Zynq-7000 All Programmable SoC: Concepts, Tools, and Techniques (CTT)

A Hands-On Guide to Effective Embedded System Design

UG873 (v14.1) May 31, 2012

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
04/24/2012	14.1	14.1 Release of Zynq™-7000 All Programmable SoC device support.
05/28/2012	14.1	Updated 14.1 version.
05/31/2012	14.1	Corrected LED "D23" to "DS23" on page 19 and page 47.

Table of Contents

Introduction

1.1 About this Guide

This document provides an introduction to using the Xilinx® ISE® Design Suite flow for using the Zynq™-7000 All Programmable SoC tools. The examples are targeted for the Xilinx ZC702 Rev C evaluation board and the tool version used in 14.1.

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 Extensible Processing Platform. 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.

The Xilinx Embedded Development Kit (EDK) is a suite of tools and intellectual property (IP) that enables you to design a complete embedded processor system for implementation in a Xilinx Field Programmable Gate Array (FPGA) device.

This guide describes the design flow for developing a custom Zynq-7000 All Programmable SoC based embedded processing system using EDK. It contains the following chapters:

- Chapter 1 (this chapter) provides a general overview.
- Chapter 2, Embedded System Design Using the Zynq Processing System, describes creation of a system with the Zynq Processing System (PS) and running a simple "Hello World" application.
- Chapter 3, Embedded System Design Using the Zynq Processing System and Programmable Logic, describes how to create a system using the Zynq Processing System (PS) and the Programmable Logic (PL, or "fabric") and how to use a simple application to exercise both the PS and PL.
- Chapter 4, Debugging with SDK and ChipScope provides debugging information from two perspectives: Software (using SDK Debug) and Hardware (using the ChipScope™ software).
- Chapter 5, Linux Booting and Application Debugging Using SDK provides information about booting the Linux OS on the Zynq™-7000 AP SoC board and application debugging.
- Appendix A, Application Software describes the details about the application software needed for the sample design used in this guide.
- Appendix B, Additional Resources provides links to additional resources related to 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. Specifications for a sample project 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

Additional documentation is listed in Appendix B, Additional Resources.

1.1.3 Training Labs

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

A link to training labs is available in Appendix B, Additional Resources.

1.2 How Zynq AP SoC and EDK Simplify Embedded Processor Design

Embedded systems are complex. Hardware and software portions of an embedded design are projects in themselves. Merging the two design components so that they function as one system creates additional challenges. Add an FPGA design project to the mix, and the situation has the potential to become very complicated.

The Zynq AP SoC solution reduces this complexity by offering an ARM Cortex A9 dual core as Hard IP and programmable logic along with it on a single SoC. It is the first of its kind in the market and has tremendous potential as a complete system.

To simplify the design process, Xilinx offers several sets of tools. It is a good idea to get to know the basic tool names, project file names, and acronyms for these tools. You can find EDK-specific terms in the Xilinx Glossary: http://www.xilinx.com/support/documentation/sw_manuals/glossary.pdf.

The Embedded Development Kit (EDK) is combination of Xilinx Platform Studio (XPS) and the Software Development Kit (SDK). It offers hardware and software application design, debug, and execution, and helps to take the design onto actual boards for verification and validation.

1.2.1 The ISE Design Suite, Embedded Edition

Xilinx offers a broad range of development system tools, collectively called the ISE Design Suite. For embedded system development, Xilinx offers the Embedded Edition of the ISE Design Suite. The Embedded Edition comprises:

- Integrated Software Environment (ISE)
- PlanAhead™ design analysis tool
- ChipScope Pro, which is useful for on-chip debugging of FPGA designs
- Embedded Development Kit (EDK).

EDK is also available with the ISE Design Suite: System Edition, which includes tools for DSP design.

For information on how to use the ISE tools for FPGA design, refer to the documentation available in Appendix B, Additional Resources.

1.2.2 The Embedded Development Kit

EDK is a suite of tools and IP that you can use to design a complete embedded processor system for implementation in a Xilinx FPGA device.

Xilinx Platform Studio

Xilinx Platform Studio (XPS) is the development environment used for designing the hardware portion of your embedded processor system. You can run XPS in batch mode or using the GUI, which is demonstrated in this guide.

Software Development Kit

The Software Development Kit (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 and might appear familiar to you or members of your design team. For more information about the Eclipse development environment, refer to http://www.eclipse.org.

Other EDK Components

Other EDK 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-A9MP processors in the Zynq Processing System
- Documentation
- Sample projects

1.3 How the ISE Tools Expedite the Design Process

The PlanAhead design and analysis tool is used to add design sources to your hardware. You can create this hardware system using XPS. XPS makes it very easy to add desired IPs to your existing design source and create connections for ports (such as clock and reset).

- You use XPS primarily for embedded processor hardware system development. Specification of the microprocessor, peripherals, and the interconnection of these components, along with their respective detailed configuration, takes place in XPS.
- You use SDK for software development. SDK is also available as a standalone application. It can be purchased and used without any other Xilinx tools installed on the machine on which it is loaded. SDK can be used to debug software applications.

The Zynq Processing System (PS) can be booted and made to run without anything being programmed inside FPGA programmable logic (PL). However, in order to use any soft IP in the fabric, or to bond out PS peripherals using EMIO, programming of the PL is required. You can do this step from within SDK.

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*. A link to this document is provided in Appendix B, Additional Resources.

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

1.4 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.

1.4.1 Installation Requirements: What You Need to Run EDK Tools

The PlanAhead Tool and EDK

The PlanAhead design tool and EDK are both included in the ISE Design Suite, Embedded Edition software. Be sure that the software, along with the latest update, is installed. Visit http://support.xilinx.com to confirm that you have the latest software versions.

Software Licensing

Xilinx software uses FLEXnet licensing. 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. If you are only evaluating the software, you can obtain an evaluation license.

For more information about licensing Xilinx software, refer to the *Xilinx Design Tools: Installation and Licensing Guide*. A link to this document is provided in Appendix B, Additional Resources.

1.4.2 Hardware Requirements for this Guide

This tutorial targets the Zynq ZC702 Rev C evaluation board.

Embedded System Design Using the Zynq Processing System

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

Zynq All Programmable SoC consists of ARM Cortex A9 hard IP and programmable logic. This offering can be used in two ways:

1. The Zynq PS can be used in a standalone mode, without attaching an additional IP to it from fabric.
2. IPs can be instantiated in fabric and attached to the Zynq PS. You can use this PS + PL combination to achieve complex and efficient design of a single SOC.

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 PS peripherals and available MIO connections meet the design requirements, no bitstream is required. This chapter guides you through creating one such design.

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.

1. Start the PlanAhead tool.
2. Select **Create New Project** to open the New Project wizard.

3. 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 **Boards**.
	Board	Select **Zynq-7 ZC702 Evaluation Board**.
New Project Summary	Project summary	Review the project summary before clicking **Finish** to create the project.

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.*

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

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

   The Add Sources wizard opens.

2. Select the **Add or Create Embedded Sources** option and click **Next**.
3. In the Add or Create Embedded Source window, click **Create Sub-Design**.
4. 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.

5. Click **Finish**.

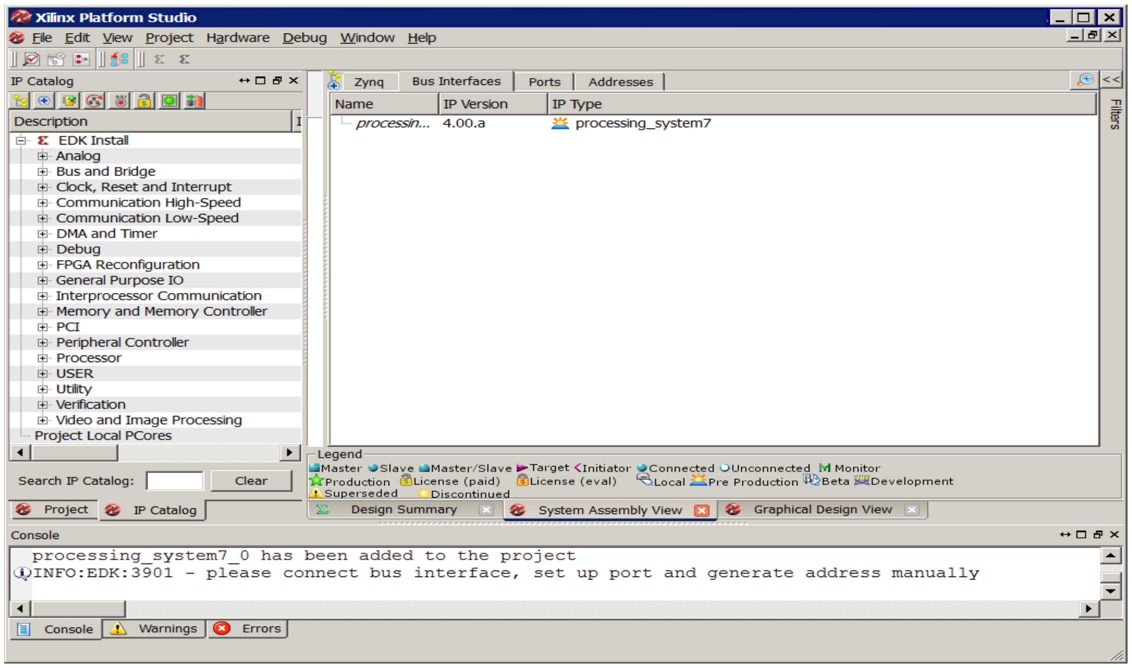
   XPS opens, and asks if you want to add Processing System7 to the system.

6. Click **Yes**.

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

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

7. Click the **Bus Interfaces** tab. Notice that processing_system7 was added.



Name	IP Version	IP Type
processin...	4.00.a	processing_system7

```
processing_system7_0 has been added to the project
INFO:EDK:3901 - please connect bus interface, set up port and generate address manually
```

Figure 2-1: **XPS System Assembly View**

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

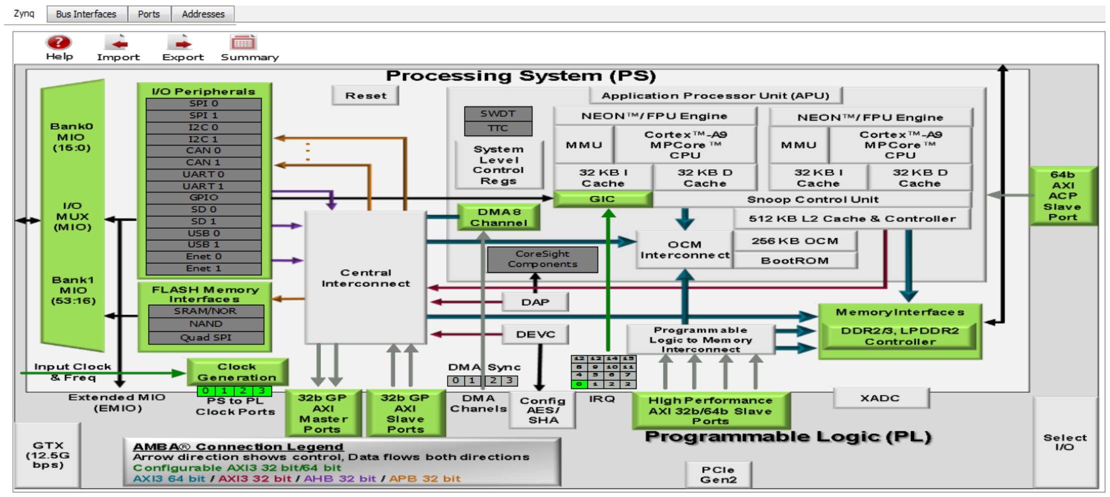


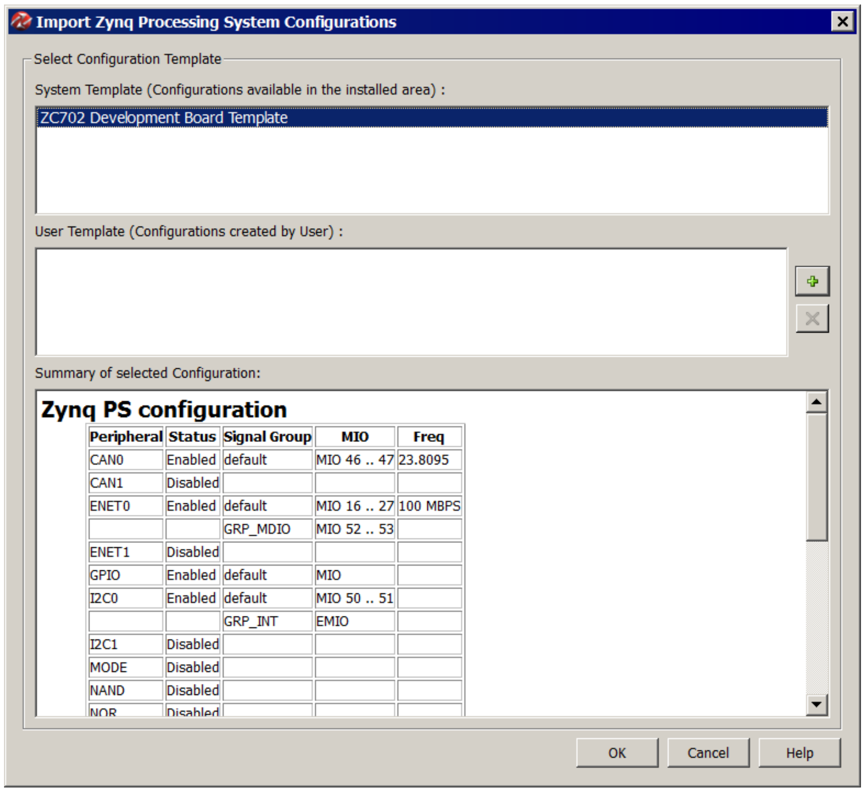
Figure 2-2: **Zynq Processing System**

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

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

   The Import Zynq Configurations dialog box opens.

