AXI Interface Based KC705 Embedded Kit MicroBlaze Processor Subsystem

Software Tutorial

UG915 (v1.1) October 26, 2012

XILINX®

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
08/21/12	1.0	Initial Xilinx release.
10/26/12	1.1	In step 1 in Creating a New SDK Workspace for Software Development, page 12, Suite changed to Suite <version>. In Adding a New Hardware Platform Project to an SDK Workspace, page 13, step 1 changed. In Creating the Hello World Software Project and BSP, page 14, step 1 through step 12 changed. In Creating a Profile Run Configuration, page 42, in step 6, 0x80000000 changed to 0x90000000.

Table of Contents

Revision History 2

Software Tutorial for the AXI Interface Based KC705 Embedded Kit MicroBlaze Processor Subsystem

Introduction 5
Hardware and Software Requirements 6
Prerequisites 6
System Overview 6
Software Platform(s) 6
Important Terminology 7
Included Files and Systems 7
Basic SDK Software Development Flow Overview 10
Board Setup 11
Stand-Alone Software Development 11
Creating an SDK Workspace and Adding a Hardware Platform Project 12
Creating a New BSP and Stand-Alone Software Application Project 13
Board Test Software Application Design Flow 18
Debugging Stand-Alone Software Applications 32
Profiling Stand-Alone Software Applications 40
Video Demonstration with xilkernel and lwip 46
Creating a New SDK Workspace for the Video Demonstration 46
Adding a Local Repository 47
Importing the Hardware Platform Project, Video Demonstration Application, and BSP 48
Video Demonstration BSP 49
Video Demonstration Application 50
Video Demonstration Hardware Setup 50
Running the Video Demonstration 50
Interacting with the Video Demonstration 51
Next Steps 53
Running the Pre-Built Software Applications 55
Running the Pre-Built Board_Test_App_Console Stand-Alone Software Application 55
Running the Pre-Built Board_Test_App_Webserver Software Application 56
Running the Pre-Built Video_Demo Software Application 56

Appendix A: Additional Resources

Xilinx Resources 59
References 59

Software Tutorial for the AXI Interface Based KC705 Embedded Kit MicroBlaze Processor Subsystem

Introduction

This tutorial guides the user through the steps to compile, modify, build, and debug stand-alone software applications for the Xilinx® KC705 Embedded Kit MicroBlaze™ Processor Subsystem. The Xilinx Software Development Kit (SDK) is the primary tool used with the tutorial.

The software application development is divided into two primary categories, based on the platform environment.:

- Application development based on a stand-alone environment.
- Application development based on the xilkernel OS environment.

For stand-alone software development, the tutorial exercises these flows:

- Creating a new workspace and importing the hardware platform into the workspace
- Creating a new board support package (BSP) and application
- Importing an existing BSP and software application into a workspace
- Debugging software applications
- Profiling software applications

For the tutorial based on the xilkernel OS environment, this tutorial demonstrates the video demonstration that is based on Xilinx VDMA, TPG, and Xylon's CVC [Ref 1] IPs provided with the KC705 Embedded Kits. This video demonstration uses a Web interface to control the video pipelines and to display graphs of AXI-MM throughput data (both read and write), on-chip temperature, and on-chip voltages (V_{CCINT} and V_{CCAUX}).

The video demonstration sections of this tutorial include:

- Creating a new workspace and adding a local repository
- Importing the hardware platform, BSP, and software application into the workspace
- Descriptions of the video demonstration BSP and application
- Setting up the video demonstration hardware
- Running the video demonstration
- Interacting with the video demonstration

Hardware and Software Requirements

Hardware and software requirements for the tutorial are detailed in UG913, *Getting Started with the Kintex-7 FPGA KC705 Embedded Kit* [Ref 2].

Prerequisites

These prerequisites are required to run the basic tutorial:

- KC705 MicroBlaze processor subsystem:
  - Hardware set up and software installed as documented in UG913, *Getting Started with the Kintex-7 FPGA KC705 Embedded Kit* [Ref 2].
  - Familiarity with the KC705 MicroBlaze processor subsystem documented in UG914, *AXI Interface Based KC705 MicroBlaze Processor Subsystem Hardware Tutorial* [Ref 3]
  - Familiarity with UG683, *EDK Concepts, Tools, and Techniques* [Ref 4].
- General basic microprocessor knowledge
- Basic knowledge of the C language
- Familiarity with compilers, linkers, and debug techniques

System Overview

This tutorial introduces developing stand-alone software applications with the Xilinx KC705 Embedded Kit. The stand-alone software development sections of this tutorial use the reference system that is documented in DS669, *AXI Interface Based KC705 Embedded Kit MicroBlaze Processor Subsystem Data Sheet* [Ref 5].

The video demonstration uses a hardware system that includes Xilinx IP cores such as VDMA and third party IP cores such as logiCVC from Xylon [Ref 1]. Creation of this system is described in UG914, *AXI Interface Based KC705 MicroBlaze Processor Subsystem Hardware Tutorial* [Ref 3].

Software Platform(s)

Different software platforms can be used to cover a variety of different needs. These platforms include:

- Stand-alone platforms for small dedicated systems that do not require an OS
- The xilkernel (kernel for Xilinx embedded processors) or uC/OS-II kernel for small systems that need to manage multiple tasks
- Linux for systems that have large software content (not demonstrated in this tutorial).

This tutorial introduces development with the stand-alone software platform. The `board_test_app` software application is provided for use with the stand-alone software platform. The video demonstration is provided as an example application that uses the xilkernel platform.

Important Terminology

Table 1 defines some important terms that are used throughout this tutorial.

Table 1: **Terms Used in this Document**

Term	Description
Hardware platform	The hardware platform is the hardware system created with the Xilinx Platform Studio (XPS).
Workspace	An SDK workspace is a directory that contains software projects and links to the associated hardware platforms. The workspace also contains SDK settings and logs.
Board support package	A board support package (BSP) in the SDK contains libraries and drivers that software applications can utilize when using the provided Application Program Interfaces (APIs).
Software project	An SDK software project includes the software application source and settings.
Stand-alone software platform	The stand-alone software platform is a single-threaded environment used when an application accesses processor functions directly. The stand-alone software platform provides functions such as processor interrupt handling, exception handling, and cache handling as well as program profiling support.
Xilkernel software platform	Xilkernel is a simple and lightweight kernel that provides services such as scheduling, threads, synchronization, and timers.
Xilinx Microprocessor Debugger (XMD)	XMD is a console-based download and debug engine.

Included Files and Systems

Table 2 summarizes how the directories provided with the KC705 Embedded Kit relate to this tutorial document. This tutorial uses the directories in the `KC705_Embedded_Kit` directory and assumes that the directories and files are placed or copied onto the local computer.

Table 2: **Directory Structure Summary**

Directory	Purpose
Folders Containing Documents	
`Documentation`	This directory contains the documents provided with the Embedded Kit, including this tutorial.
Folders Containing the Video Demonstration Application	
`Video_Demo/SW`	This directory includes the `video_demo` software project, `xilkernel_bsp_0` BSP, and the `SDK_Export` files for the hardware platform.
`Video_Demo/ready_for_download`	This directory contains the pre-built hardware bitstream and software executable files to provide a quick way to run the video demonstration application.

Table 2: **Directory Structure Summary *(Cont'd)***

Directory	Purpose
Folders Containing Board Test Applications	
KC705_System/SW/board_test_app_Console	This directory includes the board_test_app_Console software project, board_test_app_Console_platform BSP, and the SDK export files for the hardware platform. This directory supports the console-based board test application.
KC705_System/SW/board_test_app_Webserver	This directory includes the board_test_app_Webserver software project, board_test_app_Webserver_platform BSP, and the SDK export files for the hardware platform. This directory supports the Webserver-based board test application.
KC705_System/ready_for_download	This directory contains the pre-built hardware bitstream and software executable files (Console- and Webserver-based) to provide a quick way to run the board test application.
Folders Containing Tutorials and Sandbox	
Tutorial_Completed/SW	This directory includes the software applications and platforms created during this tutorial.
Tutorial_Sandbox/SW/board_test_app/board_test_app_Console	This directory is included for use as a working directory when going through the stand-alone software development sections of this tutorial. This directory is originally provided as a copy of the KC705_System/SW/board_test_app_Console directory with some added folders to facilitate completion of this tutorial. It is recommended that this directory be used when completing the stand-alone software development sections of this tutorial.
Tutorial_Sandbox/SW/board_test_app/board_test_app_Webserver	This directory is included for use as a working directory when going through the board_test_apps (based on Webserver) sections of this tutorial. This directory is originally provided as a copy of the KC705_System/SW/board_test_app_Webserver directory with some added folders to facilitate completion of this tutorial. It is recommended that this directory be used when completing the board_test_apps (based on Webserver) sections of this tutorial.
Tutorial_Sandbox/SW/Video_Demo	This directory is included for use as a working directory when going through the video demonstration sections of this tutorial. This directory is originally provided as a copy of the Video_Demo/SW directory with some added folders to facilitate completion of this tutorial. It is recommended that this directory be used when completing the video demonstration sections of this tutorial.

The Tutorial_Sandbox directory is a working area for this tutorial. Figure 1 shows the directory structure for the Tutorial_Sandbox working area.

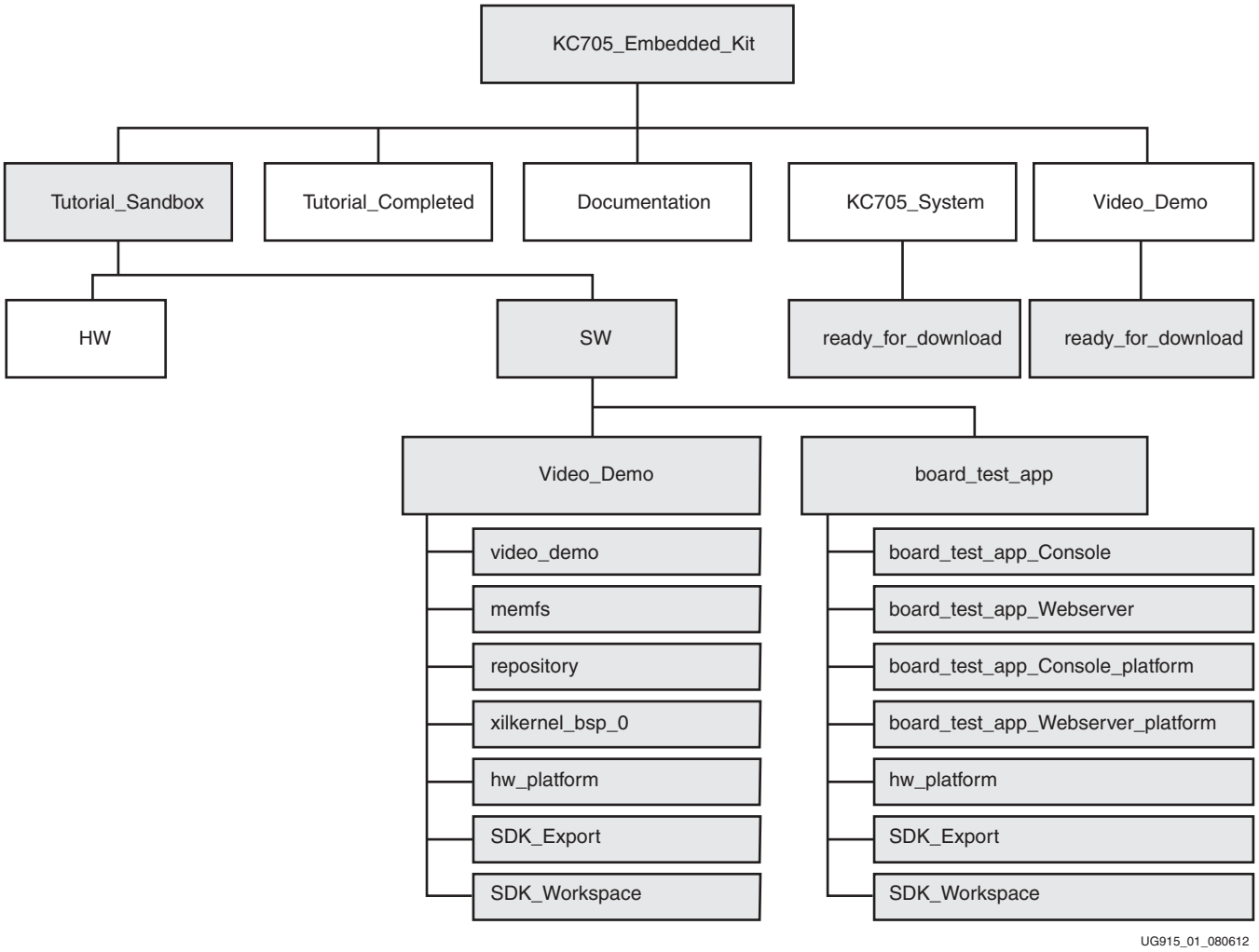


Figure 1: **Tutorial Sandbox Directory Structure**

The `Tutorial_Completed` directory includes the applications and platforms created with this tutorial, and are provided as a reference. The Tutorial_Completed directory structure is shown in Figure 2.

