there is a bitstream generated for the PL, this will also be exported to SDK.

3.1.2



Take a Test Drive! Working with SDK

SDK launches with the "Hello World" project you created with the Standalone PS in **Chapter 2**.

1. Select **Project > Clean** to clean and build the project again.
2. Open the helloworld.c file and modify the application software code. Refer to **Appendix A, Application Software** for the application software details.
3. Connect and power-on the board.
4. Open the serial communication utility with baud rate set to **115200**.
5. Because you have a bitstream for the PL, you must download the bitstream. To do this, select **Xilinx Tools > Program FPGA**. The Program FPGA dialog box, shown below, opens. It displays the bitstream exported from PlanAhead.

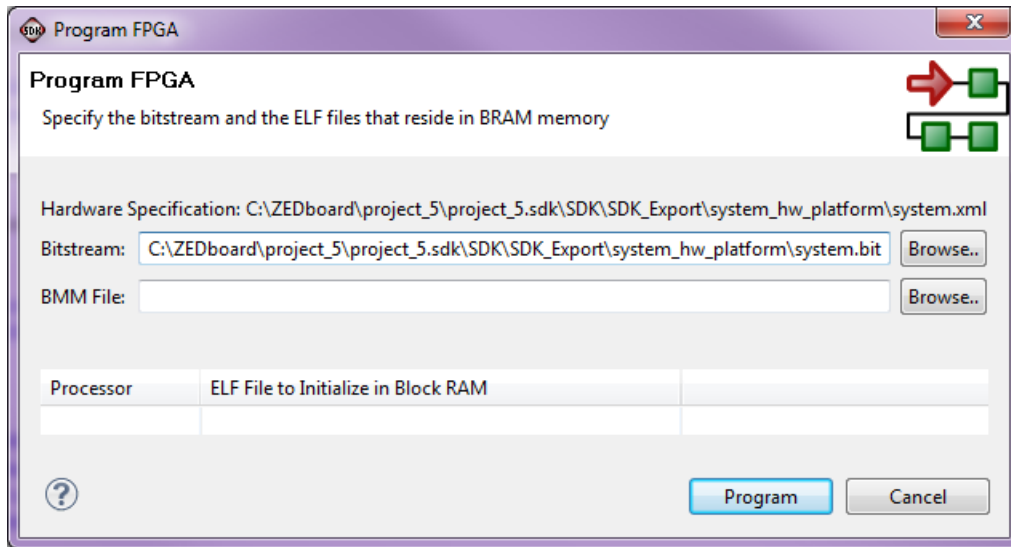


Figure 3-11: Program FPGA Dialog Box

6. Click **Program** to download the bitstream and program the PL. The Blue DONE LED (LD12) will light up.
7. Run the application similar to the steps in **Take a Test Drive! Running the “Hello World” Application**.
8. In the system, the AXI GPIO pin is connected to push button BTNU on the board, and the PS section GPIO pin is connected to push button BTNR on the board via an EMIO interface.
9. Follow the instructions printed on the serial terminal to run the application.

Chapter 4 Debugging with SDK and ChipScope Pro

This chapter describes two types of debug possibilities with the design flow you've already been working with. The first option is debugging with software using SDK. The second option is hardware debug supported by the ChipScope™ software.

4.1 Take a Test Drive! Debugging with Software, Using SDK

First you will try debugging with software using SDK.

1. In the C/C++ Perspective, right-click on the Hello_world_0 Project and select **Debug As > Debug Configurations**. Check that settings are correct for your debug operation.
2. Click **Debug**.
3. A dialog box appears with a question about the reset properties of your system.
4. Click **OK**.

Another dialog box appears to notify you that this kind of launch is configured to open the Debug perspective when it suspends.