10. 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 ZC702 board.



Import Zynq Processing System Configurations
Select Configuration Template
System Template (Configurations available in the installed area) :
ZC702 Development Board Template
User Template (Configurations created by User) :
Summary of selected Configuration:
OK
Cancel
Help

Zynq PS configuration

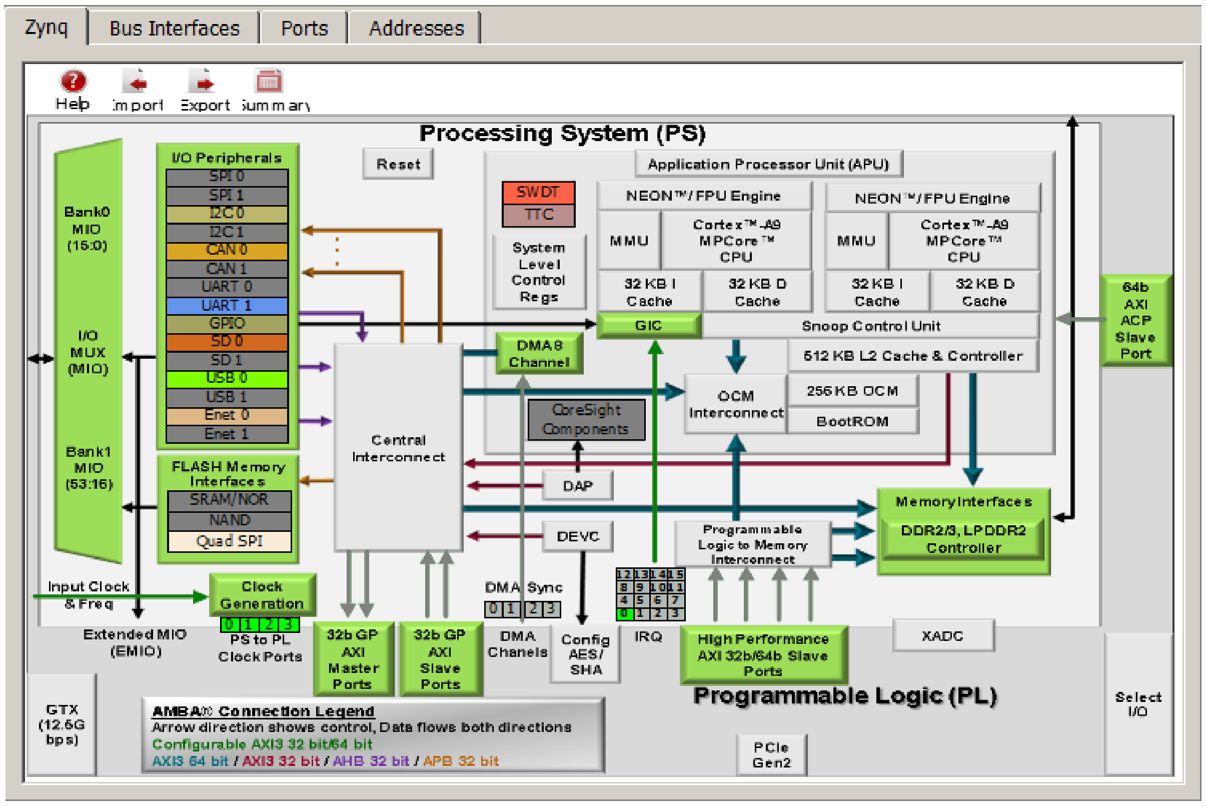
Peripheral	Status	Signal Group	MIO	Freq
CAN0	Enabled	default	MIO 46 .. 47	23.8095
CAN1	Disabled			
ENET0	Enabled	default	MIO 16 .. 27	100 MBPS
		GRP_MDIO	MIO 52 .. 53	
ENET1	Disabled			
GPIO	Enabled	default	MIO	
I2C0	Enabled	default	MIO 50 .. 51	
		GRP_INT	EMIO	
I2C1	Disabled			
MODE	Disabled			
NAND	Disabled			
NOR	Disabled			

Figure 2-3: **Import Zynq Configurations Dialog Box**

11. Click **OK**.

12. In the confirmation window that opens to verify that the Zynq MIO Configuration and Design will be updated, click **Yes**.

13. Note the change to the Zynq block diagram. The I/O Peripherals become active.



AMBA® Connection Legend
Arrow direction shows control, Data flows both directions
Configurable AXI3 32 bit/64 bit
AXI3 64 bit / AXI3 32 bit / AHB 32 bit / APB 32 bit

Figure 2-4: **Updated Zynq Block Diagram**

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

    Many peripherals are now enabled in the Processing System with some MIO pins assigned to them as per the board layout of the ZC702 board. For example, UART1 is enabled and UART0 is disabled. This is because UART1 is connected to the USB - UART connector through UART to the USB converter chip on the ZC702 board.

15. Close the Zynq PS MIO Configurations window.
16. 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, right-click **system(system.xmp)** and select **Create Top HDL**.

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

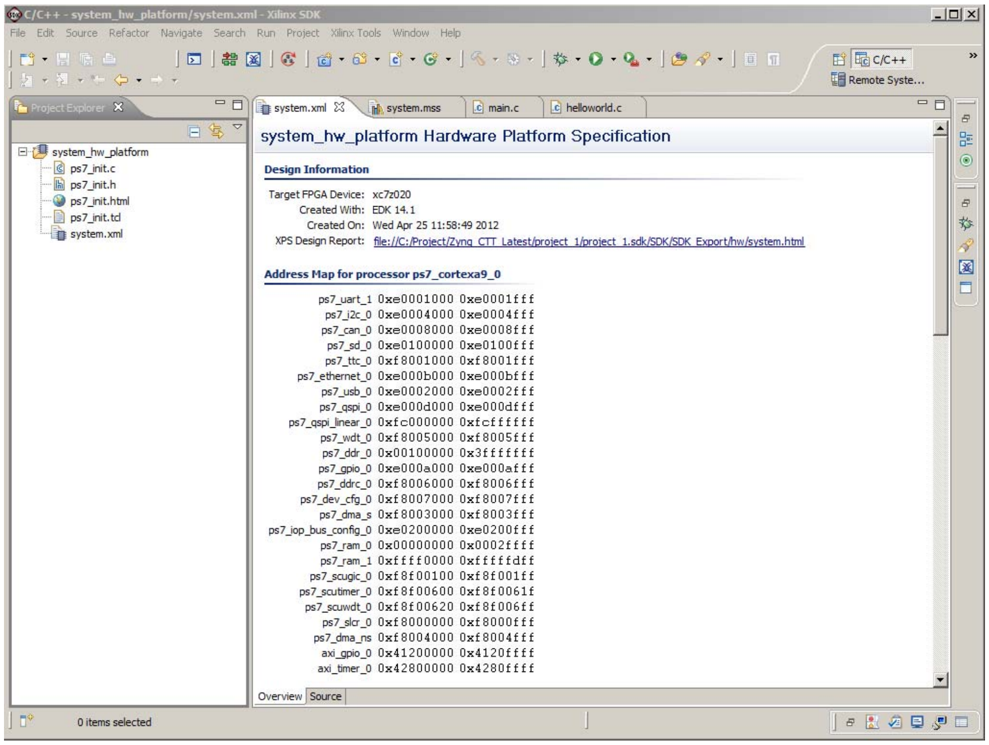
2. In the PlanAhead tool, Select **File > Export > Export Hardware**.

   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. The system.xml tab shows the address map for the entire Processing System.



system_hw_platform Hardware Platform Specification

Design Information

Target FPGA Device:	xc7z020
Created With:	EDK 14.1
Created On:	Wed Apr 25 11:58:49 2012
XPS Design Report:	file://C:/Project/Zynq_CTT_Latest/project_1/project_1.sdk/SDK/SDK_Export/hw/system.html

Address Map for processor ps7_cortexa9_0

```
          ps7_uart_1 0xe0001000 0xe0001fff
           ps7_i2c_0 0xe0004000 0xe0004fff
           ps7_can_0 0xe0008000 0xe0008fff
            ps7_sd_0 0xe0100000 0xe0100fff
           ps7_ttc_0 0xf8001000 0xf8001fff
      ps7_ethernet_0 0xe000b000 0xe000bfff
           ps7_usb_0 0xe0002000 0xe0002fff
          ps7_qspi_0 0xe000d000 0xe000dfff
   ps7_qspi_linear_0 0xfc000000 0xfcffffff
           ps7_wdt_0 0xf8005000 0xf8005fff
           ps7_ddr_0 0x00100000 0x3fffffff
          ps7_gpio_0 0xe000a000 0xe000afff
          ps7_ddrc_0 0xf8006000 0xf8006fff
       ps7_dev_cfg_0 0xf8007000 0xf8007fff
           ps7_dma_s 0xf8003000 0xf8003fff
ps7_iop_bus_config_0 0xe0200000 0xe0200fff
           ps7_ram_0 0x00000000 0x0002ffff
           ps7_ram_1 0xffff0000 0xfffffdff
        ps7_scugic_0 0xf8f00100 0xf8f001ff
      ps7_scutimer_0 0xf8f00600 0xf8f0061f
        ps7_scuwdt_0 0xf8f00620 0xf8f006ff
          ps7_slcr_0 0xf8000000 0xf8000fff
          ps7_dma_ns 0xf8004000 0xf8004fff
          axi_gpio_0 0x41200000 0x4120ffff
         axi_timer_0 0x42800000 0x4280ffff
```

Figure 2-5: **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 exported to SDK. They are `ps7_init.c`, `ps7_init.h`, `ps7_init.tcl`, and `ps7_init.html`.

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 ZC702 evaluation board.

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 power cable to the board.
2. Connect a Xilinx Platform cable USB II cable between the Windows Host machine and the Target board.
3. Connect a USB cable to connector J17 on the target board with the Windows Host machine. This is used for USB to serial transfer.
4. Power on the ZC702 board using the switch indicated in Figure 2-6.

IMPORTANT: *Ensure that jumpers J27 and J28 are placed on the side farther from the SD card slot and the rest of the jumpers in this line are placed towards the SD card slot.*

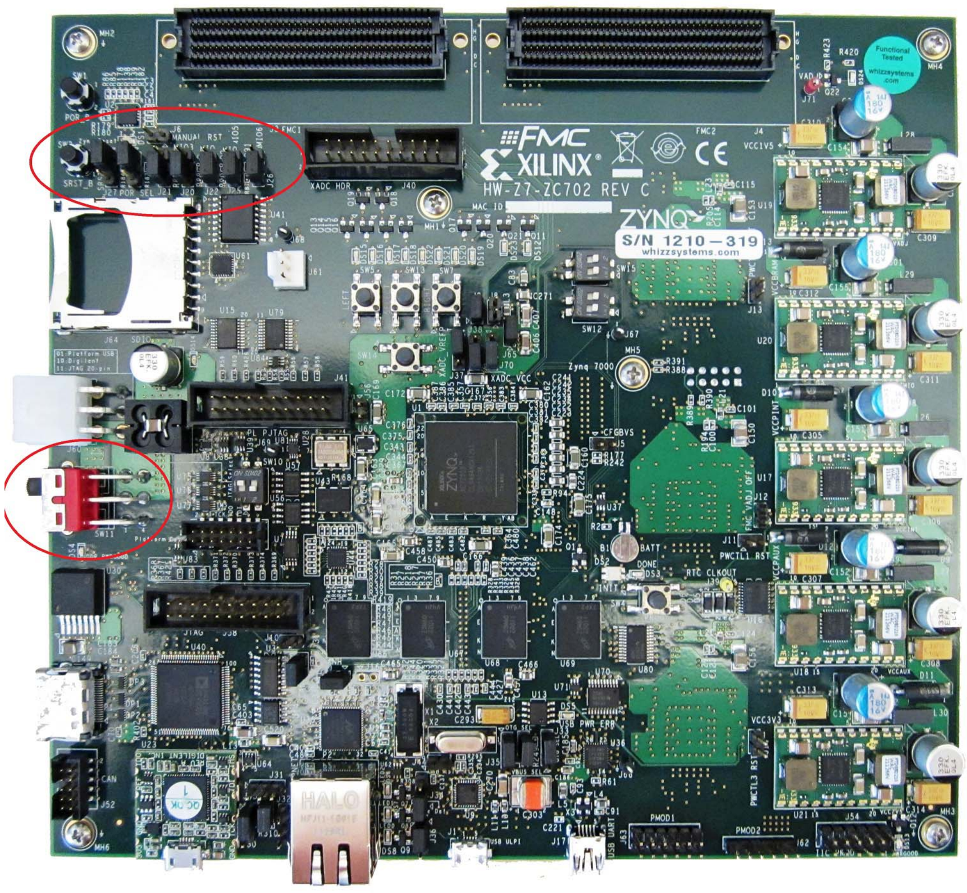


Figure 2-6: **ZC702 Board Power Switch**

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

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

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.
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`.