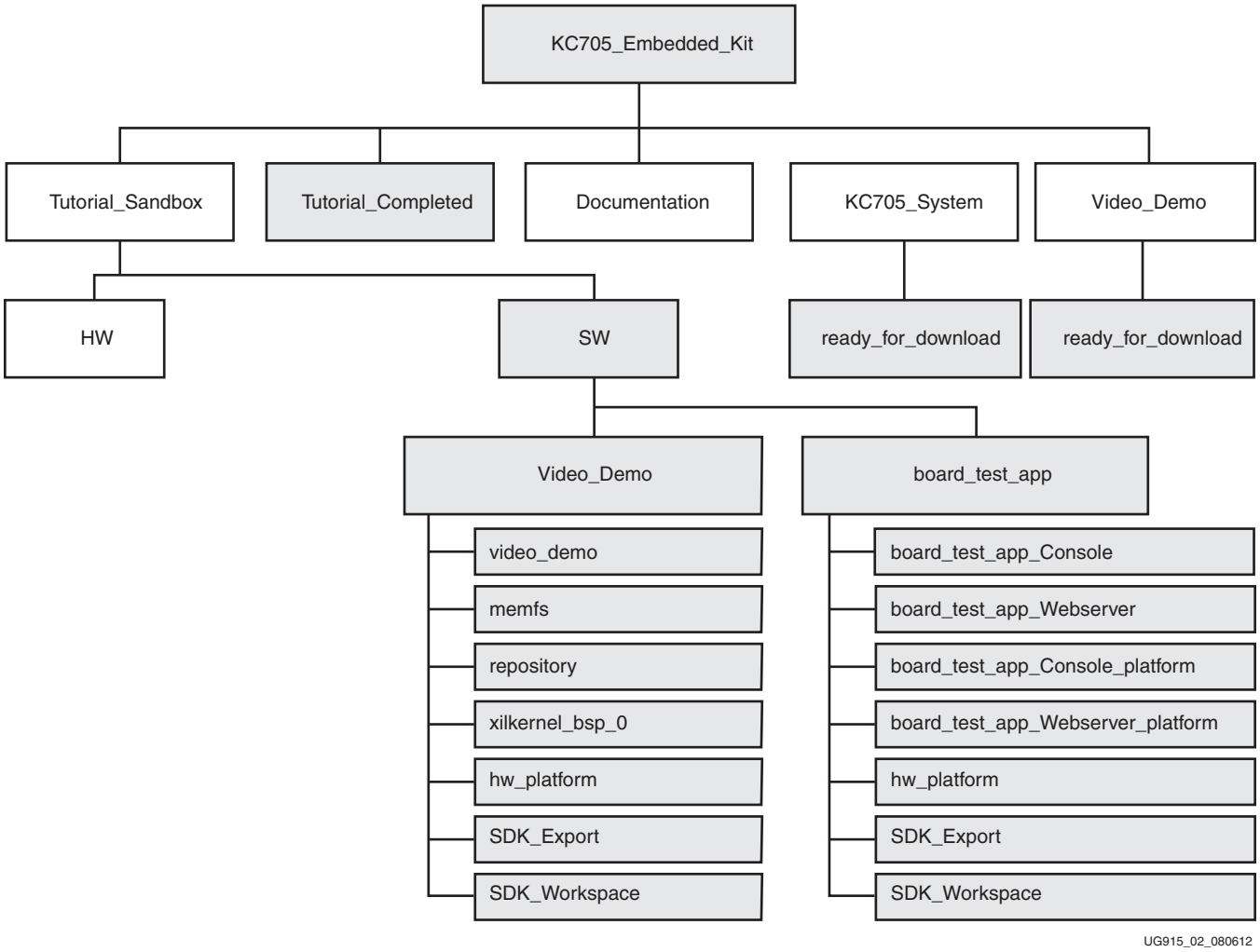


Figure 2: **Tutorial Completed Directory Structure**

Basic SDK Software Development Flow Overview

The flowchart in Figure 3 shows the basic software development flow in the SDK. This tutorial begins after the hardware platform is exported from XPS. This SDK software development flow is covered throughout this tutorial.

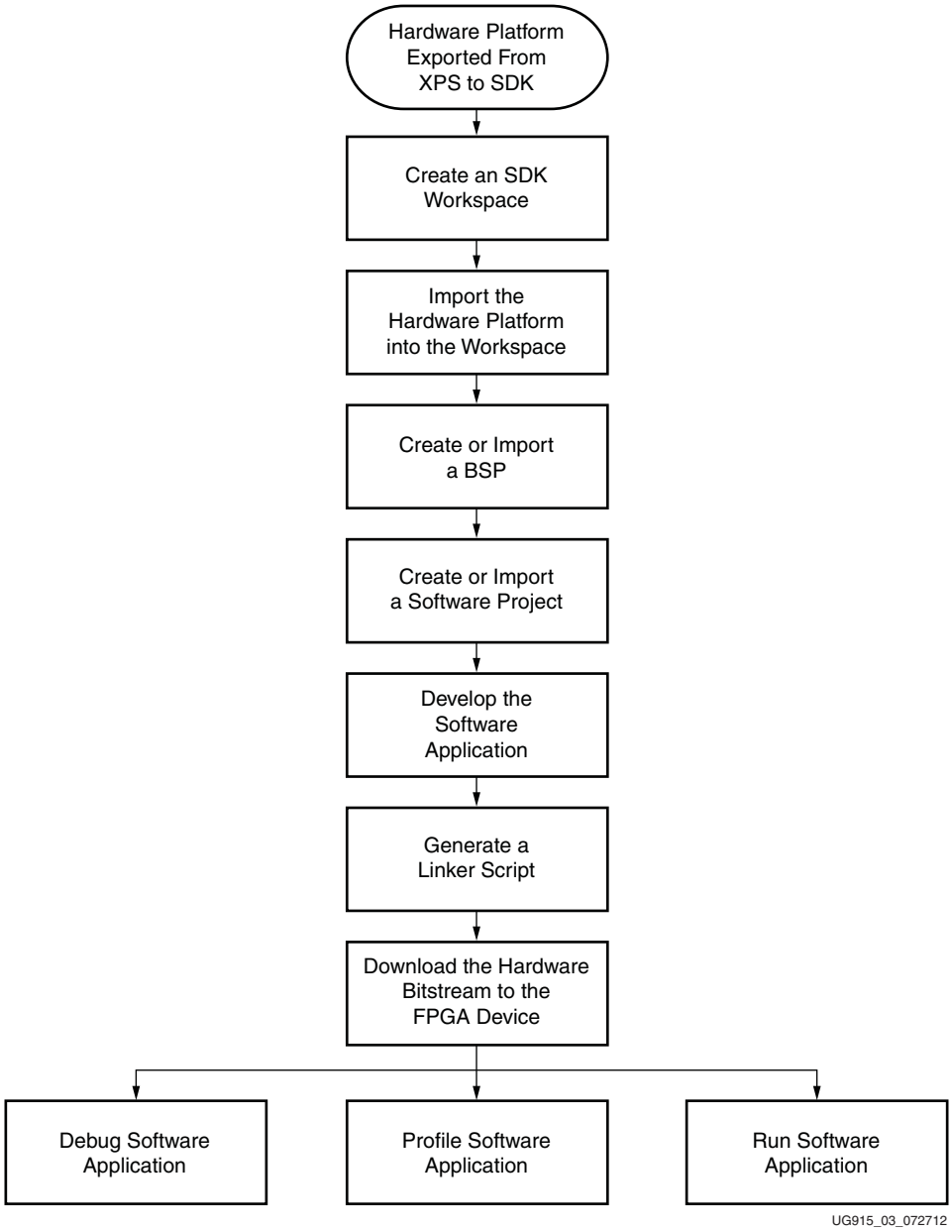


Figure 3: **SDK Software Development Flow**

Board Setup

Before beginning this tutorial, ensure that these board setup steps are completed as described in UG913, *Getting Started with the Kintex-7 FPGA KC705 Embedded Kit* [Ref 2].

Stand-Alone Software Development

As an introduction to stand-alone software development, this tutorial includes:

1. Creating an SDK Workspace and Adding a Hardware Platform Project, page 12

Includes steps on how to create a new SDK workspace and add a hardware platform project to that workspace. Completion of this section is required. All other stand-alone software development sections use the workspace and hardware platform project created in this section.

2. Creating a New BSP and Stand-Alone Software Application Project, page 13

   Describes the steps necessary to create a new software application project and corresponding BSP in the SDK.

3. Board Test Software Application Design Flow, page 18

   This section uses the provided software projects and includes the tutorial to build board test applications in Console mode and Webserver mode.

4. Debugging Stand-Alone Software Applications, page 32

   Part 4 is optional, but covers important material for an introduction to debugging software applications in the SDK.

5. Profiling Stand-Alone Software Applications, page 40

   Part 5 is important for understanding how to profile software applications in the SDK.

Creating an SDK Workspace and Adding a Hardware Platform Project

To begin working with SDK, an SDK workspace must be created. An SDK workspace is a directory that contains information on the projects being worked on. The workspace holds software project files, SDK settings, and log files.

The hardware platform is the embedded hardware design that is created in XPS and exported in the form of an XML file. The hardware platform is imported to SDK when the XML file is imported. Multiple hardware platform projects can be created in SDK.

For the stand-alone software development sections of this tutorial, the hardware platform project uses the hardware specification exported from the hardware system located in `KC705_System/HW`.

Completion of this section is required by all other stand-alone software development sections of this tutorial.

The process of creating a new SDK workspace and importing a hardware platform into the workspace includes:

- Creating a New SDK Workspace for Software Development, page 12
- Adding a New Hardware Platform Project to an SDK Workspace, page 13

Creating a New SDK Workspace for Software Development

1. Launch SDK.
   - On Windows, select **Start > All Programs > Xilinx Design Tools > ISE Design Suite <version> > EDK > Xilinx Software Development Kit**.
   - On a Linux host, enter **xsdk** at a command prompt.
2. When the Workspace Launcher appears, specify the SDK workspace as **Tutorial_Sandbox/SW/board_test_app/SDK_Workspace**. Click **OK** in the Workspace Launcher.
3. When SDK opens, a Welcome screen is displayed. Close the Welcome screen after browsing through the displayed information. When the Welcome screen is closed, the Project Explorer tab is displayed and is empty.

Adding a New Hardware Platform Project to an SDK Workspace

To develop applications with SDK, a hardware platform must be specified in the SDK workspace. To add a hardware platform project into the workspace, follow these steps:

1. Select **File > New > Project... > Xilinx > Hardware Platform Specification** in SDK.
2. Populate the new hardware platform project with these selections:
   - Project Name: **hw_platform**
   - Project Location: Ensure that the **Use default location** checkbox is selected.
   - Target Hardware Specification: Click **Browse** and select **Tutorial_Sandbox/SW/board_test_app/SDK_Export/hw/system.xml**

   This hardware platform project uses the hardware specification exported in XPS from the hardware system located in KC705/HW.
3. Click **Finish**.
4. When the hardware platform is loaded, the Project Explorer tab includes the hardware platform, as shown in Figure 4. In this view, the system XML file is shown, which provides a short summary of the hardware platform when opened in SDK.

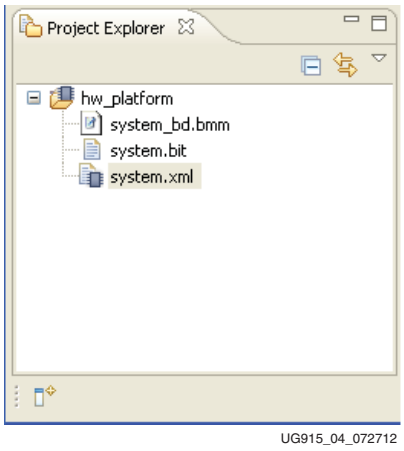


Figure 4: **SDK Project Explorer View of the Hardware Platform Project**

Creating a New BSP and Stand-Alone Software Application Project

After the SDK workspace is created and the hardware platform is imported, a new BSP can be created in the SDK. A BSP contains libraries and drivers that software applications can utilize when using the provided APIs. A software project is the software application source and settings.

The stand-alone software platform is a single-threaded environment used when an application accesses processor functions directly. Some functions that the stand-alone software platform provides are processor interrupt handling, exception handling, and cache handling. The stand-alone software platform also supports program profiling.

This section uses the SDK workspace with the imported hardware platform created in Creating an SDK Workspace and Adding a Hardware Platform Project, page 12.

This tutorial uses a Hello World software application as an example of creating a new stand-alone software project. The Hello World software project is created from one of the

example application templates provided in the SDK. The process of creating a Hello World software project with a new BSP using the stand-alone software platform includes:

- Creating the Hello World Software Project and BSP, page 14
- Running the Hello World Software Project, page 15

Creating the Hello World Software Project and BSP

Follow these steps to create the Hello World application project and a corresponding BSP. The BSP is built to use the stand-alone software platform. Although this BSP is created for use with a Hello World software application, it can be used for other software applications that require a basic stand-alone software platform.

1. Select **File > New > Application Project**. in the SDK. For Xilinx C projects, SDK automatically creates Makefiles that compile the sources into object files and link the object files into an executable file.
2. Enter the `Project Name:` as **hello_world_0**.
3. Keep **Use default location** selected.
4. Keep the default selections for **Target Hardware**.
5. Keep the `OS Platform` selection at **Standalone**, `Language` selection to **C**. For `Board Support Package` select **Create New**. This selection creates a new BSP for the Hello World software application project. Ensure that the new BSP is populated with the name `hello_world_bsp_0`.
6. Click **Next**.
7. Select the template for **Hello World**.
8. Click **Finish**. The SDK starts to compile the `hello_world_bsp_0` BSP and the `hello_world_0` application.
9. Right-click **hello_world_bsp_0** in the Project Explorer tab and select **Board Support Package Settings**. Settings for the BSP can be changed in these menus.
10. The BSP settings menu is popped up, and the Overview menu is the initial menu shown. This initial menu contains the OS type and version used, as well as a description of the selected OS type. This menu shows the target hardware specification file location and the target processor. This menu also allows the option to add libraries, such as the `Xilinx Flash` library and the `lwip TCP/IP Stack` library. No special settings are needed in this initial screen.
11. Select the **stand-alone** menu. This menu includes the configuration options for the stand-alone software platform.
12. In the Operating System configuration section of the stand-alone menu, ensure that **stdin** and **stdout** are set to **rs232_uart_1** by clicking the text in the Value column.
13. Select the **drivers** menu. In this menu, the driver and driver version used for the hardware platform peripherals can be viewed. No special settings are needed in the drivers menu.
14. Select the **cpu** menu. This menu includes processor driver parameters that can be changed, including the compiler that is used for libraries and applications, and any extra compiler flags that are needed for compilation. No special settings are needed in the cpu menu.
15. Click **OK**. The BSP and application are recompiled.
16. After the `hello_world_bsp_0` BSP and `hello_world_0` applications are compiled, expand the **microblaze_0** section of `hello_world_bsp_0` in the Project Explorer tab as shown in Figure 5.