5. Click **Yes**. The Debug Perspective opens.

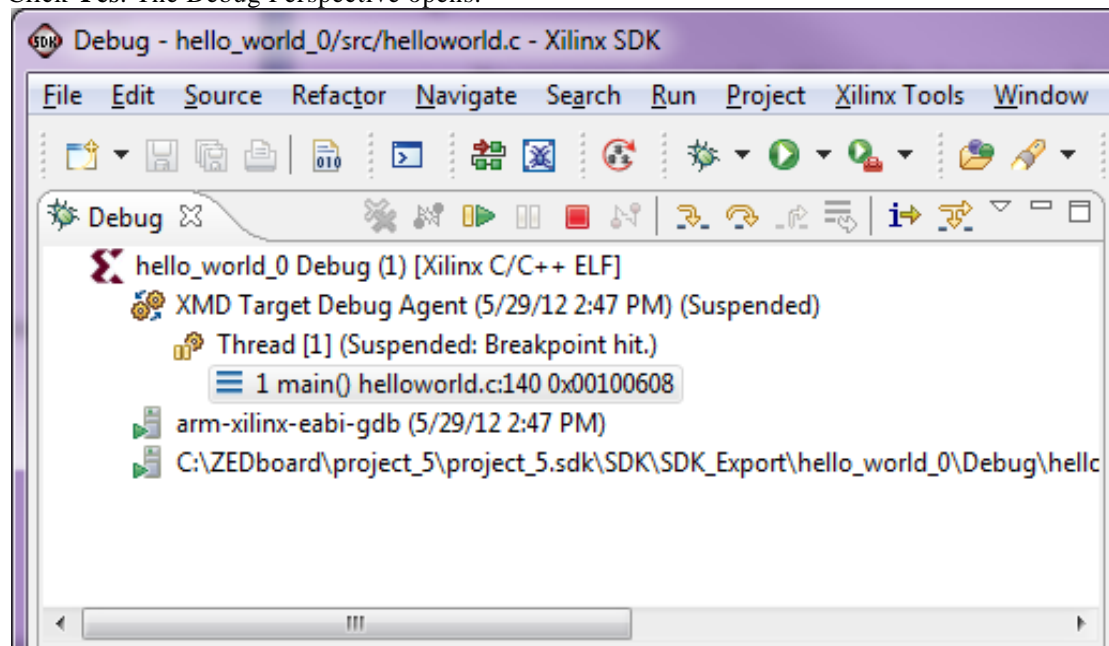


Figure 4-1: Debug Perspective Suspended

The processor is currently sitting at the beginning of `main()` with program execution suspended at line `0x00100608`. You can confirm this information with the Disassembly view, which shows the assembly-level program execution also suspended at `0x00100608`.

Note: If the disassembly view is not visible, select **Window > Show view > Disassembly**.

The `helloworld.c` window also shows execution suspended at the first executable line of C code. Select the Registers view to confirm that the program counter, `pc` register, contains `0x00100608`.

Note: If the Registers window is not visible, select **Window > Show View > Registers**.

6. Double-click in the margin of the `helloworld.c` window next to the line of code that reads `init_platform ()`. This sets a breakpoint at `init_platform ()`. To confirm the breakpoint, review the Breakpoints window.

If the Breakpoints window is not visible, select **Window > Show View > Breakpoints**.

7. Select **Run > Resume** to resume running the program to the breakpoint.

Program execution stops at the line of code that includes `init_platform ()`. The Disassembly and Debug windows both show program execution stopped at `0x00100630`.

8. Select **Run > Step Into** to step into the `init_platform ()` routine.

Program execution suspends at location `0x00100C44`. The call stack is now two levels deep.

9. Select **Run > Resume** again to run the program to conclusion.


When the program completes running, the Debug window shows that the program is suspended in a routine called `exit`. This happens when you are running under control of the debugger.

10. Re-run your code several times. Experiment with single-stepping, examining memory, changing breakpoints, modifying code, and adding print statements. Try adding and moving views.
11. Close SDK.

4.2 Take a Test Drive! Debugging Hardware Using ChipScope Software

Next you will try debugging hardware using the ChipScope software using the same application you created in **3.1.2 Take a Test Drive! Working with SDK**.

Run the application.