    The configurations associated with the application are pre-populated in the Main tab of the launch configurations.

14. Click the **Device Initialization** tab in the launch configurations and check the settings here.

    Notice that there is a configuration path to the initialization TCL file. The path of `ps7_init.tcl` is mentioned here. This file was exported when you exported your design to SDK; it contains the initialization information for the processing system.

15. 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.
16. Click **Run**.
17. "Hello World" appears on the serial communication utility.

Note: There was no bitstream download required for the above software application to be executed on the Zynq evaluation board. 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.

What Just Happened?

The application software sent the “Hello World” string to the UART1 peripheral of the PS section.

From UART1, the “Hello world” string goes, byte by byte, to the serial terminal application running on the host machine, which displays it as a string.

Associated Training Lab

The corresponding lab course for this Test Drive is *EDK: Adding and Downloading Software*. In this lab, you’ll use the SDK tools to create a software board support package and sample application. You’ll then configure the device and download the application to test.

A link to training labs is available in Appendix B, Additional Resources.

2.1.4 Additional Information

Board Support Package

The board support package (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 OS

Standalone is a simple, low-level software layer. It provides access to basic processor features such as caches, interrupts, and exceptions, as well as the basic processor features of a hosted environment. 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 the Standalone OS.

Embedded System Design Using the Zynq Processing System and Programmable Logic

One of the unique features of using the Zynq™ processor All Programmable SoC as an embedded design platform is in using the Zynq Processing System (PS) for its ARM Cortex A9 dual core processing system as well as Programmable Logic (PL) available on it.

In this chapter we will be creating a design with:

- AXI GPIO and AXI Timer with interrupt from fabric to PS section
- ChipScope™ IP instantiated in the PL
- Zynq PS GPIO pin connected to the PL side pin via the 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 keep referring to the material in it many times in this chapter.

3.1 Adding IPs in Fabric to Zynq PS

There is no restriction on the complexity of an IP that can be added in fabric 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 the proof of concept.

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

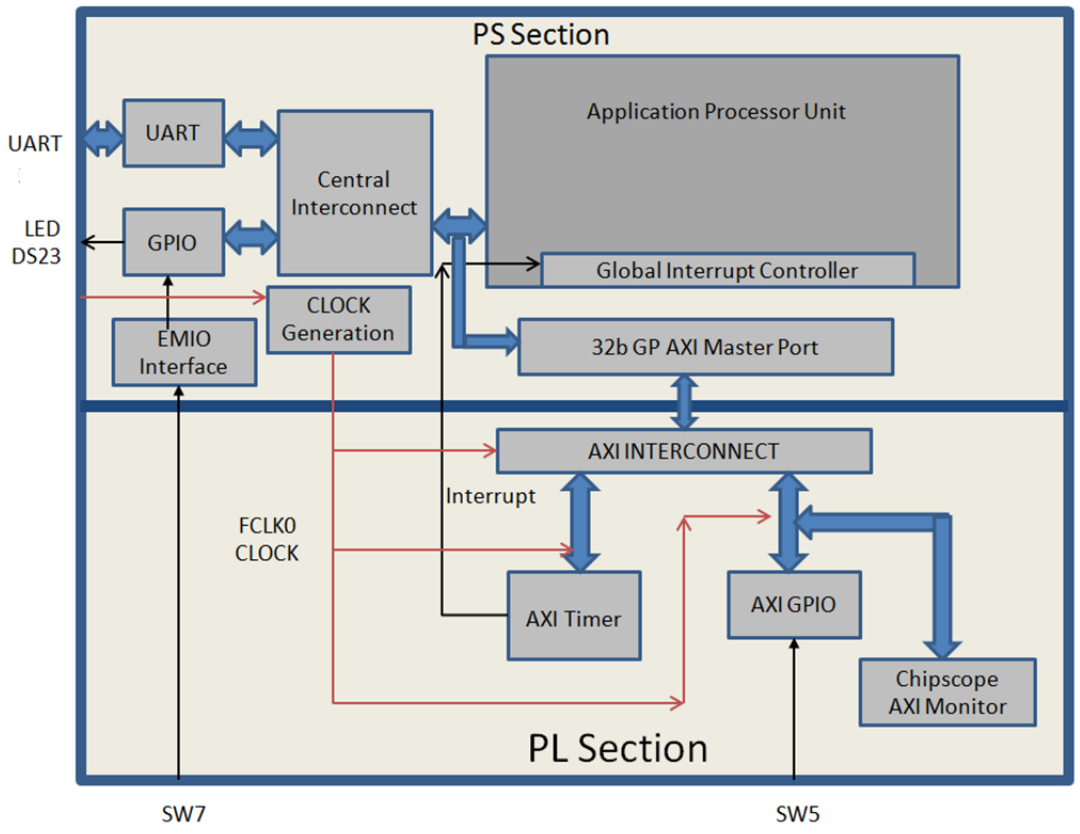


Figure 3-1: **Block Diagram**

This system covers the following connections:

1. Fabric-side AXI GPIO has only 1 bit channel width and it is connected to the push-button switch 'SW5' on theZC702 board.
2. PS section GPIO also has 1 bit interface routed to fabric pin via EMIO interface and connected to the push-button switch 'SW7' on the board.
3. In the PS section another 1 bit GPIO is connected to the LED 'DS23' on board which is on MIO port.
4. AXI timer interrupt is connected from fabric to the PS section interrupt controller. The timer starts when you press any of the selected push buttons on the board. After the timer expires, the timer interrupt is triggered.

You will write the application software code. When you run the code, a message appears in the serial terminal and asks you to select the push button switch to use on the board (either `SW7` or `SW5`). It then waits for you to press the specified push button. When the button is pressed, the timer starts automatically, switches OFF LED `DS23`, and waits for the timer interrupt to happen. After the Timer Interrupt, LED `DS23` switches ON and execution starts again and waits for you to again select the push button switch in the serial terminal.

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! Checking the Functionality of the IPs Instantiated in the Fabric

In this test drive, you'll check the functionality of the AXI GPIO, AXI Timer with interrupt instantiated in fabric 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 Fabric 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.

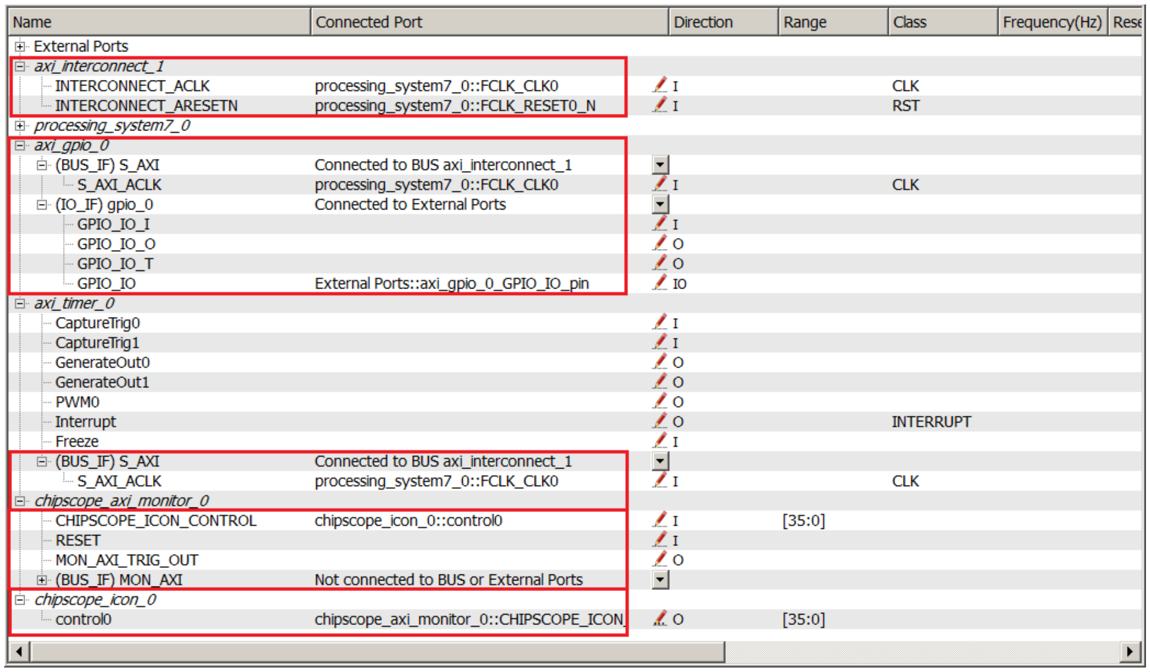
12. Click **OK**.

    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_system7_0 : FCLK_CLK0
	INTERCONNECT_ARESETN	processing_system7_0::FCLK_RESET0_N
axi_gpio_0	(BUS_IF) S_AXI::S_AXI_ACLK	processing_system7_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_system7_0 : FCLK_CLK0
Chipscope_axi_monitor_0	CHIPSCOPE_ICON_CONTROL	Chipscope_icon_0 ::control0
	(BUS_IF) MON_AXI:: MON_AXI_ACLK	processing_system7_0 : FCLK_CLK0
Chipscope_icon_0	Control0	Chipscope_axi_monitor0::CHIPSCOPE_ICON _CONTROL

Your Ports tab should be similar to Figure 3-2.

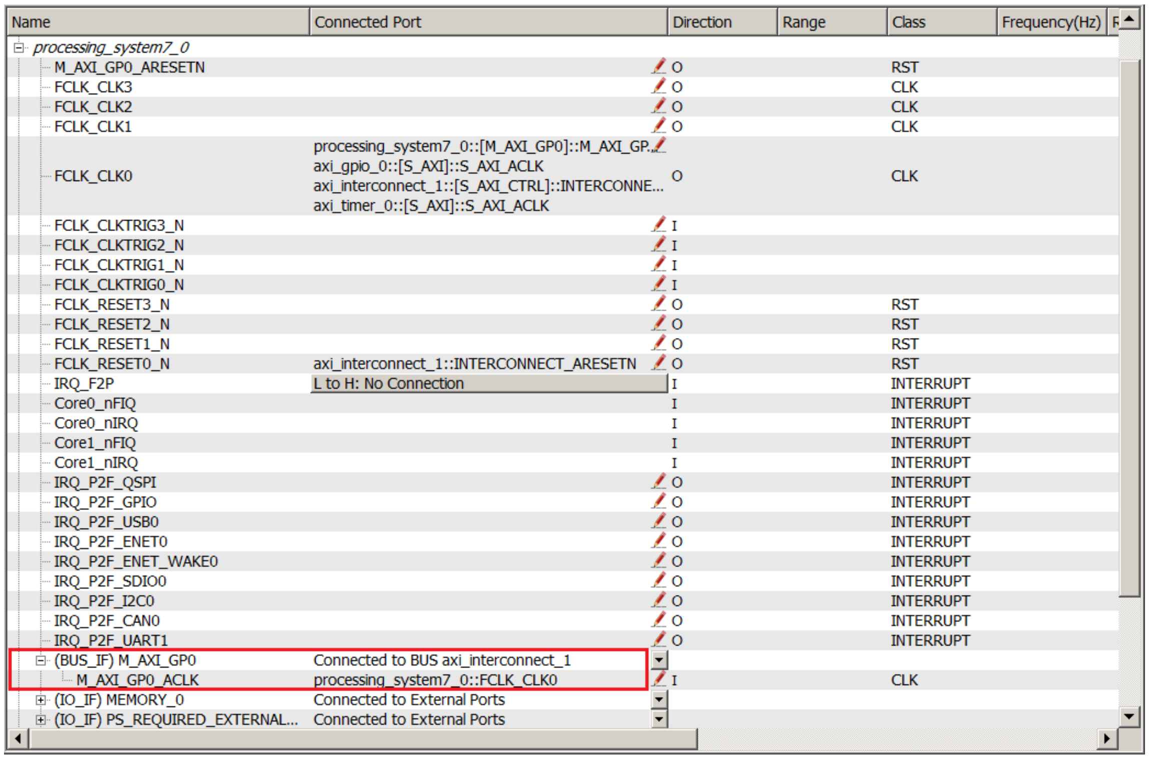


Name	Connected Port	Direction	Range	Class	Frequency(Hz)	Rese
External Ports						
axi_interconnect_1						
INTERCONNECT_ACLK	processing_system7_0::FCLK_CLK0	I		CLK		
INTERCONNECT_ARESETN	processing_system7_0::FCLK_RESET0_N	I		RST		
processing_system7_0						
axi_gpio_0						
(BUS_IF) S_AXI	Connected 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_I		I				
GPIO_IO_O		O				
GPIO_IO_T		O				
GPIO_IO	External Ports::axi_gpio_0_GPIO_IO_pin	IO				
axi_timer_0						
CaptureTrig0		I				
CaptureTrig1		I				
GenerateOut0		O				
GenerateOut1		O				
PWM0		O				
Interrupt		O		INTERRUPT		
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	Not connected to BUS or External Ports					
chipscope_icon_0						
control0	chipscope_axi_monitor_0::CHIPSCOPE_ICON	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.