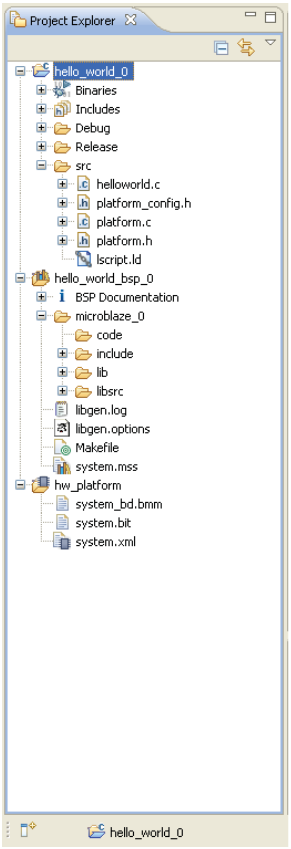


Figure 5: **Hello World Project Source Code Files**

The `code`, `include`, `lib`, and `libsrc` folders contain the libraries for the hardware peripherals in the reference system. In this view, files can be opened by double-clicking them.

17. Expand the **src** section of `hello_world_0` in the Project Explorer tab, also shown in Figure 5. The source code of the Hello World application project is stored here.
18. Double-click the **helloworld.c** file. The file opens in the SDK Editor window. The file can be modified as needed.

    Note the **lscript.ld** file in the helloworld project. This file is the linker script that was generated for the helloworld project. A linker script is required to specify where the software code is loaded in the hardware system memory.

Running the Hello World Software Project

After the Hello World application is created and built, it can be run on the KC705 board. Follow these steps to program the FPGA and run the Hello World application:

1. The FPGA must be programmed with the hardware configuration bitstream before the software application can be run. Program the FPGA with these steps:
   a. In the SDK, select **Xilinx Tools > Program FPGA**.
   b. Ensure that the `system.bit` and `system_bd.bmm` files are selected from **Tutorial_Sandbox\SW\board_test_app\SDK_Workspace\hw_platform**. Leave the initialization ELF as bootloop. Bootloop keeps the processor in a known state

while it waits for another program to be downloaded to run or be debugged. These settings are shown in Figure 6.

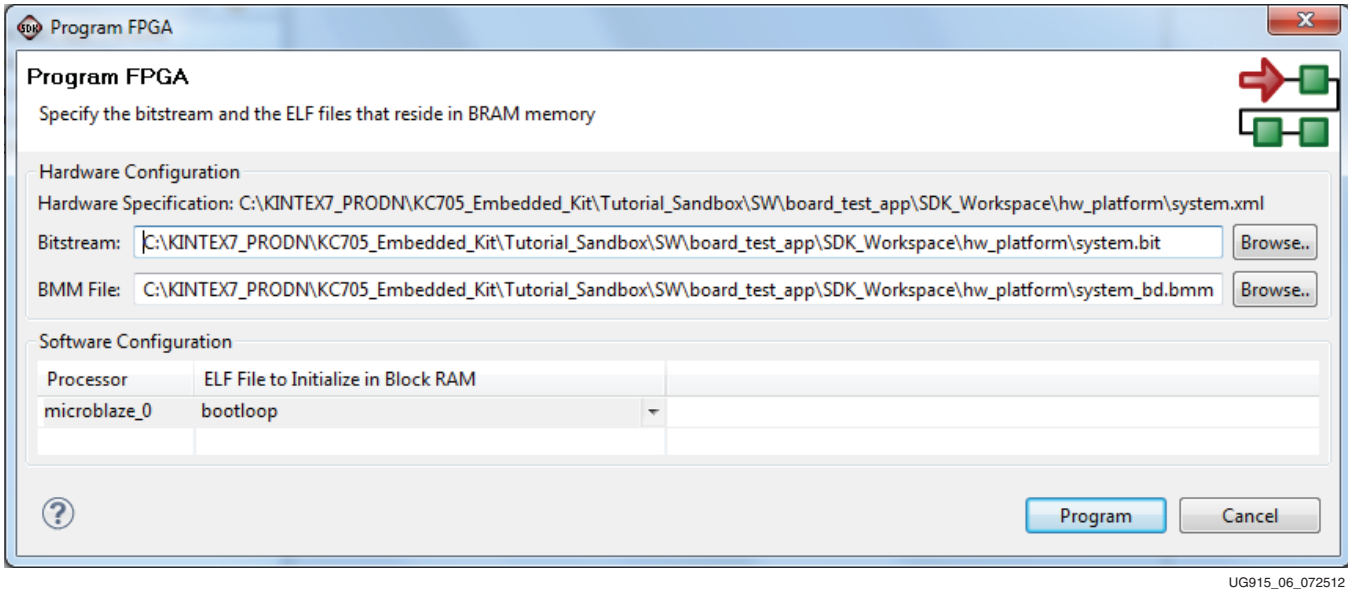


Figure 6: **Program FPGA Options**

c. Click **Program** to program the FPGA. SDK confirms when the FPGA is successfully programmed with the bitstream.

2. In the SDK, build configurations allow different versions of executables to be built with different settings. Each build configuration can customize compiler settings, macros passed for compilation, and linker settings. SDK provides two build configurations: Debug and Release.

   The application uses the Release configuration to run without debugging. Right-click the **hello_world_0** application in the Project Explorer tab and select **Build Configurations > Set Active > Release**. The `hello_world_0` application is rebuilt for the Release configuration.

3. In the Project Explorer tab, right-click **hello_world_0** and select **Run As > Run Configurations**.

4. In the Run Configurations box, select **Xilinx C/C++ ELF** and click the **New** button to create a new Run configuration as shown in Figure 7.

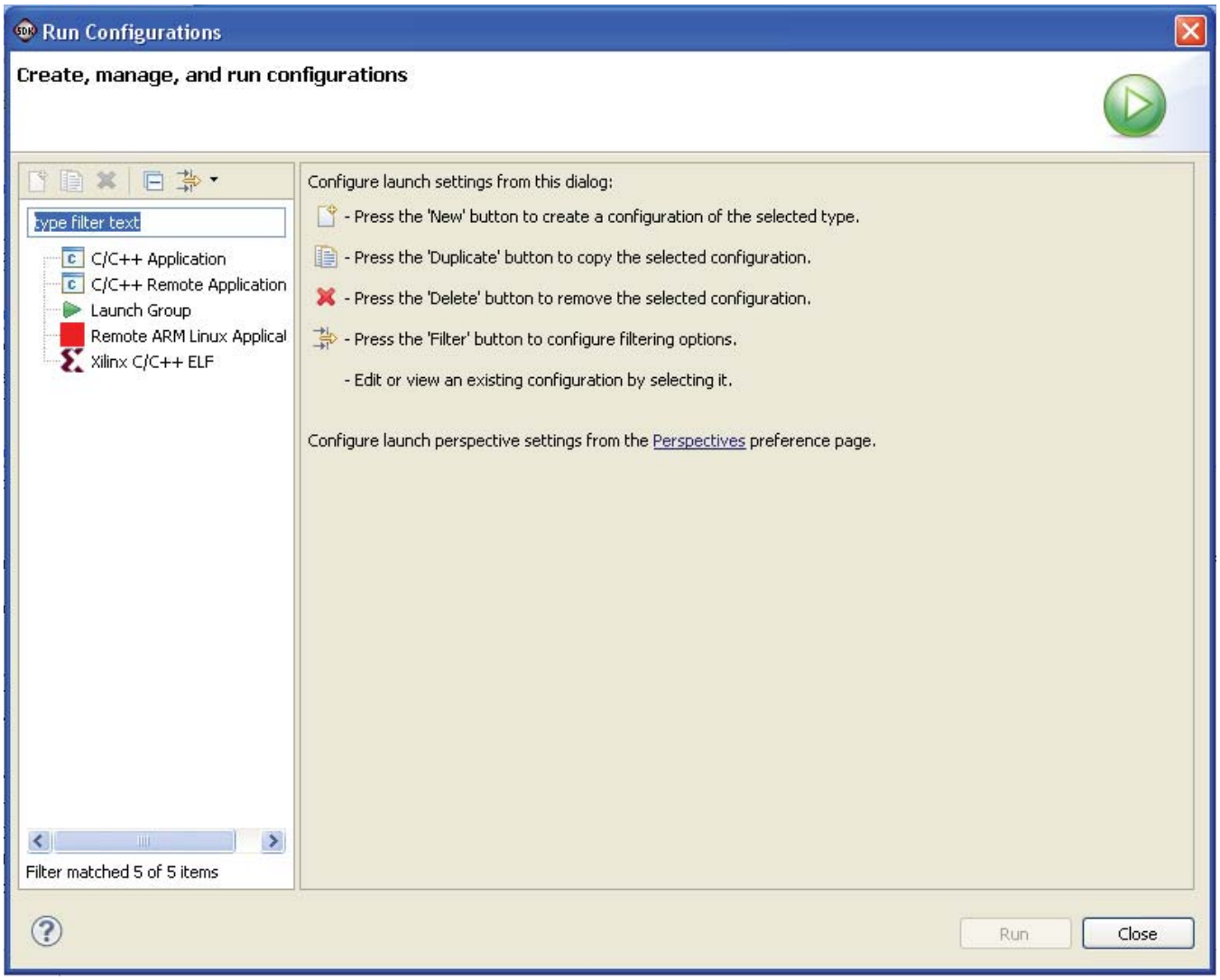


Figure 7: **New Run Configuration**

5. Populate the Main tab of the new run configuration with these selections:
   - Name: **hello_world_0 Release**
   - Project: **hello_world_0**
   - Build Configuration: **Release**
   - C/C++ Application: **Release/hello_world_0.elf**
6. Click the **STDIO Connection** tab and click the **Connect STDIO to Console** checkbox to enable the STDIO of the program to go to the SDK Console window. Set the COM port as appropriate for the host computer, and leave the BAUD rate at 9600.
7. Click **Run**. An XMD console appears which shows the `hello_world_0` application being downloaded into memory and run.
8. The `hello_world_0` application displays `Hello World` on the SDK Console, as shown in Figure 8.

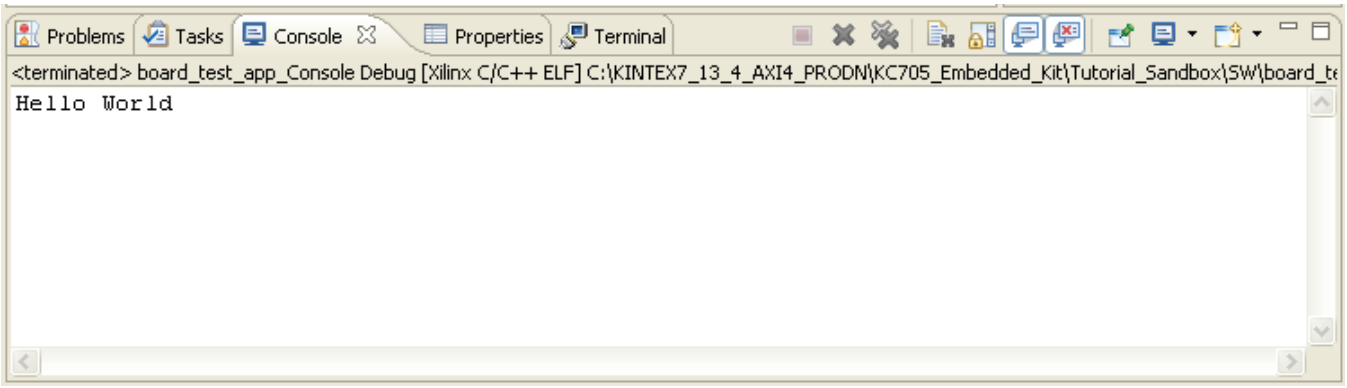


Figure 8: **Hello World Output**

Board Test Software Application Design Flow

Rather than creating a new BSP or application, the existing BSP and application provided with the Embedded Kit reference system can be imported into the SDK. This provided application is called `board_test_app` and exercises various hardware peripherals on the KC705 board.

The board test app application for KC705 can be built for two modes: Console mode and Webserver mode.

Console Mode

In Console mode, the KC705 system is built based on a stand-alone environment. The application comes up with a menu appearing in the UART console. The user can select the test applications in the console by entering the test number and pressing Enter to run the test on target. The KC705 board runs the selected test. The results of the tests are displayed on the console. It is a simple design to start with.

The BSP called `board_test_app_Console_platform` is provided for the application built in Console mode.

Webserver Mode

In Webserver mode, the KC705 system is built based on xilkernel OS and light weight IP (lwip) stack. The target supports Web server and is configured with a static IP address. When the host computer is connected to the KC705 target with an Ethernet cable and browsed with the static IP address, a web page appears on the host computer with the board test applications menu. The user can select the test applications in the web page and click to run the selected test on the target. The KC705 board runs the selected test, posts the results of the board test to the host computer, and the results are displayed in the web page.

A BSP called `board_test_app_Webserver_platform` is provided for the application built in Webserver mode.

Board Test Software Application and BSP (Console Mode)

This section describes the `board_test_app_Console` design flow. This section of the tutorial uses the SDK workspace previously created in Stand-Alone Software Development, page 11.