1. Close SDK.
2. Open ChipScope Pro™ Analyzer.
3. Make sure the on-board JTAG hardware is connected to the USB port of your computer using USB cable provided.
4. Click the **Open/Search JTAG Cable** button  .

5. Click **OK**.
6. Import a *.cdc file in ChipScope and do the following:
 - a. Select **Dev 1 MyDevice1(XC7Z020)**.
 - b. Select **File > Import**.
 - c. Click **Select New File** and select the chipscope_axi_monitor_0.cdc file from
 <project_path>\<project_name>.srcs\sources_1\edk\system\implementation\chipscope_axi_monitor_0_wrapper.
 - d. Click **OK**.
7. Set a trigger at the “ARVALID” signal by doing the following.
 - a. Expand the Trigger Setup window.
 - b. Double-click M1:MON_AXI_ARADDRCONTROL. For the M1:MON_AXI_ARADDRCONTROL unit, change the value of axi_gpio_0.S_AXI/MON_AXI_AVALID from the default of **X** to **1**. With this setting, any positive transaction on this signal triggers the waveform.

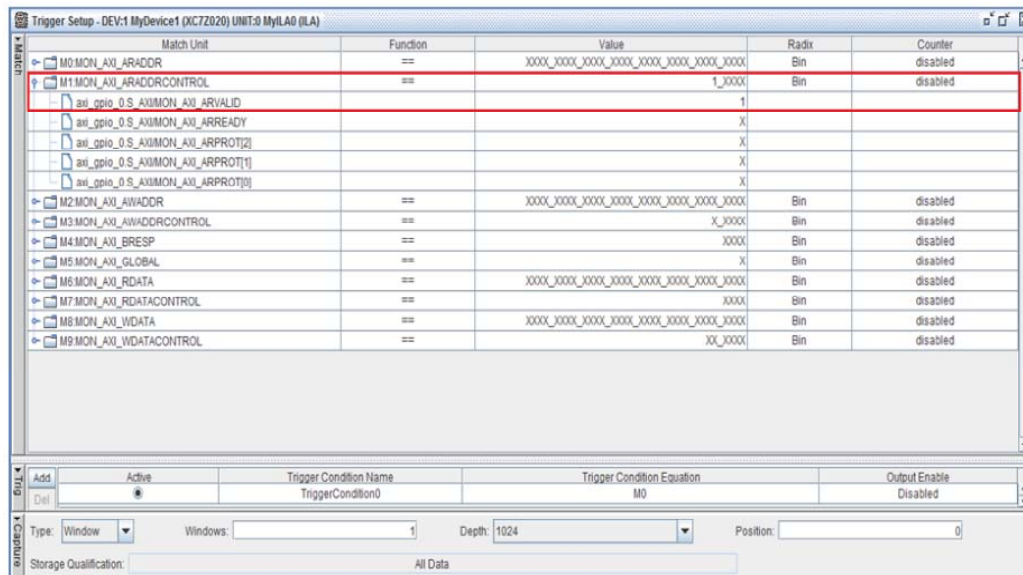


Figure 4-2: Trigger Setup Window, MON_AXI_AVALID Setting

- c. In the Trig section of the Trigger Setup window, click **M0** in the **Trigger Condition Equation** column.

The Trigger Condition dialog box opens.

- d. In the **Enable** column, unselect **M0** and select **M1**.

The trigger channel changes from M0 to M1; the ARVALID signal is on the M1 channel.