IP	Port	Connection
Processing_system7_0	(BUS_IF) M_AXI_GP0:: M_AXI_GPO_ACLK	processing_system7_0 :: FCLK_CLK0



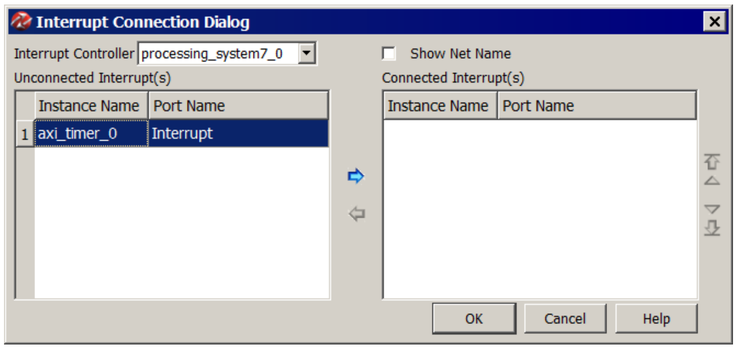
Name	Connected Port	Direction	Range	Class	Frequency(Hz)
processing_system7_0					
M_AXI_GP0_ARESETN		O		RST	
FCLK_CLK3		O		CLK	
FCLK_CLK2		O		CLK	
FCLK_CLK1		O		CLK	
FCLK_CLK0	processing_system7_0::[M_AXI_GP0]::M_AXI_GP... axi_gpio_0::[S_AXI]::S_AXI_ACLK axi_interconnect_1::[S_AXI_CTRL]::INTERCONNE... axi_timer_0::[S_AXI]::S_AXI_ACLK	O		CLK	
FCLK_CLKTRIG3_N		I			
FCLK_CLKTRIG2_N		I			
FCLK_CLKTRIG1_N		I			
FCLK_CLKTRIG0_N		I			
FCLK_RESET3_N		O		RST	
FCLK_RESET2_N		O		RST	
FCLK_RESET1_N		O		RST	
FCLK_RESET0_N	axi_interconnect_1::INTERCONNECT_ARESETN	O		RST	
IRQ_F2P	L to H: No Connection	I		INTERRUPT	
Core0_nFIQ		I		INTERRUPT	
Core0_nIRQ		I		INTERRUPT	
Core1_nFIQ		I		INTERRUPT	
Core1_nIRQ		I		INTERRUPT	
IRQ_P2F_QSPI		O		INTERRUPT	
IRQ_P2F_GPIO		O		INTERRUPT	
IRQ_P2F_USB0		O		INTERRUPT	
IRQ_P2F_ENET0		O		INTERRUPT	
IRQ_P2F_ENET_WAKE0		O		INTERRUPT	
IRQ_P2F_SDIO0		O		INTERRUPT	
IRQ_P2F_I2C0		O		INTERRUPT	
IRQ_P2F_CAN0		O		INTERRUPT	
IRQ_P2F_UART1		O		INTERRUPT	
(BUS_IF) M_AXI_GP0	Connected to BUS axi_interconnect_1				
M_AXI_GP0_ACLK	processing_system7_0::FCLK_CLK0	I		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 fabric side to the PS side interrupt controller by doing the following:

    a. In the Connected Port column of `Processing_System7_0`, click **L to H: No Connection**.

       The Interrupt Connection dialog box opens.

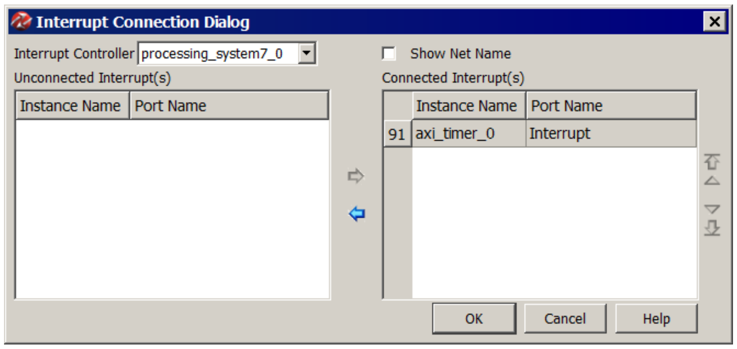


	Instance Name	Port Name
1	axi_timer_0	Interrupt

Figure 3-4: **Interrupt Connection Dialog Box**

    b. 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 displays the axi_timer_0 interrupt instance connected with Interrupt ID 91.

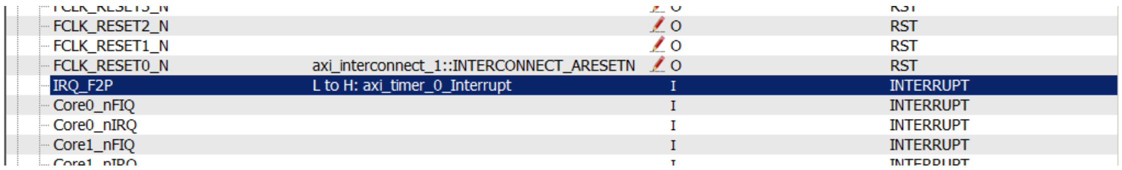


	Instance Name	Port Name
91	axi_timer_0	Interrupt

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

c. Click **OK**.

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

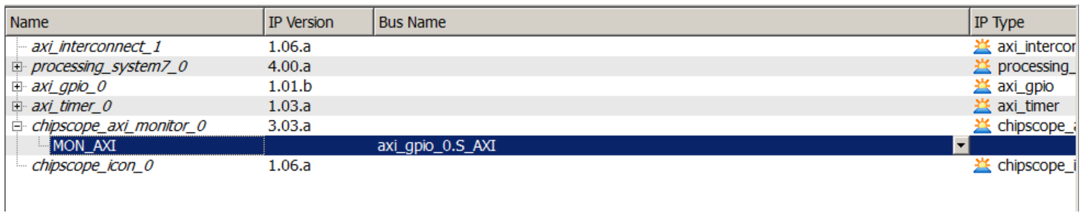


FCLK_RESET2_N		O	RST
FCLK_RESET1_N		O	RST
FCLK_RESET0_N	axi_interconnect_1::INTERCONNECT_ARESETN	O	RST
IRQ_F2P	L to H: axi_timer_0_Interrupt	I	INTERRUPT
Core0_nFIQ		I	INTERRUPT
Core0_nIRQ		I	INTERRUPT
Core1_nFIQ		I	INTERRUPT

Figure 3-6: **Timer Interrupt Connected on the Fabric 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_intercor
processing_system7_0	4.00.a		processing_
axi_gpio_0	1.01.b		axi_gpio
axi_timer_0	1.03.a		axi_timer
chipscope_axi_monitor_0	3.03.a		chipscope_
MON_AXI		axi_gpio_0.S_AXI	
chipscope_icon_0	1.06.a		chipscope_i

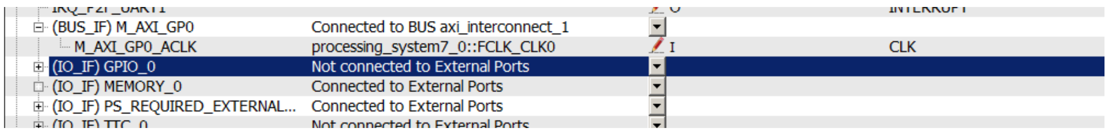
Figure 3-7: **Connected chipscope_axi_monitor**

20. Route the PS section GPIO to the PL side pad using the EMIO interface by doing the following:
    a. In the XPS System Assembly View, click the **Zynq** tab.
    b. Click the green **32b GP AXI Master Ports** button to open the XPS Core Config dialog box.
    c. In the **User** tab, expand the **General** item.
    d. Click to select the **Enable GPIO on EMIO Interface** check box.

TIP: *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.

    e. Change the GPIO width to **1** and click **OK**.
    f. 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.



```
(BUS_IF) M_AXI_GP0               Connected to BUS axi_interconnect_1
  M_AXI_GP0_ACLK                 processing_system7_0::FCLK_CLK0        I      CLK
(IO_IF) GPIO_0                   Not connected to External Ports
(IO_IF) MEMORY_0                 Connected to External Ports
(IO_IF) PS_REQUIRED_EXTERNAL...  Connected to External Ports
```

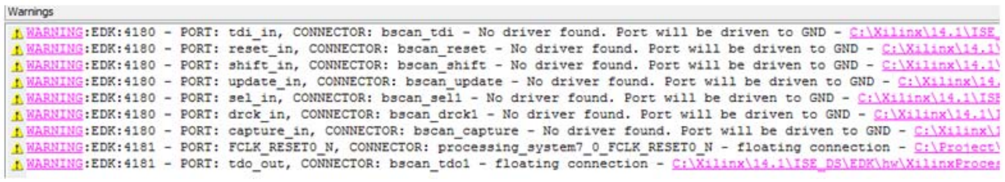
Figure 3-8: **GPIO Port Not Connected to External Ports**

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 Design Rule Check. Ensure there are no errors in the console.

    Note: If there are errors, double-check the steps you followed.



```
Warnings
WARNING:EDK:4180 - PORT: tdi_in, CONNECTOR: bscan_tdi - No driver found. Port will be driven to GND - C:\Xilinx\14.1\ISE_
WARNING:EDK:4180 - PORT: reset_in, CONNECTOR: bscan_reset - No driver found. Port will be driven to GND - C:\Xilinx\14.1\
WARNING:EDK:4180 - PORT: shift_in, CONNECTOR: bscan_shift - No driver found. Port will be driven to GND - C:\Xilinx\14.1\
WARNING:EDK:4180 - PORT: update_in, CONNECTOR: bscan_update - No driver found. Port will be driven to GND - C:\Xilinx\14.
WARNING:EDK:4180 - PORT: sel_in, CONNECTOR: bscan_sel1 - No driver found. Port will be driven to GND - C:\Xilinx\14.1\ISE
WARNING:EDK:4180 - PORT: drck_in, CONNECTOR: bscan_drck1 - No driver found. Port will be driven to GND - C:\Xilinx\14.1\I
WARNING:EDK:4180 - PORT: capture_in, CONNECTOR: bscan_capture - No driver found. Port will be driven to GND - C:\Xilinx\I
WARNING:EDK:4181 - PORT: FCLK_RESET0_N, CONNECTOR: processing_system7_0_FCLK_RESET0_N - floating connection - C:\Project\
WARNING:EDK:4181 - PORT: tdo_out, CONNECTOR: bscan_tdo1 - floating connection - C:\Xilinx\14.1\ISE_DS\EDK\hw\XilinxProce
```

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`.

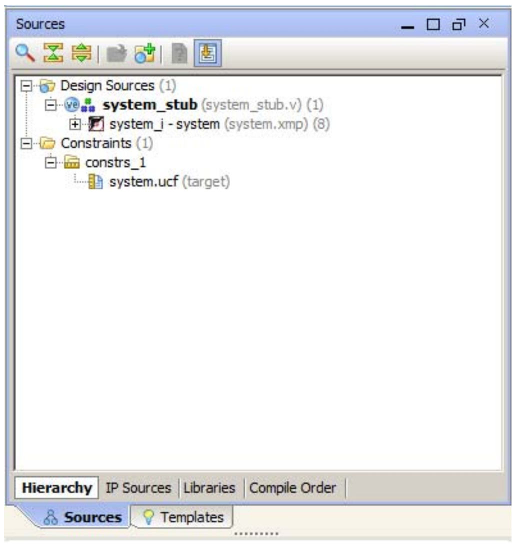


Figure 3-10: **system.ucf File Added**

30. Type the following text in the UCF file:

```
# Connect to Push Button "SW5"
NET axi_gpio_0_GPIO_IO_pin IOSTANDARD=LVCMOS25 | LOC=G19;
# Connect to Push Button "SW7"
NET processing_system7_0_GPIO_pin IOSTANDARD=LVCMOS25 | LOC=F19;
```

The following settings are made:

- The LOC constraint for NET "axi_gpio_0_IO_pin" connects the AXI GPIO pin to the G19 pin of the PL section and physically connects it to the SW5 push button on the board.
- The LOC constraint for NET "processing_system7_0 GPIO pin" connects the PS section GPIO to the F19 pin of the PL section and physically connects it to the SW7 push button on the board.
- The `IOSTANDARD=LVCMOS25` constraint sets both pins to LVCMOS 2.5V I/O standard.