The design flow for the board test software application (Console mode) includes:

- Importing the Board Test Software Application and BSP (Console Mode), page 19
- Board Test BSP (Console Mode) Settings , page 21
- Running the Board Test Software Application (Console Mode), page 22
- Tests in the Board Test Application (Console Mode), page 22

Importing the Board Test Software Application and BSP (Console Mode)

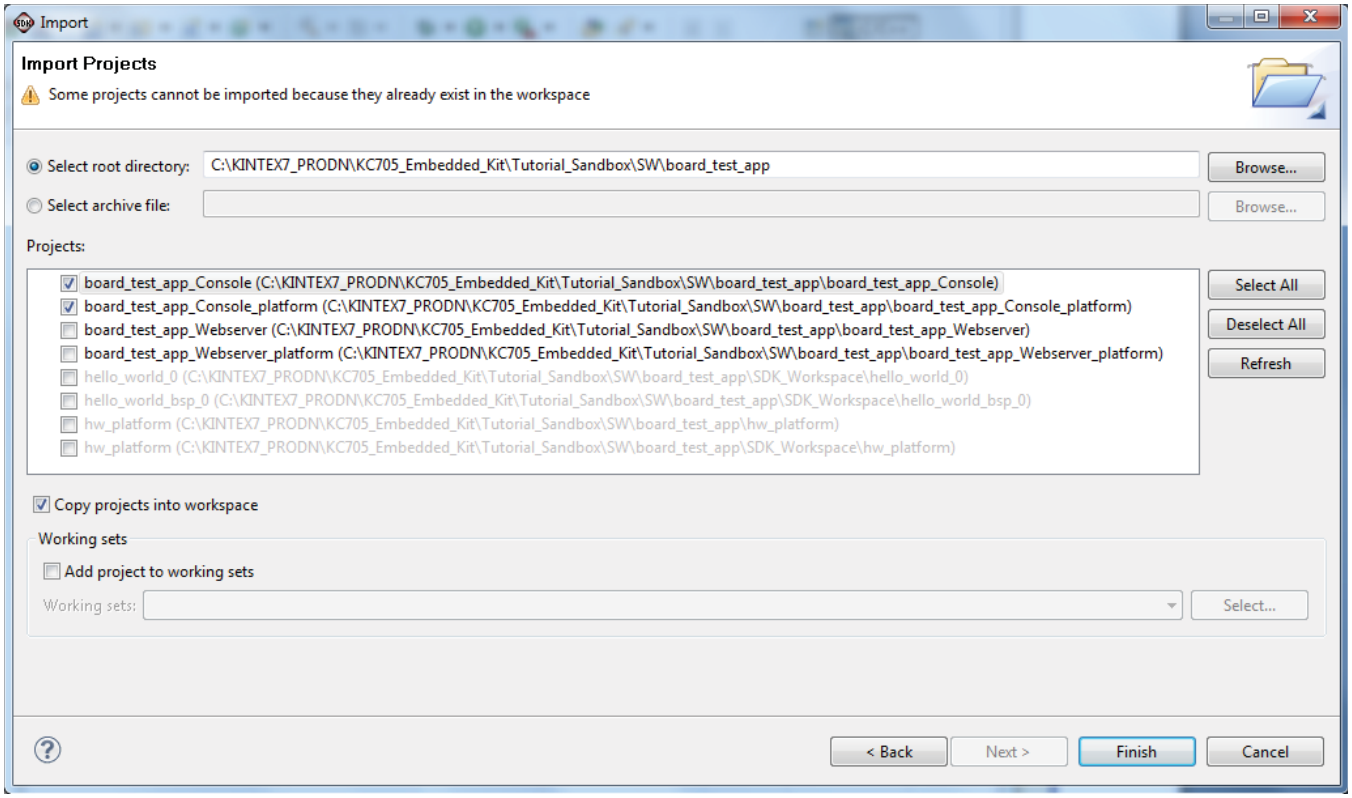
The SDK can import existing SDK projects into an SDK workspace. The application (`board_test_app_Console`) and BSP (`board_test_app_Console_platform`) are imported into the SDK workspace by following these steps:

1. In the SDK, select **File > Import**.
2. In the Import wizard, expand the General folder and select **Existing Projects into Workspace**. Click **Next**.
3. To specify the root directory from which software projects are to be imported, click **Browse...** and specify the location where the software projects are stored. For this reference system, the `board_test_app_Console` and `board_test_app_Console_platform` project are stored under `Tutorial_Sandbox/SW/board_test_app`. That directory should be specified as the root directory. Then click **OK.**
4. After the root directory is specified, the Import wizard shows a list of projects available to import, as shown in Figure 9. Make these selections:
   a. Ensure that the `board_test_app_Console` and `board_test_app_Console_platform` projects are selected. Select only the two projects mentioned. Do not select the remaining projects. They are described in following sections.
   b. The provided hardware platform project (`hw_platform`) is the same as the hardware platform project created in Creating a New BSP and Stand-Alone Software Application Project, page 13. If that section of the tutorial has been completed, the `hw_platform` project appears grayed out because it already has been created in the SDK workspace. If that section of the tutorial has not been completed, select the **hw_platform** project to import.
   c. The `hello_world_0` and `hello_world_bsp_0` projects were created in Creating a New BSP and Stand-Alone Software Application Project, page 13. They appear grayed-out because they already exist in the SDK workspace (as long as that section of the tutorial is complete).

      The project `board_test_app_Webserver` and `board_test_app_Webserver_platform` are practiced in the following section, so do not select these projects.
   d. Select the **Copy projects into workspace** checkbox to make a local copy of the projects in the workspace when the import is done.

   When a BSP is imported into a workspace, the SDK looks for a hardware platform project in the system. When a software application is imported into a workspace, the SDK looks for a hardware platform project and a BSP in the workspace. This is why the `board_test_app_Console` project must be imported into an SDK workspace that includes the `board_test_app_Console_platform` BSP and `hw_platform` project.

Note: The projects in the Import wizard might appear in a different order than shown in Figure 9.



UG915_09_072512

Figure 9: **Import the Board Test Application and Platform**

5. Click **Finish**. SDK imports the selected projects. The BSP and software application are compiled during the import process.
6. After the application and platform are imported and compiled, they appear in the Project Explorer tab, as shown in Figure 10.

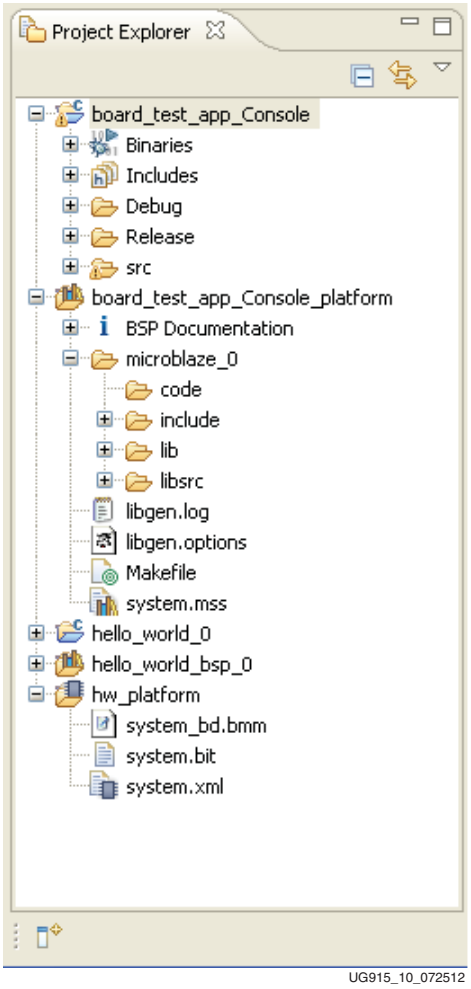


Figure 10: **Board Test Application (Console Mode) and Platform Directory Structure**

Board Test BSP (Console Mode) Settings

The board test BSP is similar to the Hello World BSP created in the Creating the Hello World Software Project and BSP, page 14, except that the board test BSP is set to use the `xilflash` library.

1. To view the BSP settings, right-click **board_test_Console_platform** and select **Board Support Package Settings**. The Supported Libraries Selection in the **Overview** menu should show a checkmark next to the `xilflash` library indicating its use.

   The `xilflash` library provides read, write, erase, lock, and unlock functionality to access a parallel flash device. The `xilflash` library supports Intel and AMD Common Flash Interface (CFI) compliant flash memory devices.

2. Select the **XilFlash** menu, which appears under the **standalone** menu. The `xilflash` library can be configured in this menu.
3. Click **Cancel** to exit the Software Platform Settings.

Running the Board Test Software Application (Console Mode)

After the board test application and platform are imported and built, the application can be run on the KC705 board. Follow these steps to set up the serial terminal console, program the FPGA, and run the board test application:

1. The FPGA programming is done as described in Running the Hello World Software Project, page 15.
2. Set the configuration to Release, as done in Running the Hello World Software Project.
3. Start Run Configurations and select files as done in Running the Hello World Software Project, (step 5) except the project files are:
   - Name: **board_test_app_Console Release**
   - Project: **board_test_app_Console**
   - C/C++ Application: **Release/board_test_app_Console.elf**
4. Set the **STDIO Connection** settings as done in Running the Hello World Software Project (step 6).
5. Click **Run**. The `board_test_app_Console` application displays the test menu to the terminal screen. The output is show in Figure 11.

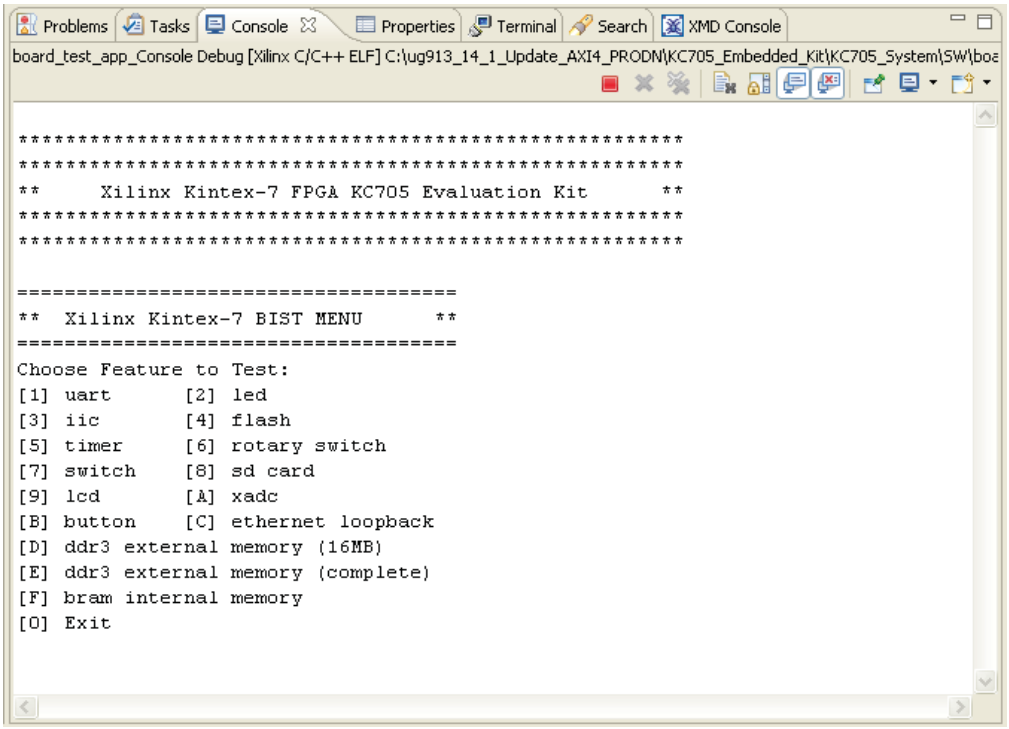


Figure 11: **Board Test Application (Console Mode) Menu**

Tests in the Board Test Application (Console Mode)

This section describes the tests included in the board test application. These tests are not extensive tests. They exercise the basic functionality of the hardware peripherals in the Embedded Kit reference system.

Table 3 describes the tests available in the `board_test_app` application.

Table 3: **Summary of Tests in the Board Test Application**

Menu Selection	Test	Description
1	UART test	Prints `Hello World!` to the terminal.
2	LED test	Flashes the LEDs on the board.
3	IIC test	Uses IIC to write to the EEPROM, read the data back, and compare the data.
4	Flash test	Tests flash memory by erasing a block in flash memory, writing to that block, reading the data back, and comparing the data. Uses the `xilflash` library.
5	Timer test	Initializes a timer counter and allows it to expire and generate an interrupt.
6	Rotary test	Rotary switch that supports detection of left turns, right turns and switch press is tested.
7	Switch test	Reads the value of the switches and prints out the ON/OFF state of each switch.
8	SD Card test	Writing and reading blocks from the Secure Digital High Capacity (SDHC) card is tested. The hardware and software driver IPs are provided by Xylon. **Note**: Back up the SD card content before this test because this test overwrites a few blocks on the SD card.
9	LCD test	Displays a message on the LCD. The LCD supports 2 lines with 16 characters per line.
A	XADC test	Reads current on-chip temperature and on-chip voltages (V_{CCINT} and V_{CCAUX}).
B	Button test	Recognizes each pushbutton press.
C	Ethernet loopback test	Sets the Ethernet PHY to loopback mode and sends and receives a single frame.
D	External memory test (minimal)	Writes and reads test patterns to a small chunk of memory in the DDR3 SDRAM.
E	External memory test (complete)	Writes and reads test patterns to complete memory in the DDR3 SDRAM. This test takes a few minutes to complete.
F	Block RAM memory test	Writes and reads test patterns to internal block RAM memory.
0	Exit	Exits the board test application.

Board Test Software Application and BSP (Webserver Mode)

This section describes the `board_test_app_Webserver` design flow. This section of the tutorial uses the SDK workspace previously created in Stand-Alone Software

Development, page 11.

The design flow for the board test software application (Webserver mode) includes:

- Importing the Board Test Software Application and BSP (Webserver Mode), page 24
- Board Test BSP (Webserver Mode) Settings, page 26
- Running the Board Test Software Application (Webserver Mode), page 29
- Tests in the Board Test Application (Webserver Mode), page 31

Note on the MFS library

The software platform for the `board_test_app_Webserver` contains a built-in memory-based file system (MFS). The MFS image contains the HTML, JavaScript, and SVG files that are used as part of the Web server. These source files are stored in a `./memfs/` folder.