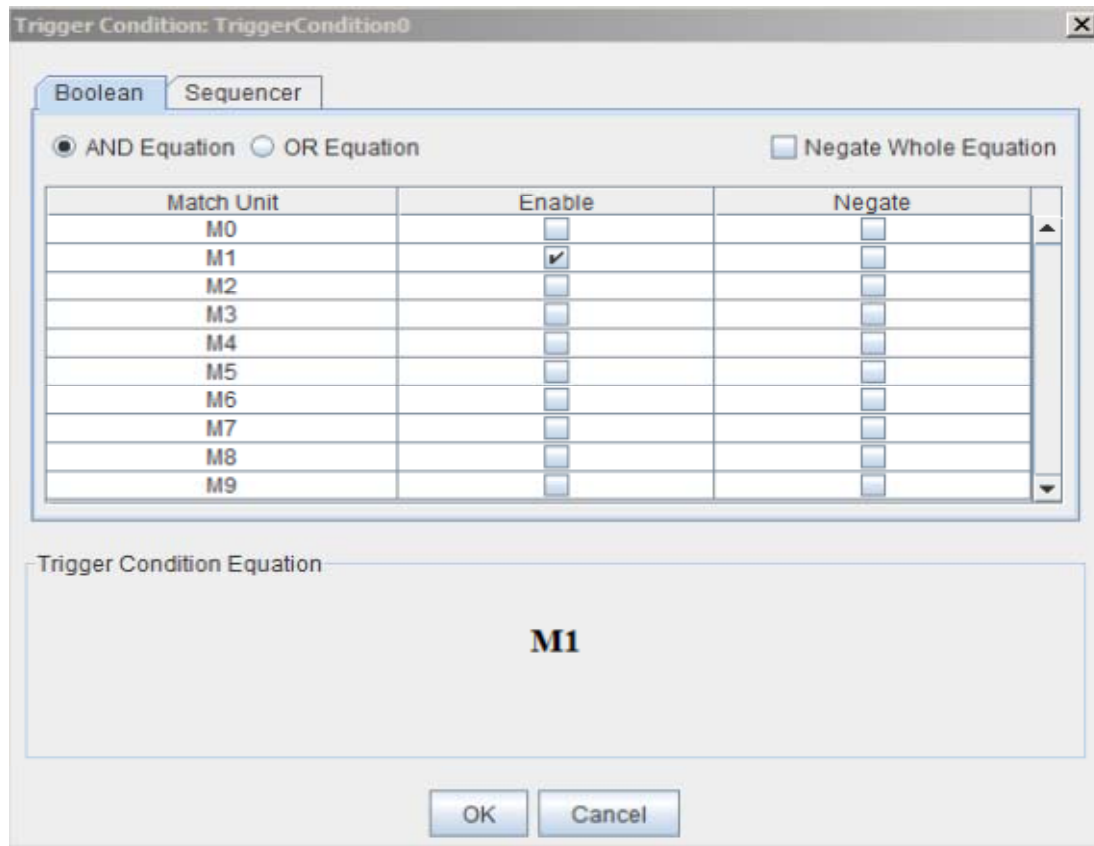


Figure 4-3: Trigger Condition Dialog Box

Click **OK**.

8. In the Capture section of the Trigger Setup window, change the **Position** setting from **0** to **512**.

The Trigger Point moves to the middle of the waveform as the sample depth changes to 1024.

9. Click the **Run** button .

ChipScope Analyzer waits for the trigger event.