31. Save all modified files.
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 Fabric, this will also be exported to SDK.

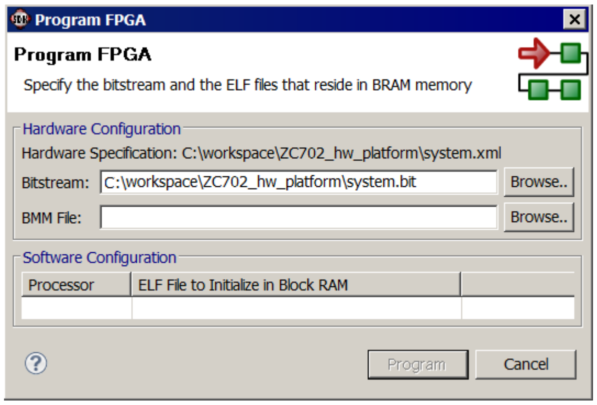
Associated Training Lab

The corresponding lab course for this Test Drive is *SDK: Basic System Implementation*. In this lab, you'll begin with the Processing System Configuration wizard (Zynq AP SoC) to create a hardware design. You'll then specify a basic software platform and add a software application to the system.

A link to training labs is available in Appendix B, Additional Resources.

3.1.2 Take a Test Drive! Working with SDK

1. SDK launches with the "Hello World" project you created with the Standalone PS in Chapter 2.
2. Select **Project > Clean** to clean and build the project again.
3. Open the `helloworld.c` file and modify the application software code. Refer to Appendix A, Application Software for the application software details.
4. Open the serial communication utility with baud rate set to **115200**.
5. Connect the board.
6. Because you have a bitstream for the PL Fabric, you must download the bitstream. To do this, select **Xilinx Tools > Program FPGA**. The Program FPGA dialog box, shown in Figure 3-11, opens. It displays the bitstream exported from PlanAhead.


Program FPGA

Program FPGA

Specify the bitstream and the ELF files that reside in BRAM memory

Hardware Configuration

Hardware Specification: C:\workspace\ZC702_hw_platform\system.xml

Bitstream: C:\workspace\ZC702_hw_platform\system.bit

BMM File:

Software Configuration

Processor	ELF File to Initialize in Block RAM

Figure 3-11: **Program FPGA Dialog Box**

7. Click **Program** to download the bitstream and program the PL Fabric.
8. Run the project similar to the steps in Take a Test Drive! Running the "Hello World" Application, page 16.
9. In the system, the AXI GPIO pin is connected to push button SW5 on the board, and the PS section GPIO pin is connected to push button SW7 on the board via an EMIO interface.
10. Follow the instructions printed on the serial terminal to run the application.

Debugging with SDK and ChipScope

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.

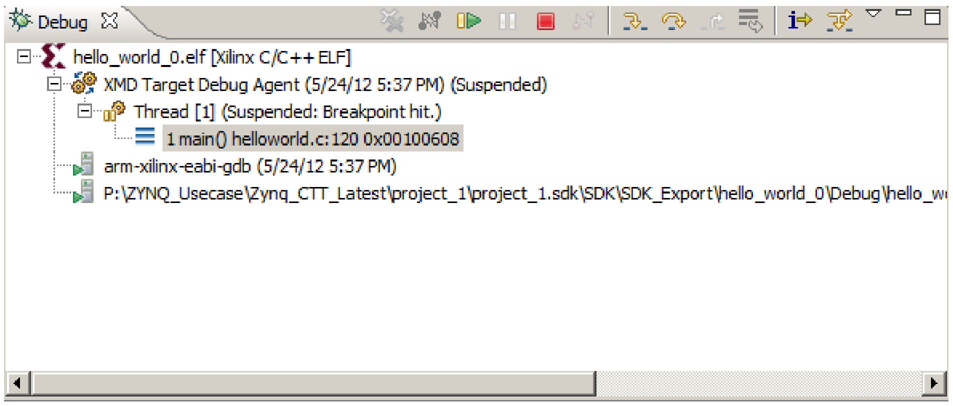
4.1 Take a Test Drive! Debugging with Software Using SDK

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**.

   A dialog box appears with a question about the reset properties of your system.
3. Click **OK.**

   Another dialog box appears to notify you that this kind of launch is configured to open the Debug perspective when it suspends.
4. Click **Yes**. The Debug Perspective opens.



```
hello_world_0.elf [Xilinx C/C++ ELF]
  XMD Target Debug Agent (5/24/12 5:37 PM) (Suspended)
    Thread [1] (Suspended: Breakpoint hit.)
      1 main() helloworld.c:120 0x00100608
  arm-xilinx-eabi-gdb (5/24/12 5:37 PM)
  P:\ZYNQ_Usecase\Zynq_CTT_Latest\project_1\project_1.sdk\SDK\SDK_Export\hello_world_0\Debug\hello_w
```

Figure 4-1: **Debug Perspective Suspended**

Note: The addresses shown on this page might be slightly different from the addresses shown on your system.

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**.

5. 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.

   Note: If the Breakpoints window is not visible, select **Window > Show View > Breakpoints**.
6. 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.
7. Select **Run > Step Into** to step into the `init_platform ()` routine.

   Program execution suspends at location 0x00100C44. The call stack is now two level deep.
8. 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.
9. Re-run your code several times. Experiment with single-stepping, examining memory, breakpoints, modifying code, and adding print statements. Try adding and moving views.
10. Close SDK.

Associated Training Lab

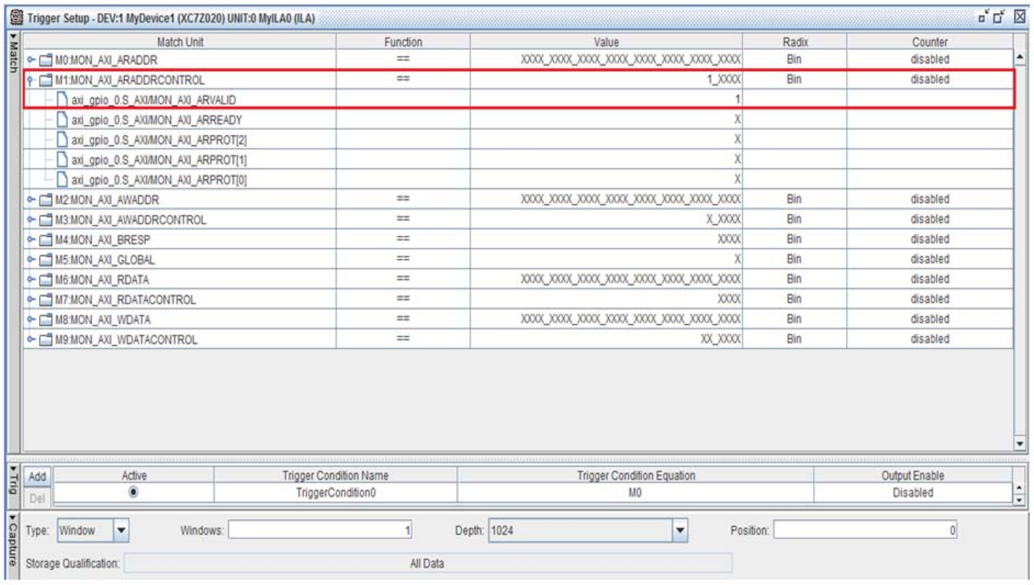
The corresponding lab course for this Test Drive is *SDK: Debugging*. In this lab, you'll launch the SDK Debug Perspective and the stopwatch application. You'll use these to practice debugging, setting breakpoints, calculating interrupt latency, and stepping through the program's operation.

A link to training labs is available in Appendix B, Additional Resources.

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.

1. Re-download the bitstream and application to the ZC702 as described in 3.1.2 Take a Test Drive! Working with SDK.
2. Run the Application and Close SDK.
3. Open ChipScope Pro™ Analyzer.
4. Make sure the hardware is connected to the USB port of your computer using a Xilinx® Platform Cable. You must also have the Xilinx Platform Cable device driver installed.
5. Click the **Open/Search JTAG Cable** button .
6. Click **OK**.
7. Import a `*.cdc` file in ChipScope and do the following:
    a. Select **Dev 1 Mydevice1(XC7020)**.
    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**.
8. Set a trigger at the "ARVALID" signal by doing the following.
    a. Expand the Trigger Setup window.
    b. 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.



Trigger Setup - DEV:1 MyDevice1 (XC7Z020) UNIT:0 MyILA0 (ILA)

Match

Match Unit	Function	Value	Radix	Counter
M0:MON_AXI_ARADDR	==	XXXX_XXXX_XXXX_XXXX_XXXX_XXXX_XXXX_XXXX	Bin	disabled
M1:MON_AXI_ARADDRCONTROL	==	1_XXXX	Bin	disabled
axi_gpio_0.S_AXI/MON_AXI_ARVALID		1		
axi_gpio_0.S_AXI/MON_AXI_ARREADY		X		
axi_gpio_0.S_AXI/MON_AXI_ARPROT[2]		X		
axi_gpio_0.S_AXI/MON_AXI_ARPROT[1]		X		
axi_gpio_0.S_AXI/MON_AXI_ARPROT[0]		X		
M2:MON_AXI_AWADDR	==	XXXX_XXXX_XXXX_XXXX_XXXX_XXXX_XXXX_XXXX	Bin	disabled
M3:MON_AXI_AWADDRCONTROL	==	X_XXXX	Bin	disabled
M4:MON_AXI_BRESP	==	XXXX	Bin	disabled
M5:MON_AXI_GLOBAL	==	X	Bin	disabled
M6:MON_AXI_RDATA	==	XXXX_XXXX_XXXX_XXXX_XXXX_XXXX_XXXX_XXXX	Bin	disabled
M7:MON_AXI_RDATACONTROL	==	XXXX	Bin	disabled
M8:MON_AXI_WDATA	==	XXXX_XXXX_XXXX_XXXX_XXXX_XXXX_XXXX_XXXX	Bin	disabled
M9:MON_AXI_WDATACONTROL	==	XX_XXXX	Bin	disabled

Trig

Active	Trigger Condition Name	Trigger Condition Equation	Output Enable
●	TriggerCondition0	M0	Disabled

Capture

Type: Window Windows: 1 Depth: 1024 Position: 0

Storage Qualification: All Data

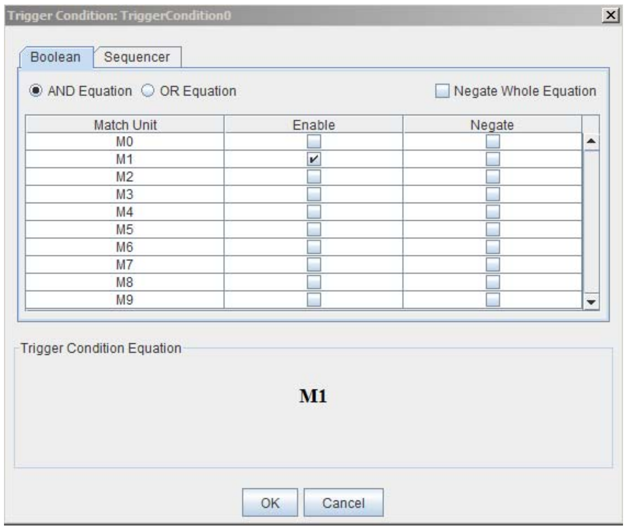
Figure 4-2: **Trigger Setup Window, MON_AXI_AVALID Setting**

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.



Trigger Condition: TriggerCondition0

(● AND Equation) ○ OR Equation ☐ Negate Whole Equation

Match Unit	Enable	Negate
M0	☐	☐
M1	✓	☐
M2	☐	☐
M3	☐	☐
M4	☐	☐
M5	☐	☐
M6	☐	☐
M7	☐	☐
M8	☐	☐
M9	☐	☐

Trigger Condition Equation

M1

Figure 4-3: **Trigger Condition Dialog Box**

9. Click **OK**.
10. 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.

11. Click the **Run** button .

    ChipScope Analyzer waits for the trigger event.