The binary image of the MFS library (`libmfsimage.a`) is part of the source files and linked to the ELF. If the developer would like to make changes to any of these HTML, JavaScript, or SVG files, the files should be modified in the `./memfs/` folder, a new `libmfsimage.a` file should be created, and the copy should be moved to the `./src/` folder.

Refer to these files on how to build the `libmfsimage.a` file:

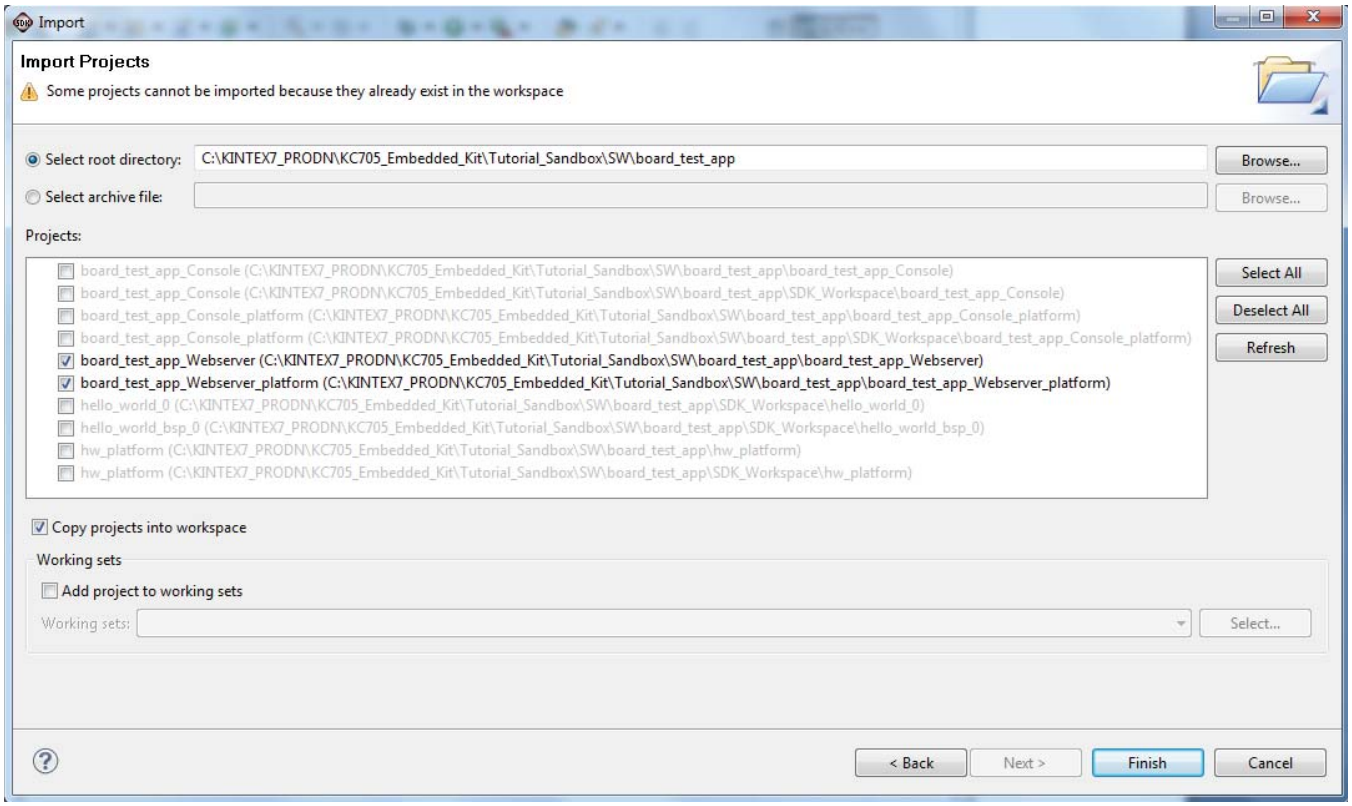
- On Linux: `./memfs/make_libmfsimage`
- On Windows: `./memfs/make_libmfsimage.bat`

Importing the Board Test Software Application and BSP (Webserver Mode)

The SDK can import existing SDK projects into an SDK workspace. The application (`board_test_app_Webserver`) and BSP (`board_test_app_Webserver_platform`) are imported into the SDK workspace by following these steps:

1. Importing the `board_test_app_Webserver` project is similar to the `board_test_app_Console` project described in Importing the Board Test Software Application and BSP (Console Mode), page 19.
2. In the Import wizard, select **board_test_app_Webserver** and **board_test_app_Webserver_platform** projects.

   Note: The projects in the Import wizard might appear in a different order than shown in Figure 12.



UG915_12_072512

Figure 12: **Import the Board Test Application and Platform**

3. Click **Finish**. The SDK imports the selected projects. The BSP and software application are compiled during the import process.
4. After the application and platform are imported and compiled, they appear in the Project Explorer tab as shown in Figure 13.

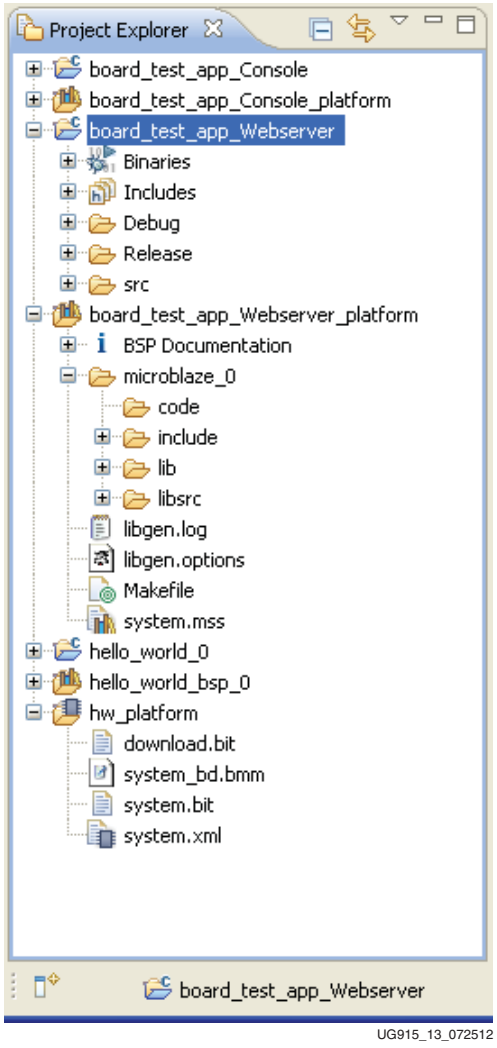


Figure 13: **Board Test Application (Webserver Mode) and Platform Directory Structure**

Board Test BSP (Webserver Mode) Settings

The board test BSP is not like the `board_test_app_Console` BSP created in the Board Test BSP (Console Mode) Settings , page 21. This BSP is based on the xilkernel OS and is set to use the lwip140, Xilinx memory file system (`xilmfs`), and `xilflash` libraries.

1. To view the BSP settings, right-click **board_test_Webserver_platform** and select **Board Support Package Settings**. The Supported Libraries Selection in the Overview menu should show a checkmark next to the **lwip140**, **xilflash**, and **xilmfs** libraries indicating their use.

   The lwip140 library provides the connectivity part of the Web server over Ethernet. The lwip140 is set to SOCKET_API mode.

   The `xilmfs` library supports the MFS on the KC705 board. The web page (`index.html` and associated content files) are stored on the target in the form of an MFS image. When the system starts execution, the `xilmfs` library reads this MFS image and extracts the web page content files.

The `xilflash` library provides read, write, erase, lock, and unlock functionality to access a parallel flash device. The `xilflash` library supports Intel and AMD Common Flash Interface (CFI) compliant flash memory devices.

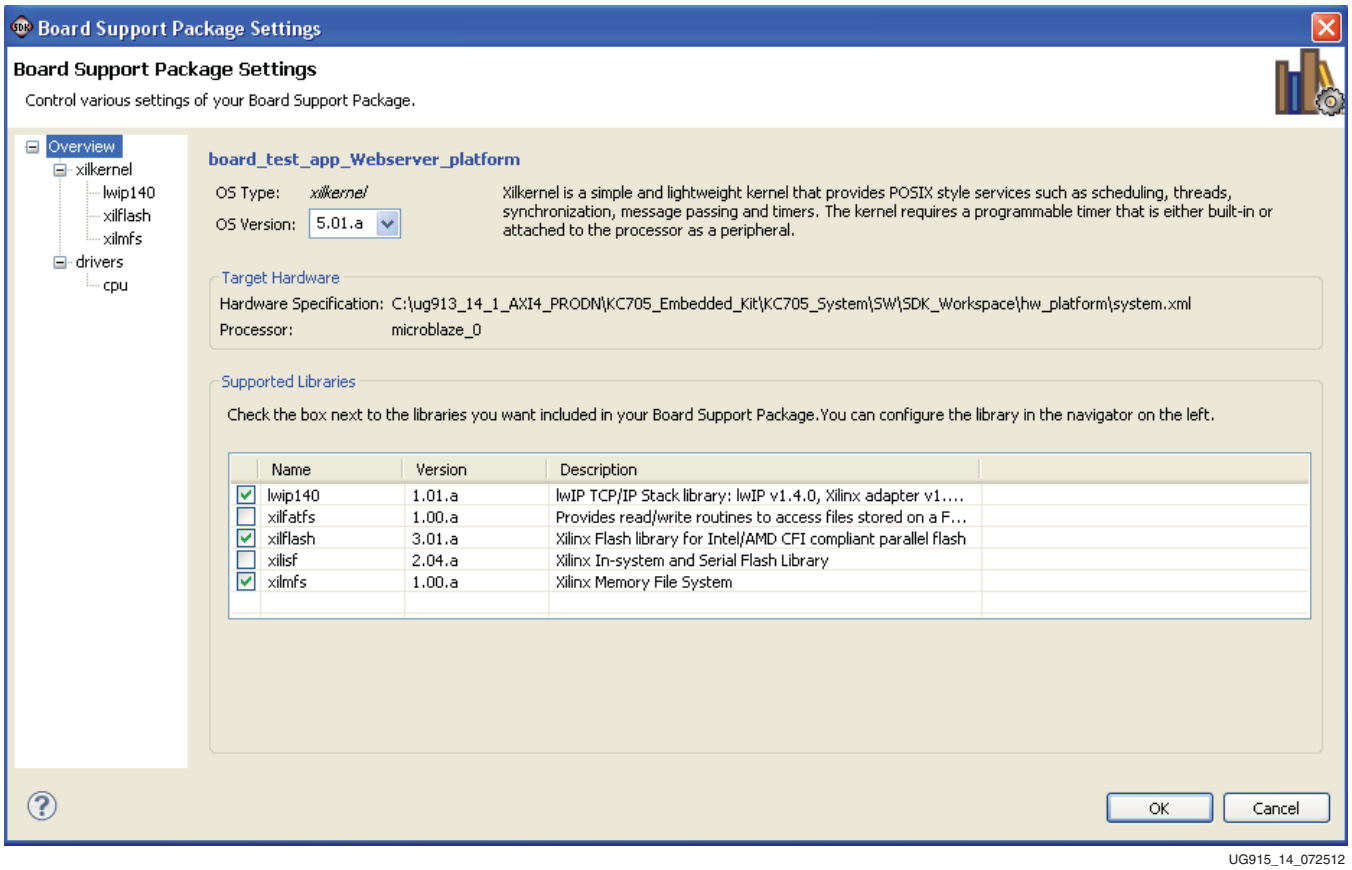


Figure 14: **BSP Overview**

Note: The versions of the supported libraries shown in Figure 14 might vary depending on the SDK tool version.

2. Select the **xilmfs** menu, which appears under the **xilkernel** menu. The **numbytes** (maximum image size) is set to 266000 bytes and base address of the `mfs` image is set to `0xB0000000` (see Figure 15).

Figure 15: **BSP xilmfs Settings**

3. Select the **lwip140** menu, which appears under the **xilkernel** menu. Select the API mode **SOCKET_API.**
4. Select the **xilflash** menu, which appears under the **xilkernel** menu. The `xilflash` library can be configured in this menu.
5. The **xilkernel** settings need a few changes as shown in Figure 16.