10. Follow the instructions on the serial terminal to select the NORMAL AXI GPIO use case. This triggers the waveform.

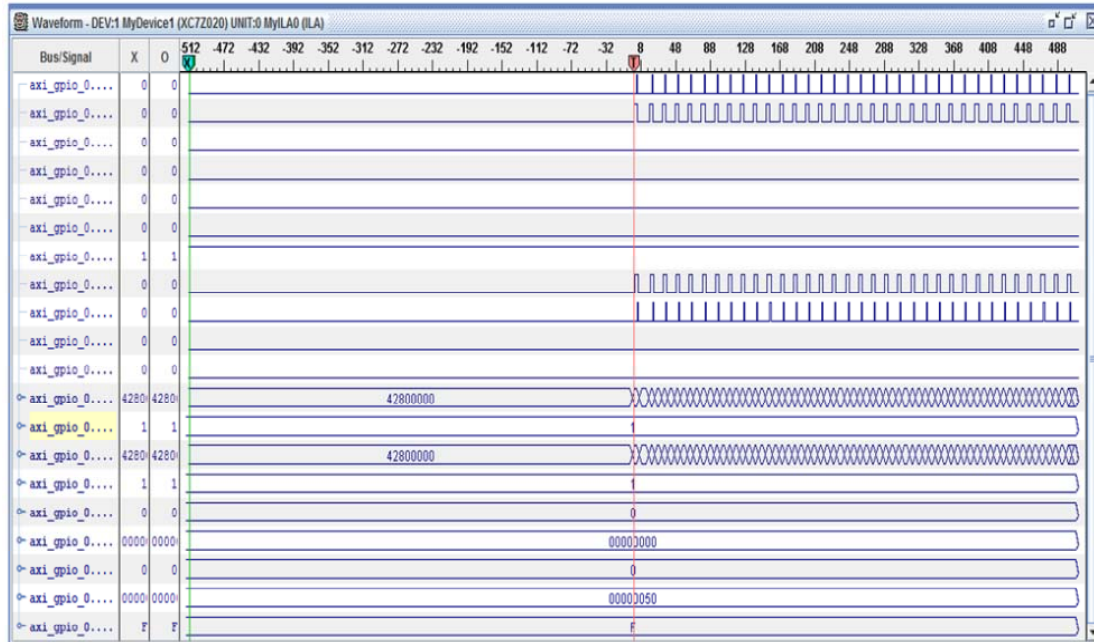


Figure 4-4: Waveforms

Appendix A

Application Software

A.1 About the Application Software

The system you designed in this guide requires application software for the execution on the board. This appendix describes the details about the application software.

The `main()` function in the application software is the entry point for the execution. This function includes initialization and the required settings for all peripherals connected in the system. It also has a selection procedure for the execution of the different use cases, such as AXI GPIO and PS GPIO using EMIO interface. You can select different use cases by following the instruction on the serial terminal.

A.2 Application Software Steps

Application Software comprises the following steps:

Initialize the AXI GPIO module.

1. Set a direction control for the AXI GPIO pin as an input pin, which is connected with BTNU push button on the board. The location is fixed via LOC constraint in the user constraint file (UCF) during system creation.
2. Initialize the AXI TIMER module with device ID 0.
3. Associate a timer callback function with AXI timer ISR.
4. This function is called every time the timer interrupt happens. This callback switches on the LED 'LD9' on the board and sets the interrupt flag.
5. The `main()` function uses the interrupt flag to halt execution, wait for timer interrupt to happen, and then restarts the execution.
6. Set the reset value of the timer, which is loaded to the timer during reset and timer starts.
7. Set timer options such as Interrupt mode and Auto Reload mode.
8. Initialize the PS section GPIO.

9. Set the PS section GPIO, channel 0, pin number 10 to the output pin, which is mapped to the MIO pin and physically connected to the LED 'LD9' on the board.
10. Set PS Section GPIO channel number 2 pin number 0 to input pin, which is mapped to PL side pin via the EMIO interface and physically connected to the BTNR push button switch.
11. Initialize Snoop control unit Global Interrupt controller. Also, register Timer interrupt routine to interrupt ID '91', register the exceptional handler, and enable the interrupt.
12. Execute a sequence in the loop to select between AXI GPIO or PS GPIO use case via serial terminal.

The software accepts your selection from the serial terminal and executes the procedure accordingly.

After the selection of the use case via the serial terminal, you must press a push button on the board as per the instruction on terminal. This action switches off the LED 'LD9', starts the timer, and tells the function to wait for the Timer interrupt to happen. After the Timer interrupt happens, LED 'LD9' switches ON and restarts execution.