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

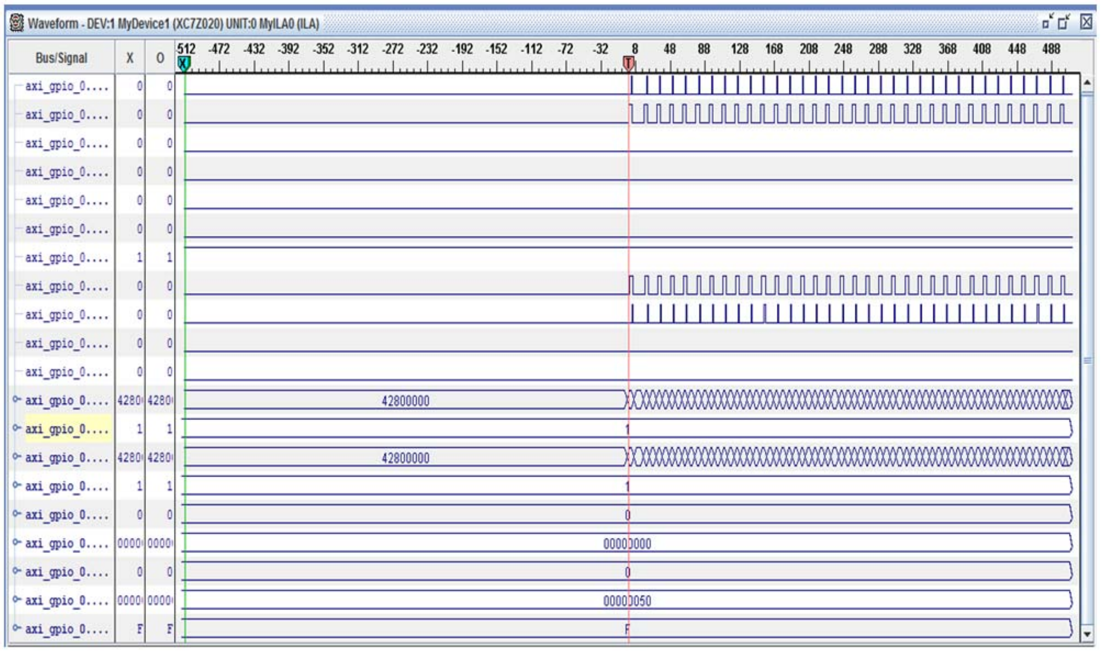


Figure 4-4: **Waveforms**

Associated Training Lab

The corresponding lab course for this Test Drive is *Advanced EDK: Debugging with ChipScope Pro Software*. In this lab, you'll perform simultaneous hardware and software debugging with the ChipScope Pro Analyzer software, the SDK Debug Perspective (GDP), and XMD.

A link to training labs is available in Appendix B, Additional Resources.

Chapter 5

Linux Booting and Application Debugging Using SDK

This chapter describes the steps to boot the Linux OS on the Zynq™-7000 All Programmable SoC board. It also provides information about downloading images precompiled by Linux on the target memory using a JTAG interface. The later part of this chapter covers programming of the following non-volatile memory with the Linux precompiled images, which are used for automatic Linux booting after switching on the board:

- On-board QSPI Flash
- SD card

This chapter also describes using the SDK remote debugging feature to debug Linux applications running on the target board. The SDK tool software runs on the Windows host machine. For application debugging, SDK establishes an Ethernet connection to the target board that is already running the Linux OS.

5.1 Requirements

In this chapter, the target platform points to a Zynq board. The host platform points a Windows machine that is running the ISE® Design Suite tools.

Note: The Das U-Boot universal bootloader is required for the tutorials in this chapter. It is included in the precompiled images that you will download next.

From the Xilinx documentation website, download the `ug873_design_files.zip` file. A link to this document is available in Appendix B, Additional Resources. It includes the following files:

- `BOOT.bin`: Binary image containing the FSBL and U-Boot images produced by bootgen.
- `Boot.bif`: The file to control bootgen during the creation of BOOT.BIN.
- `Devicetree.dtb`: Device tree binary large object (blob) used by Linux, loaded into memory by U-Boot.
- `ramdisk8M.image.gz`: Ramdisk image used by Linux, loaded into memory by U-Boot.
- `README.txt`: Description of the release.
- `U-boot.elf`: U-Boot file used to create the BOOT.BIN image.
- `zImage`: Linux kernel image, loaded into memory by U-Boot
- `zynq_fsbl_0.elf`: FSBL image used to create BOOT.BIN image

5.2 Booting Linux on a Zynq Board

This section covers the flow for booting Linux on the target board using the precompiled images that you downloaded in 5.1 Requirements.

Note: The compilations of the different images like Kernel image, U-Boot, Device tree, and root file system is beyond the scope of this guide.

5.2.1 Boot Methods

The following boot methods are available:

- Master Boot Method
- Slave Boot Method

Master Boot Method

In the master boot method, different kinds of non-volatile memories like QSPI, NAND, NOR flash, and SD cards are used to store boot images. In this method, the CPU loads and executes the external boot images from non-volatile memory into the Processor System (PS). The master boot method is further divided into *Secure* and *Non Secure* modes. Refer to the *Zynq-7000 Extensible Processing Platform Technical Reference Manual* (UG585) for more detail. A link to this document is available in Appendix B, Additional Resources.

The boot process is initiated by one of the ARM Cortex-A9 CPUs in the processing system (PS) and it executes on-chip ROM code. The on-chip ROM code is responsible for loading the first stage boot loader (FSBL). The FSBL does the following:

- Configures the FPGA with the hardware bitstream (if it exists)
- Configures the MIO interface
- Initializes the DDR controller
- Initializes the clock PLL
- Loads and executes the Linux U-Boot image from non-volatile memory to DDR

The U-Boot loads and starts the execution of the Kernel image, the root file system, and the device tree from non-volatile RAM to DDR. It finishes booting Linux on the target platform.

Slave Boot Method

JTAG can only be used in slave boot mode. An external host computer acts as the master to load the boot image into the OCM using a JTAG connection.

Note: The PS CPU remains in idle mode while the boot image loads. The slave boot method is always a non-secure mode of booting.

In JTAG boot mode, the CPU enters halt mode immediately after it disables access to all security related items and enables the JTAG port. You must download the boot images into the DDR memory before restarting the CPU for execution.

5.2.2 Booting Linux from JTAG

The following flow chart describes the process used to boot Linux on the target platform.

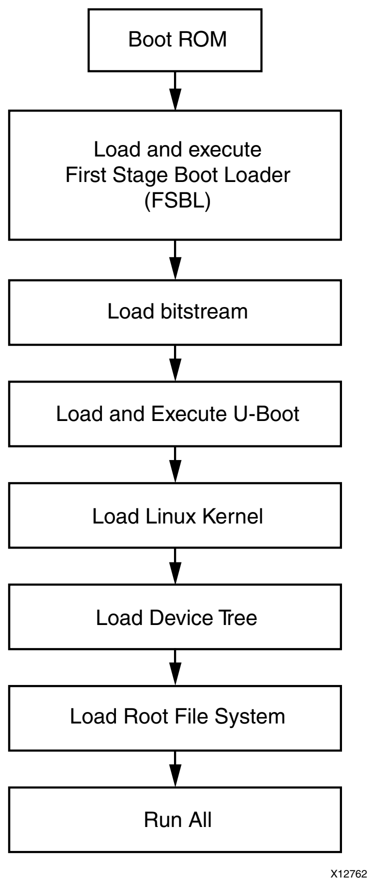


Figure 5-1: **Linux Boot Process on the Target Platform**

5.2.3 Take a Test Drive! Linux Booting Using JTAG Mode

1. Check the following Board Connection and Setting for Linux booting using JTAG mode:
   - Ensure that the settings of Jumpers J27 and J28 are as described in Take a Test Drive! Running the "Hello World" Application, page 16.
   - Connect an Ethernet cable from the Zynq board to your Windows host machine.
   - Connect the power cable to the board.
   - Connect a Xilinx Platform cable USB II cable between the Windows Host machine and the Target board.
   - Connect a USB cable to connector J17 on the target board with the Windows Host machine. This is used for USB to serial transfer.
   - Change Ethernet Jumper J30 and J43 as shown in Figure 5-2.

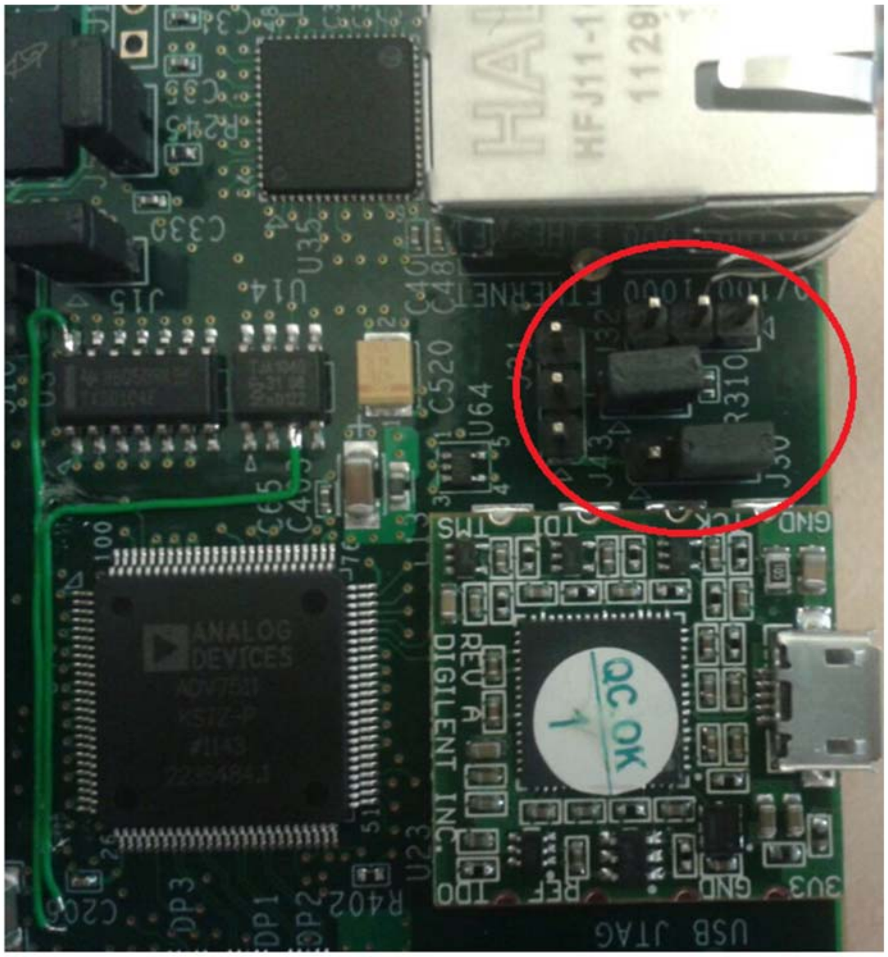


Figure 5-2: **Change Jumpers J30 and J43**

   - Power on the target board.
2. Launch SDK and open same workspace you used in Chapter 2 and Chapter 3.
3. If the serial terminal is not open, connect the serial communication utility with the baud rate set to 115200.

4. Download the bitstream by selecting **Xilinx Tools > Program FPGA**, then clicking **Program**.
5. Open the XMD tool by selecting **Xilinx Tools > XMD console**.
6. At the XMD prompt, do following:
   a. Type **`connect arm hw`** to connect with the PS section CPU.
   b. Type **`source <Project Dir>/project_1/project_1.sdk/SDK_Export/system_hw_platform/ps7_init.tcl`** and then type **`ps7_init`** to initialize the PS section (such as Clock PLL, MIO, and DDR initialization).
   c. Type **`dow directory/u-boot.elf`** to download Linux U-Boot.
   d. Type **`con`** to start execution of U-Boot.

      On the serial terminal, the autoboot countdown message appears:

      ```
      Hit any key to stop autoboot: 10
      ```
   e. Press **Enter**.

      Automatic booting from U-Boot stops and a command prompt appears on the serial terminal.
   f. At the XMD Prompt, type **`stop`**.

      The U-Boot execution is stopped.
   g. Type **`dow -data directory/zImage 0x8000`** to download the Linux Kernel image (zImage) at location 0x8000.
   h. Type **`dow -data directory/ramdisk8M.image.gz 0x800000`** to download the Linux root file system image at location 0x800000.
   i. Type **`dow -data directory/devicetree.dtb 0x1000000`** to download the Linux device tree at location 0x1000000.
   j. Type **`con`** to start executing U-Boot.
7. At the command prompt of the serial terminal, type **`go 0x8000`**.

   The Linux OS boots. After booting completes, the `Zynq>` prompt appears on the serial terminal.
8. At the `Zynq>` prompt, do the following:
   a. Set the IP address of the board by typing the following command at the `Zynq>` prompt:
      `ifconfig eth0 10.10.70.120 netmask 255.255.255.0`

      This command sets the board IP address to `10.10.70.120`.
   b. Check the connection with the board by typing **`ping 10.10.70.120`**. The following ping response displays in a continuous loop:
      `64 bytes from 10.10.70.120: seq=2 ttl=64 time=0.074 ms`

      This response means that the connection between the Windows host machine and the target board is established.
   c. Press **`Ctrl+C`** to stop displaying the ping response.

      Linux booting completes on the target board and the connection between the host machine and the target board is done. The next Test Drive describes using SDK to debug the Linux application.