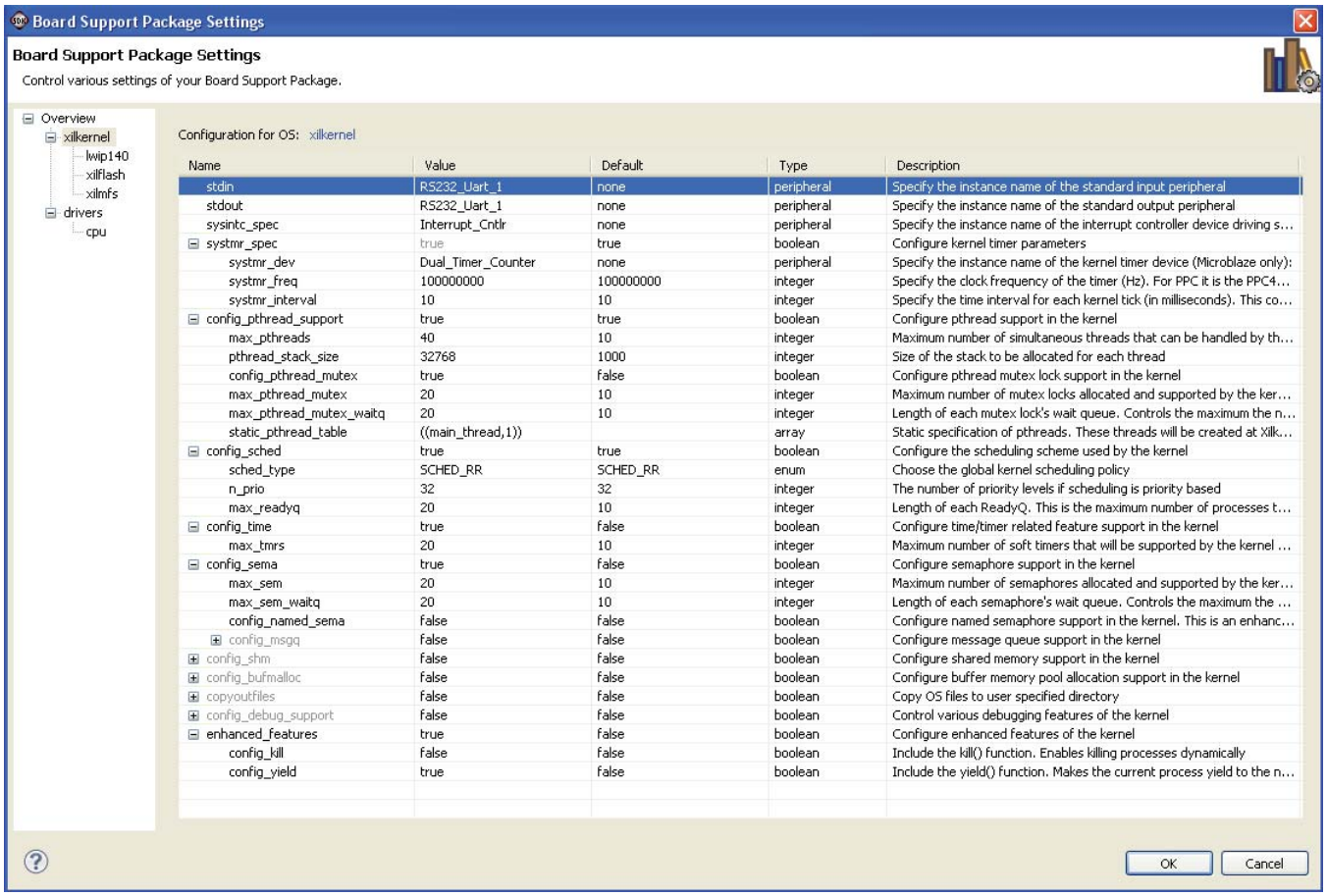


Figure 16: **BSP xilkernel Settings**

6. Click **Cancel** to exit the Software Platform Settings.
7. The imported project comes with all the above settings and the user need not change any settings to start with.

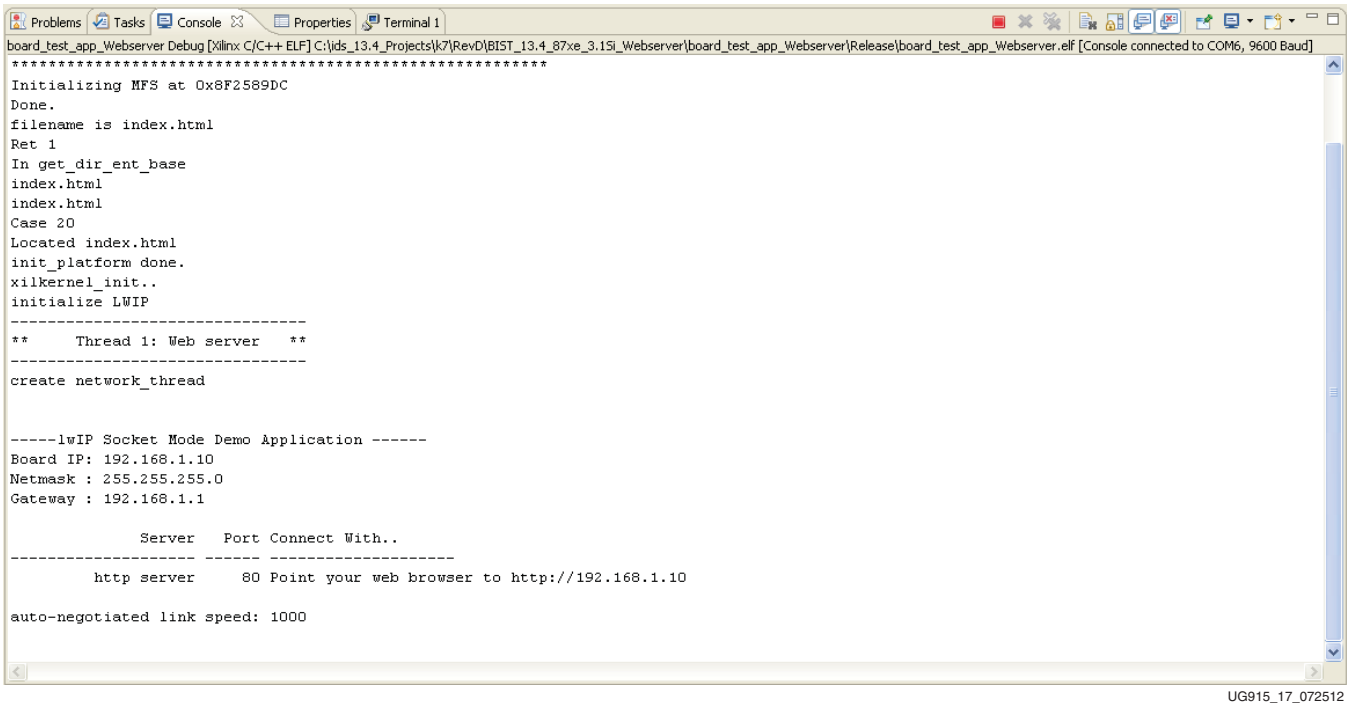
Running the Board Test Software Application (Webserver Mode)

Before executing the application, the user needs to set up the Ethernet connection on the host computer, so that the Web server can execute. The KC705 board is configured to the static IP address of 192.168.1.10.

1. Connect the KC705 board and host computer with an Ethernet cable.
2. On the host computer, set the Ethernet IP address to the static address as follows:
   a. Click **Start > Control Panel > Network Connections > Local Area Connection**.
   b. In the General tab, select **Internet Protocol (TCP/IP)** and click **Properties**.
   c. Select **Use the following IP address**.
   d. Enter this address in IP address area: **192.168.1.100**.
   e. Enter this address in Subnet mask area: **255.255.255.0**.
   f. Click **OK**.

After the board test application and platform are imported and built, the application can be run on the KC705 board. Follow these steps to set up the serial terminal console and web browser, program the FPGA, and run the board test application:

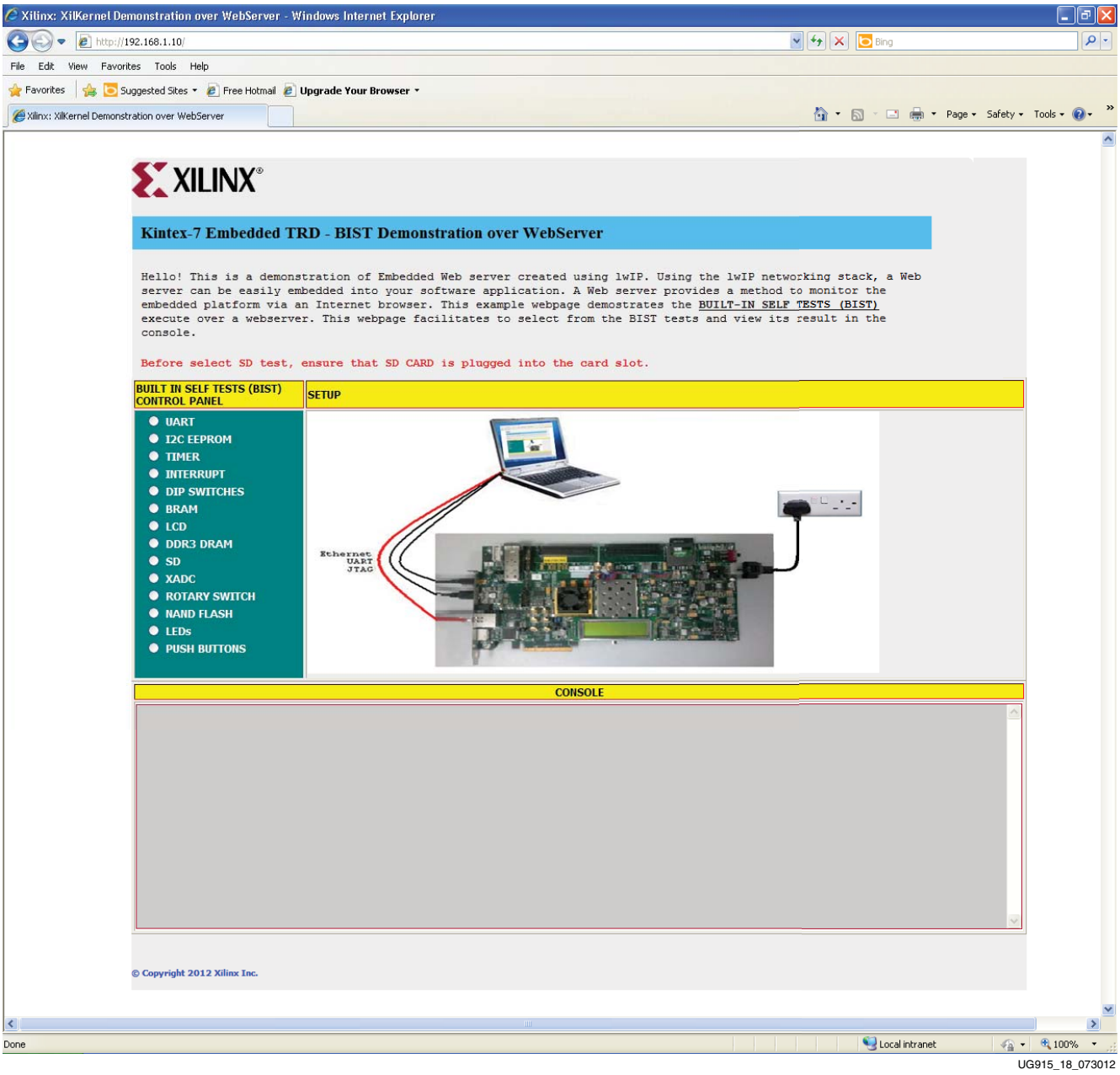
1. The FPGA programming is done as described in Running the Hello World Software Project, page 15.
2. Set the Build Configuration to **Release**, as done in Running the Hello World Software Project.
3. Start Run configuration and select files as done in Running the Hello World Software Project, except the project files are as listed here:
   - Name: **board_test_app_Webserver Release**
   - Project: **board_test_app_Webserver**
   - C/C++ Application: **Release/board_test_app_Webserver.elf**
4. Set the **STDIO** settings as done in Running the Hello World Software Project. Click **Run**. The `board_test_app_Webserver` application displays messages on the terminal screen. The output is shown in Figure 17.



UG915_17_072512

Figure 17: **Board Test Apps (Webserver) Console Output**

5. Wait till the last message `auto-negotiated link speed: 1000` appears. Now the Web server is ready to load the content.
6. On the host computer, start the web browser (Internet Explorer version 8 or 9) and type **http://192.168.1.10** in the address bar, and click **Go**.
7. The Web server loads the content from target to the host computer. It contains the menu options for board test apps and a console screen as shown in Figure 18.



UG915_18_073012

Figure 18: **Web Page Output when Loaded into Browser**

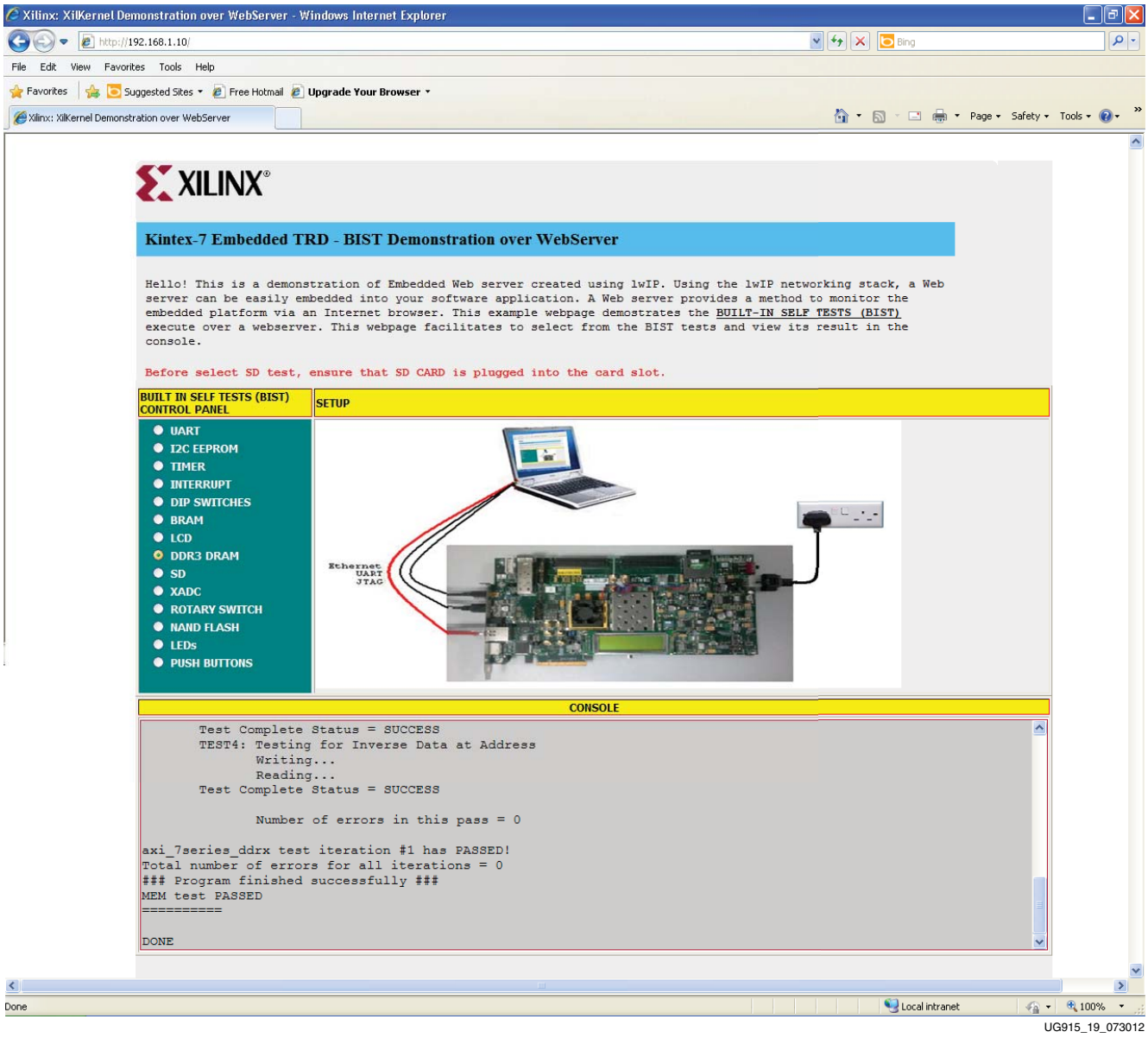
Tests in the Board Test Application (Webserver Mode)

This section describes the tests included in the board test application. These tests are not extensive tests. They exercise the basic functionality of the hardware peripherals in the Embedded Kit reference system.