- For more details about the API related to device drivers, refer to the *Zynq-7000 Software Developers Guide (UG821)*. ***Zynq-7000 Software Developers Guide (UG821)***:
http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_1/ug821-zynq-7000-swdev.pdf

A.3 Application Software Code

Below is the Application software for the system:

```
/*
 * Copyright (c) 2009 Xilinx, Inc. All rights reserved.
 *
 * Xilinx, Inc.
 * XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS" AS A
 * COURTESY TO YOU. BY PROVIDING THIS DESIGN, CODE, OR INFORMATION AS
 * ONE POSSIBLE IMPLEMENTATION OF THIS FEATURE, APPLICATION OR
 * STANDARD, XILINX IS MAKING NO REPRESENTATION THAT THIS IMPLEMENTATION
 * IS FREE FROM ANY CLAIMS OF INFRINGEMENT, AND YOU ARE RESPONSIBLE
 * FOR OBTAINING ANY RIGHTS YOU MAY REQUIRE FOR YOUR IMPLEMENTATION.
 * XILINX EXPRESSLY DISCLAIMS ANY WARRANTY WHATSOEVER WITH RESPECT TO
 * THE ADEQUACY OF THE IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO
 * ANY WARRANTIES OR REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE
 * FROM CLAIMS OF INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS FOR A PARTICULAR PURPOSE.
 */

/*
 * helloworld.c: simple test application
 */
#include <stdio.h>
#include "platform.h"
#include "xil_types.h"
#include "xgpio.h"
#include "xtmrctr.h"
#include "xparameters.h"
#include "xgpiops.h"
#include "xil_io.h"
#include "xil_exception.h"
#include "xscugic.h"
static XGpioPs psGpioInstancePtr;
extern XGpioPs_Config XGpioPs_ConfigTable[XPAR_XGPIOPS_NUM_INSTANCES];
```