5.2.4 Take a Test Drive! Debugging the Linux Application Using SDK Remote Debugging

In this section, you will create an SDK default Linux `hello world` application and practice the steps for debugging the Linux application from the Windows host machine.

1. Set up your Windows machine as host by doing the following:
   a. Select **`Start > All Programs > Accessories`**.
   b. Right-click the **Command Prompt** and select **Run as administrator**. The DOS shell opens.
   c. Set the IP address of your Windows machine by typing the following in the DOS shell:
   ```
   netsh interface ip set address name="Local Area Connection" static
   10.10.70.101 255.255.255.0 192.168.0.1 1
   ```
2. In SDK, select **File > New > Xilinx C Project**.
3. The New Project wizard opens. Click the **Linux** option in the Target Software.
4. Select **Linux Hello World** in the Template list and keep the remaining default options. The processor is `ps7_cortexa9_0`.

   The Location of your project, hardware platform used, and processor are visible in this dialog box.
5. Click **Finish** to generate the application.

   The Hello world application compiles and the .elf file generates.
6. Right-click `linux_hello_world_0` and select **Debug as > Debug Configurations**.

   The Debug Configuration wizard opens.
7. In the Debug Configuration wizard, right-click **Remote ARM Linux Application** and click **New**.
8. In the Connection drop-down list, click **New**.

   The New Connection wizard opens.
9. Click the **SSH Only** tab and click **Next**.
10. In the **Host Name** tab, type the target board IP.

    Note: Use a target IP that is similar to **`10.10.70.120`**, which you already populated in 5.2.3 Take a Test Drive! Linux Booting Using JTAG Mode, step 8a.
11. Set the connection name and description in the respective tabs.
12. Click **Finish** to create the connection.
13. In the Debug Configuration wizard, under Remote "Absolute File Path for C/C++ Application," click the **Browse** button [...]. The Select Remote C/C++ Application File wizard opens.

14. Do the following:
    a. Expand the root directory. It opens the Enter Password wizard.
    b. Provide the user ID and Password (**`root/root`**); select the **Save ID** and **Save Password** options.
    c. Click **OK**.

       The window displays the root directory contents, because you previously established the connection between the Windows host machine and the target board.
    d. Right-click on the "/" in the path name and create a new directory; name it `Apps`.
    e. In the `Apps` directory, create a new file titled `linux_hello_world_0.elf`.
    f. Provide an application absolute path, such as `/Apps/linux_hello_world_0.elf`.
15. Click **Apply**.
16. Click **Debug**.

    The Debug Perspective opens.
17. Follow the debugging procedure outlined in Take a Test Drive! Debugging with Software Using SDK, page 29.

    Note: The Linux application output displays in the SDK console, not the Terminal application you have open for running Linux.
18. After you finish debugging the Linux application, close SDK.
19. Revert back to the Windows machine IP by doing the following:
    a. In Windows, select **Start > All Programs > Accessories**. Right-click the Command Prompt and select **Run as administrator**.

       The DOS shell opens.
    b. In the DOS shell, type: **`netsh interface ip set address "Local Area Connection" dhcp`**.

5.2.5 Take a Test Drive! Booting Linux from QSPI Flash

This Test Drive covers the following steps:

1. Create the First Stage Boot Loader Executable File
2. Make a Linux Bootable Image for QSPI Flash
3. Program QSPI Flash With the Boot Image:
   a. Using SDK Program Flash Utility
   b. Using JTAG and U-Boot Command
4. Booting Linux from QSPI Flash

Create the First Stage Boot Loader Executable File

Note: You can skip this step by using the `zynq_fsbl_0.elf` provided in the downloaded precompiled images.

1. In SDK, select **File > New > Xilinx C Project**.

   The New Project wizard opens.

2. Select **Zynq FSBL** in the Template list and keep the remaining default options. The Location of your project, the hardware platform used, and the processor are visible in this window. The processor is `ps7_cortexa9_0`.
3. Click **Finish** to generate the FSBL.

   The Zynq FSBL compiles and .elf file is generated.

Make a Linux Bootable Image for QSPI Flash

1. In SDK, select **Xilinx Tools > Create Boot Image**.

   The Create Zynq Boot Image wizard opens.

2. Provide the `zynq_fsbl_0.elf` path in the FSBL ELF tab.

   Note: You can find `zynq_fsbl_0.elf` in `<project dir>/project_1/project_1.sdk/SDK/SDK_Export/zynq_fsbl_0/Debug`.

   Alternately, you can use `zynq_fsbl_0.elf` from the file you downloaded in 5.1 Requirements.

3. Add the U-Boot image.
4. Add the Linux Kernel image, such as `zImage.bin`, and provide the offset **`0x100000`**.

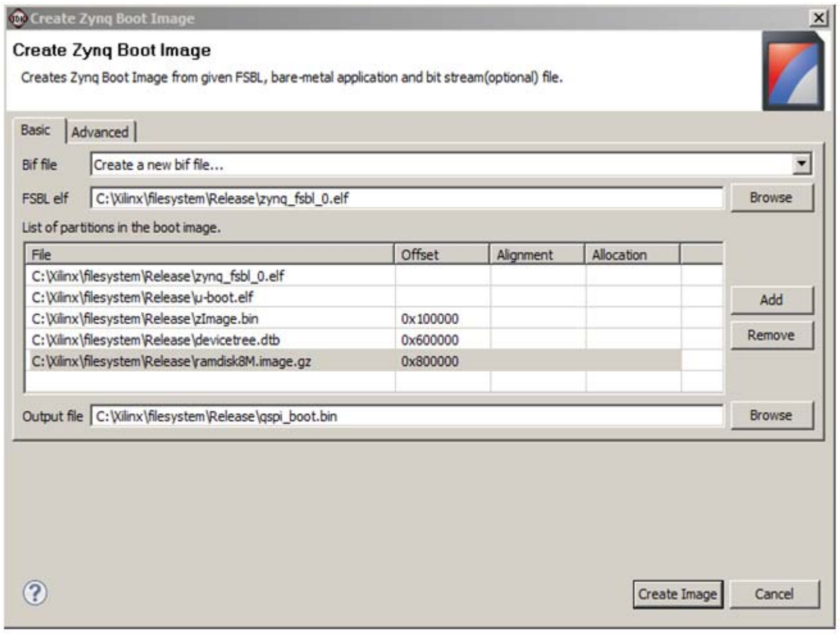
IMPORTANT: *There is a Known Issue with the Bootgen command: it does not accept a file without a file extension. To work around this issue, change the `zImage` downloaded file to `zImage.bin`.*

5. Add the device tree image (`devicetree.dtb`) and provide offset - **`0x600000`**.

6. Add the root file system image (`ramdisk8M.image.gz`) and provide offset **`0x800000`**.

   The provided offsets are predefined in the U-Boot. U-Boot expects those addresses when booting from QSPI flash. If you want to change the offset, you must modify and rebuild the U-Boot.

7. Provide the output file name as `qspi_boot.bin` in Output file tab.



Bif file: Create a new bif file...

FSBL elf: C:\Xilinx\filesystem\Release\zynq_fsbl_0.elf

List of partitions in the boot image.

File	Offset	Alignment	Allocation
C:\Xilinx\filesystem\Release\zynq_fsbl_0.elf			
C:\Xilinx\filesystem\Release\u-boot.elf			
C:\Xilinx\filesystem\Release\zImage.bin	0x100000		
C:\Xilinx\filesystem\Release\devicetree.dtb	0x600000		
C:\Xilinx\filesystem\Release\ramdisk8M.image.gz	0x800000		

Output file: C:\Xilinx\filesystem\Release\qspi_boot.bin

Figure 5-3: **Creating a Zynq Boot Image**

8. Click **Create Image**.
9. The Create Zynq Boot Image window creates the `qspi_boot.bin` file, which contains the specified files at the offset addresses.

Program QSPI Flash With the Boot Image

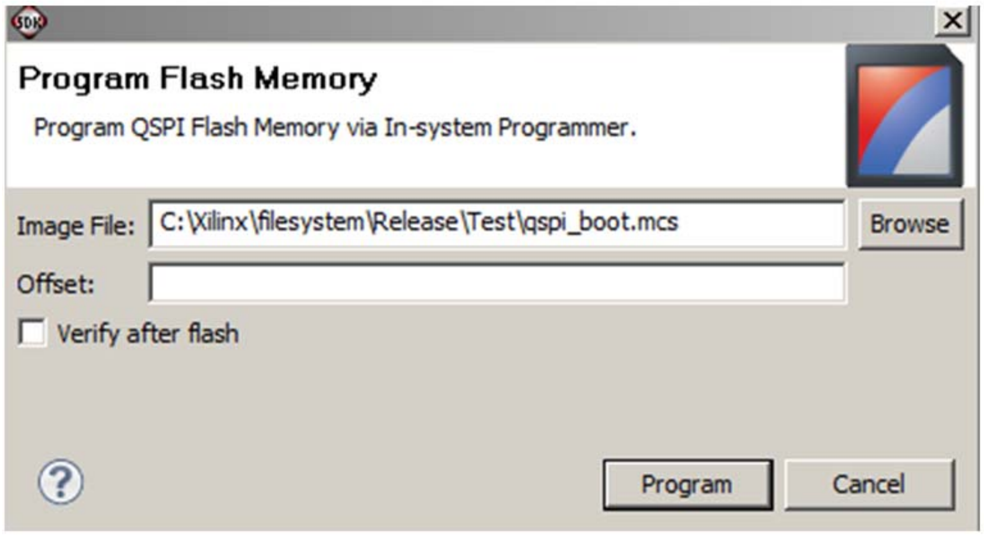
You can program QSPI Flash with the boot image using either the SDK utility or JTAG.

Using SDK Program Flash Utility

To program QSPI Flash with the boot image using the SDK Program Flash Utility, do the following:

1. Power on the ZC702 board.
2. Open SDK and provide the path of workspace you have been using throughout this guide.
3. Select **Xilinx Tools > Program Flash**.

   The Program Flash Memory dialog box opens.



Program Flash Memory

Program QSPI Flash Memory via In-system Programmer.

Image File: C:\Xilinx\filesystem\Release\Test\qspi_boot.mcs

Offset:

☐ Verify after flash

Figure 5-4: **Program Flash Memory Dialog Box**

4. Provide the location of the image file, `qspi_boot.mcs`. Don't provide the offset.
   Note: Click to select the **Verify after Flash** check box. This optional setting is used to verify the correctness of the data written to Flash.
5. Click **Program**.
   The Flash Memory will take time to program. You can monitor the progress in the SDK Console window.

Using JTAG and U-Boot Command

Note: If you already programmed QSPI Flash using the SDK Program Flash utility, skip this section.

1. Power on the ZC702 Board.
2. If a serial terminal is not open, connect the serial terminal with the baud rate set to 115200.
3. Select **Xilinx Tools > XMD Console** to open the XMD tool.
4. From the XMD prompt, do the following:
   a. Type **`connect arm hw`** to connect with the PS section CPU.
   b. type **`source <Project Dir>/project_1/project_1.sdk/SDK_Export/system_hw_platform/ps7_init.tcl`** and then type **`ps7_init`** to initialize the PS section.
   c. Type **`dow directory/u-boot.elf`** to download the Linux U-Boot to the QSPI Flash.
   d. Type **`dow -data qspi_boot.bin 0x08000000`** to download the Linux bootable image to the target memory at location 0x08000000.

      Note: You just downloaded the binary executable to DDR memory. You can download the binary executable to any address in DDR memory, but do not change the U-Boot executable, which is loaded at 0x04000000. You run this file after loading the `qspi_boot.bin` data file.
   e. Type **`con`** to start execution of U-Boot.

   On the serial terminal, the autoboot countdown message appears:

   ```
   Hit any key to stop autoboot: 10
   ```

5. Press **Enter**.

   Automatic booting from U-Boot stops and the U-Boot command prompt appears on the serial terminal.

6. Do the following steps to program U-Boot with the bootable image:

   a. At the prompt, type **`sf probe 0 0 0`** to select the QSPI Flash.

   b. Type **`sf erase 0 0x01000000`** to erase the Flash data.

      This command completely erases 16 MB of on-board QSPI Flash memory.

   c. Type **`sf write 0x08000000 0 0xFFFFFF`** to write the boot image on the QSPI Flash.

      Note that you already copied the bootable image at DDR location 0x08000000. This command copied the data, of the size equivalent to the bootable image size, from DDR to QSPI location `0x0`.

      For this example, because you have 16 MB of Flash memory, you copied 16 MB of data. You can change the argument to adjust the bootable image size.

7. Power off the board.

Booting Linux from QSPI Flash

1. After you program the QSPI Flash, set the Jumper settings on your board as shown in Figure 5-5. You'll need to set the following Jumpers: J20, J21, J22, J25, J26, J27, and J28.