An example of test execution is described as follows (see Figure 19):

1. Click the radio button **DDR3 DRAM**.
2. The text windows display `Executing DDR3 Test`.

3. After the test is completed, the text window displays the complete test results. DONE appears at the end.
4. Other tests can be executed in a similar fashion.



UG915_19_073012

Figure 19: **Web Page Output at the End of a Test (Example: DDR3 DRAM)**

The test case descriptions mentioned in Table 3 apply here as well.

Debugging Stand-Alone Software Applications

The SDK provides source-level debugging capabilities. This section introduces these debugging capabilities, which include viewing the assembly-level code, viewing registers, setting breakpoints, and stepping into functions.

These steps are included for this exercise on debugging stand-alone software applications:

- Importing the Board Test Application and BSP (Console Mode) for Debugging
- Creating a Debug Run Configuration
- Debugging the Application

Importing the Board Test Application and BSP (Console Mode) for Debugging

The debugging portion of this tutorial uses the `board_test_app_Console` software application and `board_test_app_Console_platform` BSP as the example software application and BSP for debugging. If the `board_test_app_Console` software application and `board_test_app_Console_platform` BSP are not present in the SDK workspace, complete the steps in Importing the Board Test Software Application and BSP (Console Mode), page 19.

Creating a Debug Run Configuration

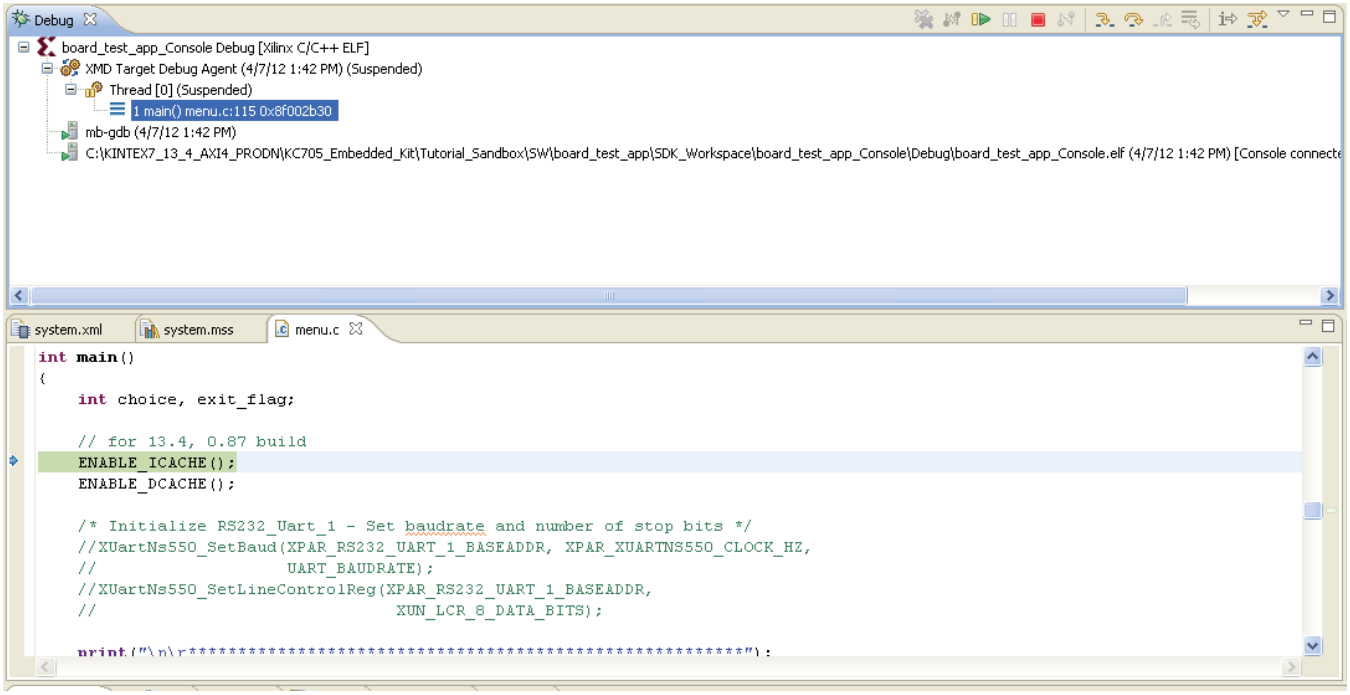
1. To set the application to use the Debug configuration, right-click the **board_test_app_Console** application in the Project Explorer tab and select **Build Configurations > Set Active > Debug**. The `board_test_app_Console` application is rebuilt for the Debug configuration.
2. Program the FPGA as described in Running the Board Test Software Application (Console Mode), page 22.
3. Right-click **board_test_app** and select **Debug As > Debug Configurations**.
4. In the Debug configurations box, select **Xilinx C/C++ ELF** and click the **New** button icon to create a new Debug configuration.
5. Populate the Main tab of the new debug configuration with these selections:
   - Name: **board_test_app_Console Debug**
   - Project: **board_test_app_Console**
   - Build Configuration: **Debug**
   - C/C++ Application: **Debug/board_test_app_Console.elf** to select the executable that was built with the Debug configuration
6. Set the **STDIO** settings as done in Running the Hello World Software Project, page 15.
7. Click **Debug**. The `board_test_app_Console.elf` file executable is downloaded to the board. After the download is successfully completed, the SDK asks for confirmation to switch to the **Debug Perspective**. Click **Yes** in this confirmation box.

Debugging the Application

The SDK provides facilities to debug an application. Some of these debug facilities are described here.

Start Debug

When the debug perspective opens, the processor code is positioned at the beginning of main() and program execution is suspended at the call to ICACHE(), as shown in Figure 20. The call stack is only one deep because the program is only in the main() function.



UG915_20_072512

Figure 20: **Debug Perspective**

Disassembly View

The Disassembly View also shows that execution is suspended at the call to ENABLE_ICACHE() macro expanded to function microblaze_enable_icache(). If the Disassembly View tab is not visible, it can be opened by selecting **Window > Show View > Disassembly**.

The example Disassembly View in Figure 21 indicates that assembly-level execution is suspended at the instruction address.

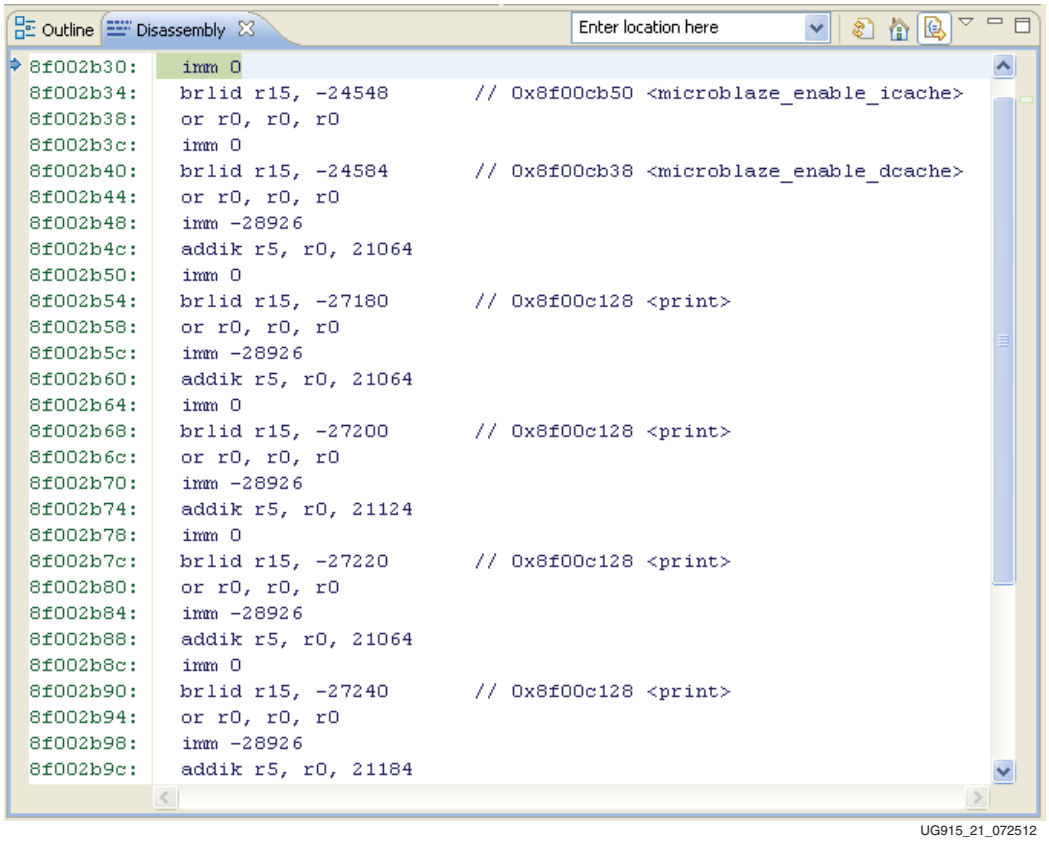


Figure 21: **Disassembly View**

Registers View

In the Registers View shown in Figure 22, the RPC register (the program counter) contains the address of the instruction where the program is suspended. If the Registers View tab is not visible, select **Window > Show View > Registers**. Click the **+** preceding **Main** to expand and view the registers.

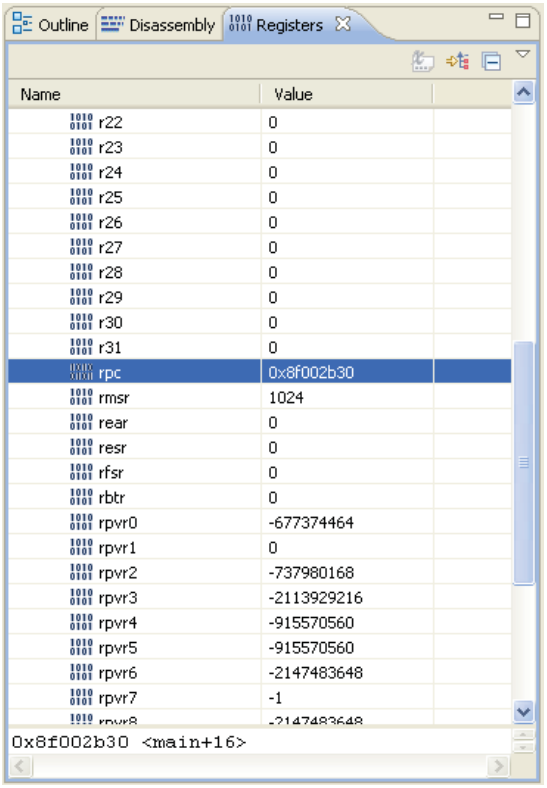
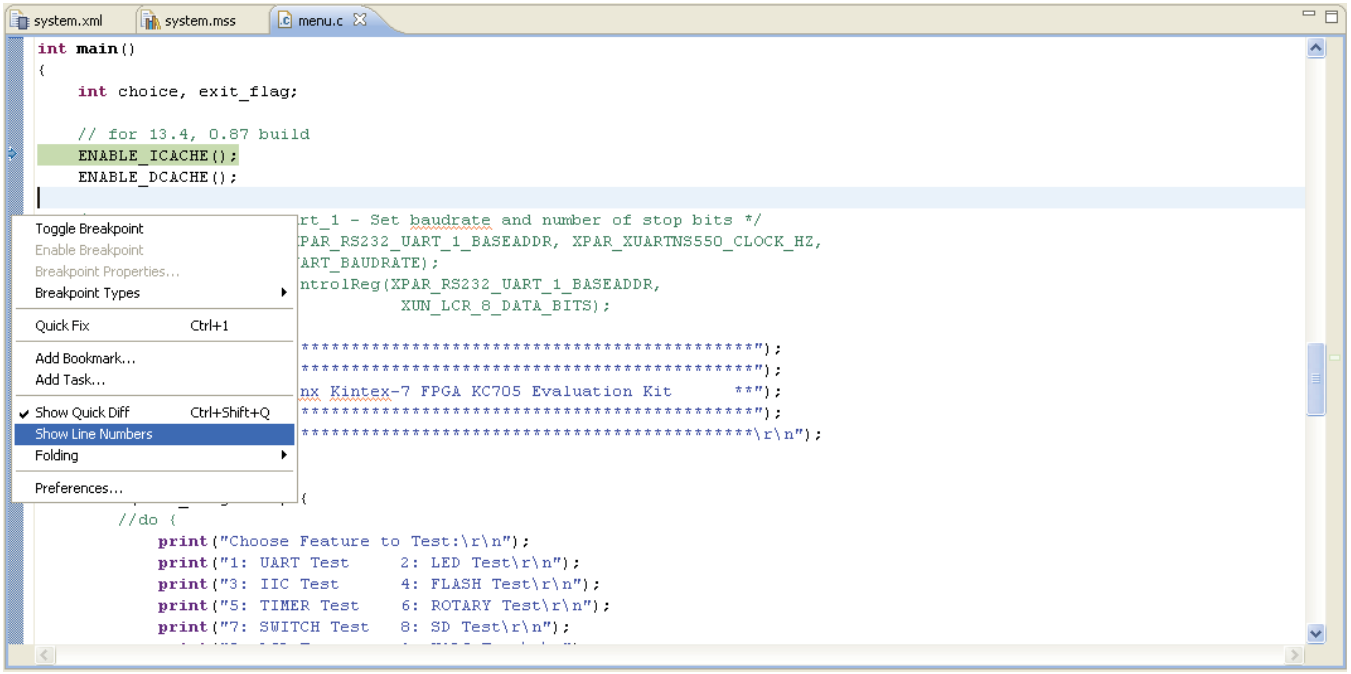


Figure 22: **Registers View**

Line Number View

In the **menu.c** file, right-click in the left margin and select **Show Line Numbers** (as shown in Figure 23) to see the line numbers in the application source, if the numbers do not already appear.