© Copyright 2012 Xilinx

```
static int iPinNumber = 7; /*Led LD9 is connected to MIO pin 7*/
XScuGic InterruptController; /* Instance of the Interrupt Controller */
static XScuGic_Config *GicConfig; /* The configuration parameters of the
controller */
static int InterruptFlag;
void print(char *str);
extern char inbyte(void);

void Timer_InterruptHandler(void *data, u8 TmrCtrNumber)
{
    print("\r\n");
    print("\r\n");
    print("@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@r\n");
    print(" Inside Timer ISR \n \r ");
    XTmrCtr_Stop(data,TmrCtrNumber);
    // PS GPIO Writing
    print("LED 'LD9' Turned ON \r\n");
    XGpioPs_WritePin(&psGpioInstancePtr,iPinNumber,1);
    XTmrCtr_Reset(data,TmrCtrNumber);
    print(" Timer ISR Exit\n \n \r");
    print("@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@r\n");
    print("\r\n");
    print("\r\n");
    InterruptFlag = 1;
}

int SetUpInterruptSystem(XScuGic *XScuGicInstancePtr)
{
    /*
     * Connect the interrupt controller interrupt handler to the hardware
     * interrupt handling logic in the ARM processor.
     */
    Xil_ExceptionRegisterHandler(XIL_EXCEPTION_ID_INT,
        (Xil_ExceptionHandler) XScuGic_InterruptHandler,
        XScuGicInstancePtr);
    /*
     * Enable interrupts in the ARM
     */
    Xil_ExceptionEnable();
    return XST_SUCCESS;
}

int ScuGicInterrupt_Init(u16 DeviceId,XTmrCtr *TimerInstancePtr)
{
    int Status;
    /*
     * Initialize the interrupt controller driver so that it is ready to
     * use.
     */
    GicConfig = XScuGic_LookupConfig(DeviceId);
    if (NULL == GicConfig) {
        return XST_FAILURE;
    }
    Status = XScuGic_CfgInitialize(&InterruptController, GicConfig,
        GicConfig->CpuBaseAddress);
    if (Status != XST_SUCCESS) {
        return XST_FAILURE;
    }
    /*
     * Setup the Interrupt System
     */
    Status = SetUpInterruptSystem(&InterruptController);
    if (Status != XST_SUCCESS) {
        return XST_FAILURE;
    }
    /*
     * Connect a device driver handler that will be called when an
     * interrupt for the device occurs, the device driver handler performs
     * the specific interrupt processing for the device
     */
    Status = XScuGic_Connect(&InterruptController,
```

```

        XPAR_FABRIC_AXI_TIMER_0_INTERRUPT_INTR,
        (Xil_ExceptionHandler)XTmrCtr_InterruptHandler,
        (void *)TimerInstancePtr);
    if (Status != XST_SUCCESS) {
        return XST_FAILURE;
    }
    /*
     * Enable the interrupt for the device and then cause (simulate) an
     * interrupt so the handlers will be called
     */
    XScuGic_Enable(&InterruptController, XPAR_FABRIC_AXI_TIMER_0_INTERRUPT_INTR);
    return XST_SUCCESS;
}

int main()
{
    static XGpio GPIOInstance_Ptr;
    XGpioPs_Config*GpioConfigPtr;
    XTmrCtr TimerInstancePtr;
    int xStatus;
    u32 Readstatus=0,OldReadStatus=0;
    //u32 EffectiveAdress = 0xE000A000;
    int iPinNumberEMIO = 54;
    u32 uPinDirectionEMIO = 0x0;
    // Input Pin
    // Pin direction
    u32 uPinDirection = 0x1;
    int exit_flag,choice,internal_choice;
    init_platform();
    /* data = *(u32 *) (0x42800004);
       print("OK \n");
       data = *(u32 *) (0x41200004);
       print("OK-1 \n");
    */
    print("##### Application Starts #####\n\r");
    print("\r\n");
    //~~~~~
    //Step-1 :AXI GPIO Initialization
    //~~~~~
    xStatus = XGpio_Initialize(&GPIOInstance_Ptr,XPAR_AXI_GPIO_0_DEVICE_ID);
    if(XST_SUCCESS != xStatus)
        print("GPIO INIT FAILED\n\r");
    //~~~~~
    //Step-2 :AXI GPIO Set the Direction
    //~~~~~
    XGpio_SetDataDirection(&GPIOInstance_Ptr, 1,1);
    //~~~~~
    //Step-3 :AXI Timer Initialization
    //~~~~~
    xStatus = XTmrCtr_Initialize(&TimerInstancePtr,XPAR_AXI_TIMER_0_DEVICE_ID);
    if(XST_SUCCESS != xStatus)
        print("TIMER INIT FAILED \n\r");
    //~~~~~
    //Step-4 :Set Timer Handler
    //~~~~~
    XTmrCtr_SetHandler(&TimerInstancePtr,
        Timer_InterruptHandler,
        &TimerInstancePtr);
    //~~~~~
    //Step-5 :Setting timer Reset Value
    //~~~~~
    XTmrCtr_SetResetValue(&TimerInstancePtr,
        0, //Change with generic value
        0xf0000000);
    //~~~~~
    //Step-6 :Setting timer Option (Interrupt Mode And Auto Reload )
    //~~~~~
    XTmrCtr_SetOptions(&TimerInstancePtr,
        XPAR_AXI_TIMER_0_DEVICE_ID,
        (XTC_INT_MODE_OPTION | XTC_AUTO_RELOAD_OPTION ));
    //~~~~~