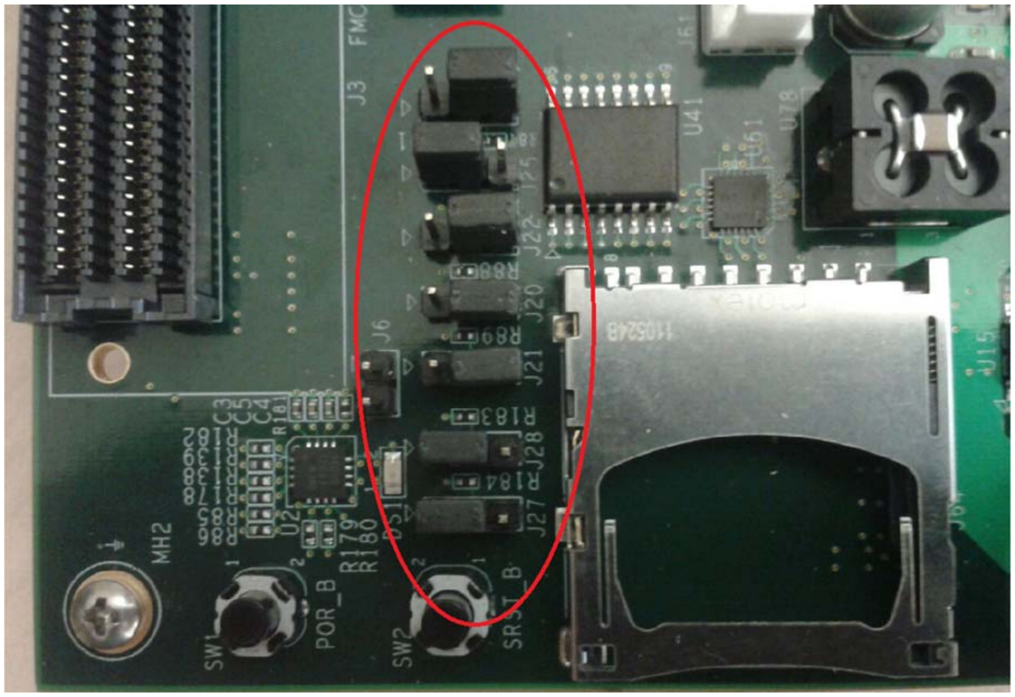


Figure 5-5: **Jumper Settings for Booting Linux from QSPI Flash**

2. Connect the Serial terminal with a 115200 baud rate setting.

3. Switch on the board power.

   A Linux booting message appears on the serial terminal. After booting finishes, the `Zynq>` prompt appears.

4. Set the Board IP address and check the connectivity as discussed in Take a Test Drive! Linux Booting Using JTAG Mode, page 37.

For Linux Application creation and debugging, refer to Take a Test Drive! Debugging the Linux Application Using SDK Remote Debugging, page 39.

5.2.6 Take a Test Drive! Booting Linux From the SD Card

1. Change the settings for Jumpers J20, J21, J22, J25, J26, J27, and J28 as shown in Figure 5-6.

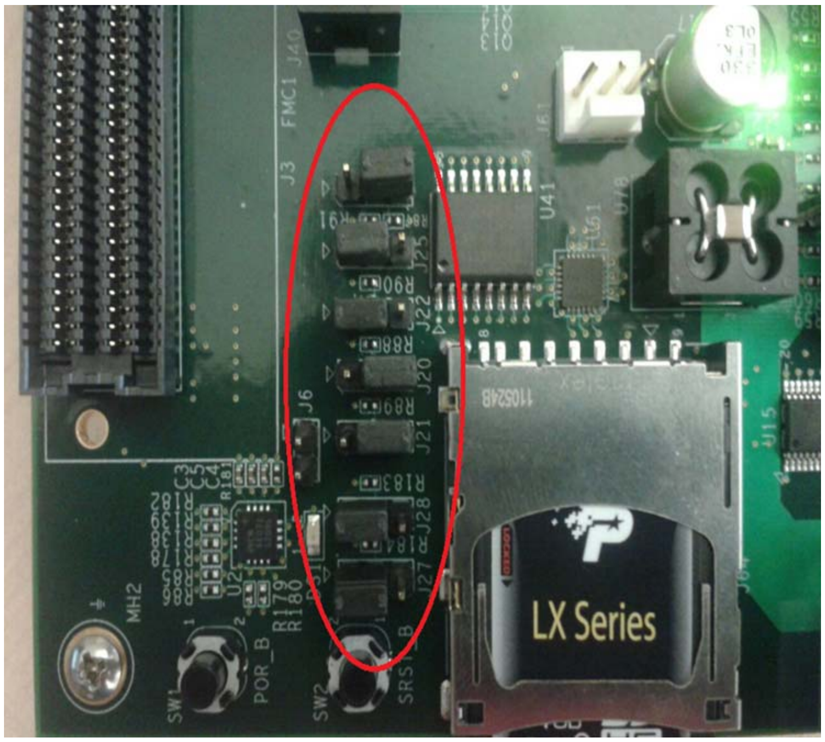


Figure 5-6: **Jumper Settings for Booting Linux from SD Card**

2. Make the board settings as described in Take a Test Drive! Linux Booting Using JTAG Mode, page 37.

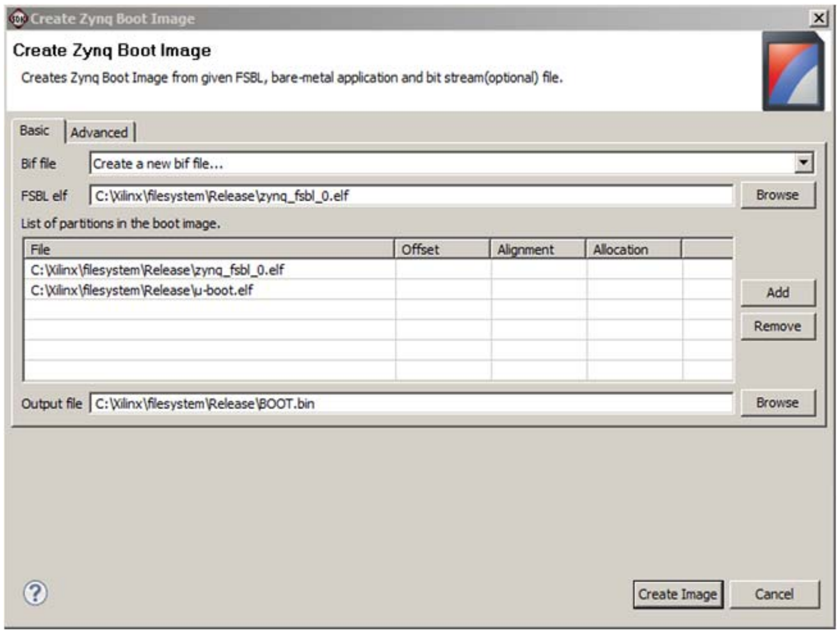
3. Create an FSBL for your design as described in Create the First Stage Boot Loader Executable File, page 41.

   Note: If you do not need to change the default FSBL image, you can use the `zynq_fsbl_.elf` file that you downloaded as part of the .zip file for this guide.

4. In SDK, select **Xilinx Tools > Create Boot Image** to open the Create Zynq Boot Image wizard.

TIME SAVER: *If there is no change in the* `zynq_fsbl_0.elf` *and* `u-boot.bin` *downloaded files, you can use the downloaded* `BOOT.bin` *file as a bootable image and skip steps 5, 6, and 7.*

5. Add `zynq_fsbl_0.elf` and `u-boot.elf`.
6. Provide the output file name as `BOOT.bin` in the **Output file** field.



Create Zynq Boot Image

Creates Zynq Boot Image from given FSBL, bare-metal application and bit stream(optional) file.

Bif file: Create a new bif file...

FSBL elf: C:\Xilinx\filesystem\Release\zynq_fsbl_0.elf

List of partitions in the boot image.

File	Offset	Alignment	Allocation
C:\Xilinx\filesystem\Release\zynq_fsbl_0.elf			
C:\Xilinx\filesystem\Release\u-boot.elf			

Output file: C:\Xilinx\filesystem\Release\BOOT.bin

Figure 5-7: **Creating the Zynq Boot Image**

7. Click **Create Image**. SDK generates the `BOOT.bin` file.
8. Copy `BOOT.bin`, `zImage`, `devicetree.dtb` and `ramdisk8M.image.gz` to the SD card.

IMPORTANT: *Do not change the file names. U-Boot searches for the file names in the SD card while booting the system.*

9. Turn on the power to the board and check the messages on the Serial terminal. The `Zynq>` prompt appears after Linux booting is complete on the target board.
10. Set the board IP address and check the connectivity as described in Take a Test Drive! Linux Booting Using JTAG Mode, page 37.

For Linux application creation and debugging, see Take a Test Drive! Debugging the Linux Application Using SDK Remote Debugging, page 39.

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:

1. Initialize the AXI GPIO module.
2. Set a direction control for the AXI GPIO pin as an input pin, which is connected with SW5 push button on the board. The location is fixed via LOC constraint in the user constraint file (UCF) during system creation.
3. Initialize the AXI TIMER module with device ID `0`.
4. Associate a timer callback function with AXI timer ISR.

   This function is called every time the timer interrupt happens. This callback switches on the LED 'DS23' on the board and sets the interrupt flag.

   The `main()` function uses the interrupt flag to halt execution, wait for timer interrupt to happen, and then restarts the execution.
5. Set the reset value of the timer, which is loaded to the timer during reset and timer starts.
6. Set timer options such as Interrupt mode and Auto Reload mode.
7. Initialize the PS section GPIO.
8. 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 'DS23' on the board.

9. 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 SW7 push button switch.
10. Initialize Snoop control unit Global Interrupt controller. Also, register Timer interrupt routine to interrupt ID '91', register the exceptional handler, and enable the interrupt.
11. 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 'DS23', starts the timer, and tells the function to wait infinitely for the Timer interrupt to happen. After the Timer interrupt happens, LED 'DS23'' switches ON and restarts execution.

For more details about API related to device drivers, refer to the *Zynq-7000 Software Developers Guide* (UG821). A link to this document is available in Appendix B, Additional Resources.

A.3 Application Software Code

The Application software for the system is included in <filename>, which is available in the `ug873_design_files.zip` file, which accompanies this guide. A link to this ZIP file is located in Appendix B, Additional Resources.

Additional Resources

B.1 Resources for This Document

The .zip file associated with this document contains the design files for the tutorials in Chapter 5, Linux Booting and Application Debugging Using SDK.

You can download this file, `ug873_design_files.zip`, from:
http://www.xilinx.com/support/documentation/zynq-7000_user_guides.htm

B.2 Xilinx Resources

- *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/irn.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/

B.3 EDK Documentation

You can also access the entire documentation set online at:
http://www.xilinx.com/support/documentation/dt_edk_edk14-1.htm

- *EDK Concepts, Tools, and Techniques* (UG683):
  http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_1/edk_ctt.pdf
- *Embedded System Tools Reference Manual* (UG111):
  http://www.xilinx.com/support/documentation/xilinx14_1/est_rm.pdf
- **MicroBlaze™ Processor User Guide** (UG081):
  http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_1/mb_ref_guide.pdf
- *Platform Specification Format Reference Manual* (UG642):
  http://www.xilinx.com/support/documentation/xilinx14_1/psf_rm.pdf

- *PowerPC 405 Processor Block Reference Guide* (UG018):
  http://www.xilinx.com/support/documentation/user_guides/ug018.pdf
- *PowerPC 405 Processor Reference Guide* (UG011):
  http://www.xilinx.com/support/documentation/user_guides/ug011.pdf
- *PowerPC 440 Embedded Processor Block in Virtex®-5 FPGAs* (UG200):
  http://www.xilinx.com/support/documentation/user_guides/ug200.pdf
- *Zynq-7000 Software Developers Guide* (UG821):
  http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_1/ug821-zynq-7000-swdev.pdf
- *Zynq-7000 Extensible Processing Platform Technical Reference Manual* (UG585):
  (UG821):http://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf

B.4 Training Labs

Training labs relating to Test Drives in this guide are located at http://www.xilinx.com/training/embedded/embedded-design-tutorials.htm.

B.5 EDK Additional Resources

- Xilinx Platform Studio and EDK website:
  http://www.xilinx.com/ise/embedded_design_prod/platform_studio.htm
- Xilinx Platform Studio and EDK Document website:
  http://www.xilinx.com/ise/embedded/edk_docs.htm
- Xilinx XPS/EDK Supported IP website:
  http://www.xilinx.com/ise/embedded/edk_ip.htm
- Xilinx Tutorial website:
  http://www.xilinx.com/support/documentation/dt_edk_edk14-1_tutorials.htm
- Xilinx Data Sheets:
  http://www.xilinx.com/support/documentation/data_sheets.htm
- Xilinx Problem Solvers:
  http://www.xilinx.com/support/troubleshoot/psolvers.htm
- Xilinx ISE® Design Suite Manuals:
  http://www.xilinx.com/support/software_manuals.htm
- GNU Manuals:
  http://www.gnu.org/manual