```

```

//Step-7 :PS GPIO Initialization
//~~~~~
GpioConfigPtr = XGpioPs_LookupConfig(XPAR_PS7_GPIO_0_DEVICE_ID);
if(GpioConfigPtr == NULL)
    return XST_FAILURE;
xStatus = XGpioPs_CfgInitialize(&psGpioInstancePtr,
    GpioConfigPtr,
    GpioConfigPtr->BaseAddr);
if(XST_SUCCESS != xStatus)
    print(" PS GPIO INIT FAILED \n\r");
//~~~~~
//Step-8 :PS GPIO pin setting to Output
//~~~~~
XGpioPs_SetDirectionPin(&psGpioInstancePtr, iPinNumber,uPinDirection);
XGpioPs_SetOutputEnablePin(&psGpioInstancePtr, iPinNumber,1);
//~~~~~
//Step-9 :EMIO PIN Setting to Input port
//~~~~~
XGpioPs_SetDirectionPin(&psGpioInstancePtr,
    iPinNumberEMIO,uPinDirectionEMIO);
XGpioPs_SetOutputEnablePin(&psGpioInstancePtr, iPinNumberEMIO,0);
//~~~~~
//Step-10 : SCUGIC interrupt controller Initialization
//Registration of the Timer ISR
//~~~~~
xStatus=
    ScuGicInterrupt_Init(XPAR_PS7_SCUGIC_0_DEVICE_ID,&TimerInstancePtr);
if(XST_SUCCESS != xStatus)
    print(" :( SCUGIC INIT FAILED \n\r");
//~~~~~
//Step-11 :User selection procedure to select and execute tests
//~~~~~
exit_flag = 0;
while(exit_flag != 1)
{
    print(" SELECT the Operation from the Below Menu \r\n");
    print("##### Menu Starts #####\r\n");
    print("Press '1' to use NORMAL GPIO as an input (BTNU switch)\r\n");
    print("Press '2' to use EMIO as an input (BTNR switch)\r\n");
    print("Press any other key to Exit\r\n");
    print("##### Menu Ends #####\r\n");
    choice = inbyte();
    printf("Selection : %c \r\n",choice);
    internal_choice = 1;
    switch(choice)
    {
        //~~~~~
        // Use case for AXI GPIO
        //~~~~~
        case '1':
            exit_flag = 0;
            print("Press Switch 'BTNU' push button on board \r\n");
            print(" \r\n");
            while(internal_choice != '0')
            {
                Readstatus = XGpio_DiscreteRead(&GPIOInstance_Ptr, 1);
                if(1== Readstatus && 0 == OldReadStatus )
                {
                    print("$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$\r\n");
                    print("BTNU PUSH Button pressed \n\r");
                    print("LED 'LD9' Turned OFF \r\n");
                    XGpioPs_WritePin(&psGpioInstancePtr,iPinNumber,0);
                    //Start Timer
                    XTmrCtr_Start(&TimerInstancePtr,0);
                    print("timer start \n\r");
                    //Wait For interrupt;
                    print("Wait for the Timer interrupt to tigger \r\n");
                    print("$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$\r\n");
                    print(" \r\n");
                    while(InterruptFlag != 1);
                    InterruptFlag = 0;
                }
            }
        }
    }
}

```

```
print(" #####\r\n ");
printf("Press '0' to go to Main Menu \n\r ");
printf("Press any other key to remain in AXI GPIO Test \n\r ");
printf("#####\r\n");
internal_choice = inbyte();
printf("Select = %c \r\n", internal_choice);
if(internal_choice != '0')
{
    printf("Press Switch 'BTNU' push button on board \r\n");
}
OldReadStatus = Readstatus;
}
printf("\r\n");
break;
case '2':
//~~~~~
//Usecase for PS GPIO
//~~~~~
exit_flag = 0;
printf("Press Switch 'BTNRR' push button on board \r\n");
while(internal_choice != '0')
{
    Readstatus = XGpioPs_ReadPin(&psGpioInstancePtr, iPinNumberEMIO);
    if(1== Readstatus && 0 == OldReadStatus )
    {
        print("$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$\\r\\n");
        print("BTNRR PUSH Button pressed \\r\\n");
        XGpioPs_WritePin(&psGpioInstancePtr,iPinNumber,0);
        //Start Timer
        XTmrCtr_Start(&TimerInstancePtr,0);
        print("timer start \\n\\r");
        //Wait For interrupt;
        print("Wait for the Timer interrupt to trigger \\r\\n");
        print("$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$\\r\\n");
        while(InterruptFlag != 1);
        InterruptFlag = 0;
        print("#####\\r\\n");
        printf("Press '0' to go to Main Menu \\n\\r ");
        printf("Press any other key to remain in EMIO Test \\n\\r ");
        printf("#####\\r\\n");
        internal_choice = inbyte();
        printf("Select = %c \\r\\n", internal_choice);
        if(internal_choice != '0')
        {
            printf("Press Switch 'BTNRR' push button on board \\r\\n");
        }
    }
    OldReadStatus = Readstatus;
}
printf("\r\n");
break;
default :
    exit_flag = 1;
    break;
}
}
return 0;
```