UG915_23_072512

Figure 23: **Show Line Numbers**

SingleStep/StepOver Execution:

1. Select **Run > Step Over** or click the **Step Over** icon near the Debug tab to move to the ENABLE_DCACHE() function without going into the ENABLE_ICACHE() function. Program execution has now suspended at new address location (seen in Disassembly view/Register view).
2. Step over all functions until the debugger reaches the first while () loop of main () function in `menu.c`. The application has started to output text to the serial terminal console. Program execution has suspended at a new address location (seen in the Disassembly and Registers views).

Breakpoints

1. Scroll down in the `menu.c` file to the switch statement.
2. Double-click in the margin of the `menu.c` window next to the switch statement. This sets a breakpoint at the switch statement.
3. Confirm that the breakpoint is set by checking the Breakpoints View tab. If the Breakpoints View tab is not visible, select **Window > Show View > Breakpoints**.
4. Select **Run > Resume** or click the **Resume** icon near the Debug tab to resume program execution. Observe the output text in the Console that displays the menu of test choices.
5. The program waits for user input into the serial terminal console. Enter choice **1** into the serial terminal for the UART Test and press the **Enter** key.
6. The program runs to the breakpoint. Program execution stops at the new address location (seen in Disassembly and Registers views).

View Variables

The switch statement uses the variable named `choice`. To view the value of the variable, hover the mouse cursor over it (see Figure 24). If the user input is **1** as mentioned above, the current value of `choice` is `49`, which is the ASCII decimal value of the character *1*.

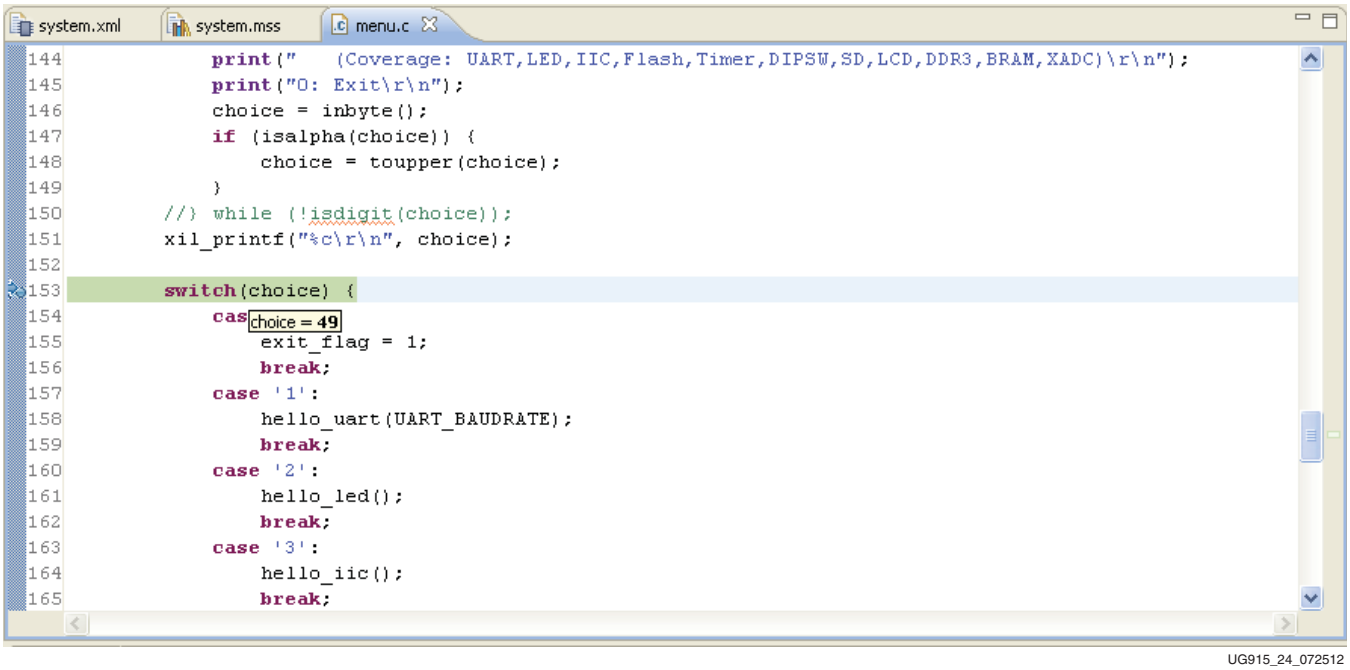


Figure 24: **Value of a Variable**

Values of variables can also be viewed in the Variables View tab, as shown in Figure 25. If the Variables View tab is not visible, select **Window > Show View > Variables**. This view also shows that the `choice` variable currently has a value of *49*. The `exit_flag` variable currently has a value of *0*.

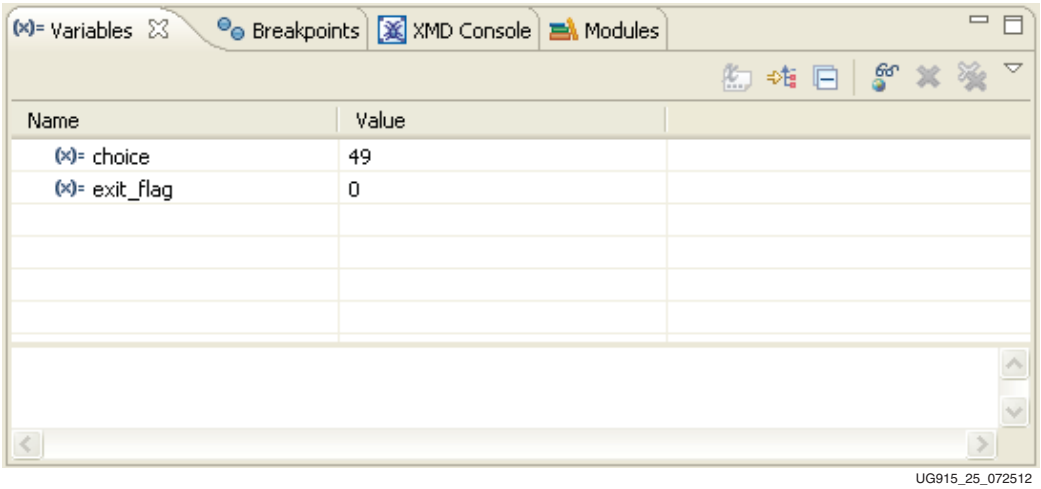


Figure 25: **Variables View**

Step over the switch statement to reach the hello_uart() function call. Program execution has now suspended and the Disassembly View is updated with the new location address.

View Constant Values

Highlight **UART_BAUDRATE** with the mouse and right-click. Select **Open Declaration** as shown in Figure 26.

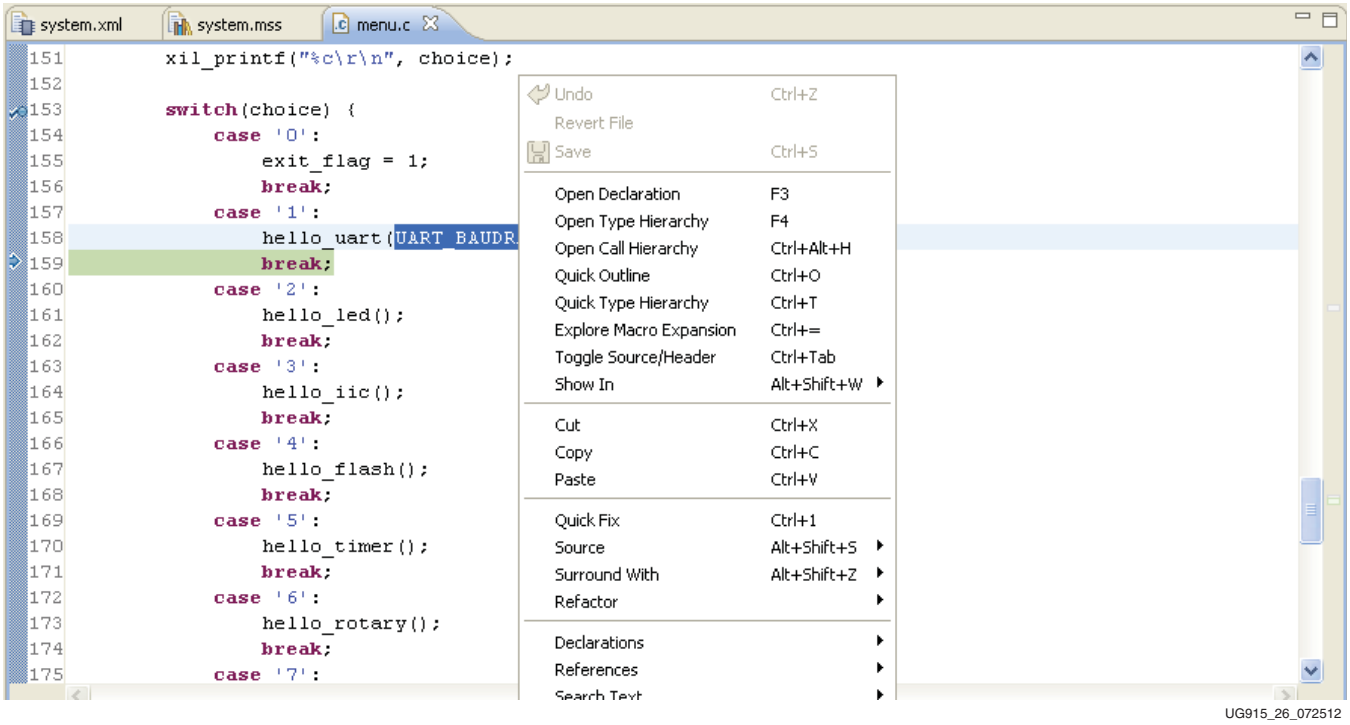


Figure 26: **Open Declaration of UART Parameter**

The `menu.h` file opens with **UART_BAUDRATE** highlighted. As shown, the UART baud rate is 9600. Close the `menu.h` file.

Continue Execution

1. Select R**un > Step Into** or click the **Step Into** icon near the Debug tab to move into the `hello_uart ()` function. The `hello_uart.c` file opens and shows that the program is now in the `hello_uart()` function. The call stack is now two deep because `main()` called `hello_uart()` to get to the current program location.
2. Select **Run > Resume** or click the **Resume** icon near the Debug tab and the Console displays the **board_test_app** menu. The program is again waiting for user input.

End of Debug

1. Enter choice **0** into the serial terminal to choose the **Exit** option.
2. Resume program execution to run the program to conclusion. When the program is finished executing, the Debug View tab shows that the program has suspended in an exit routine, which occurs when running under the control of the debugger.
3. Observe the output text in the terminal serial program that indicates that the program has run and exited.

4. To terminate the Debug session, click the **Terminate** icon by the Debug tab or select **Run > Terminate**.
5. To exit and close the Debug perspective, select **Window > Close Perspective** from the Debug perspective.

Profiling Stand-Alone Software Applications

SDK supports the capability to profile a program running on embedded hardware. The profiling method is software-intrusive meaning the execution flow of the program is altered so that the profiling data can be collected.

These steps on profiling stand-alone software applications are included for this exercise:

- Importing the Board Test Application and BSP for Profiling
- Hardware System Requirements for Profiling
- Creating a BSP for Profiling
- Setting Up the Software Application
- Creating a Profile Run Configuration
- Profiling the Application

Importing the Board Test Application and BSP for Profiling

The profiling portion of this tutorial uses the `board_test_app_Console` software application and `board_test_app_Console_platform` BSP as the example software application and BSP for profiling. If the `board_test_app_Console` software application and `board_test_app_Console_platform` BSP are not present in the SDK workspace, complete the steps in Importing the Board Test Software Application and BSP (Console Mode), page 19.

Hardware System Requirements for Profiling

Profiling an application requires that interrupts are raised periodically to sample the program counter value. The reference system provided with this Embedded Kit includes the axi_timer core with the timer interrupt signal connected to the processor through an interrupt controller. This configuration satisfies the hardware requirements for program profiling.

Creating a BSP for Profiling

The BSP must have profiling support enabled. An existing BSP can be changed to have profiling support. To create a new BSP that has profiling support:

1. The `board_test_app_Console_platform` BSP is used, but it must be modified. To preserve the original application, create a new BSP that is based off of `board_test_app_Console_platform` as follows:
   a. Right-click **board_test_app_Console_platform** and select **Copy**.
   b. Right-click **board_test_app_Console_platform** and select **Paste**.
   c. Give the new application the name **board_test_app_Console_platform_profile** and click **OK**. This creates a copy of the `board_test_app_Console_platform` application called `board_test_app_Console_platform_profile` in the workspace.

2. After the new BSP is created, right-click **board_test_platform_app_Console_profile** and select **Board Support Package Settings**.
3. Select the **standalone** menu. Make these changes:
   a. Expand the menu at **enable_sw_intrusive_profiling**.
   b. Set **enable_sw_intrusive_profiling** to **true**.
   c. Set **profile_timer** to **Dual_Timer_Counter**.

   The Board Support Package Settings are shown in Figure 27.

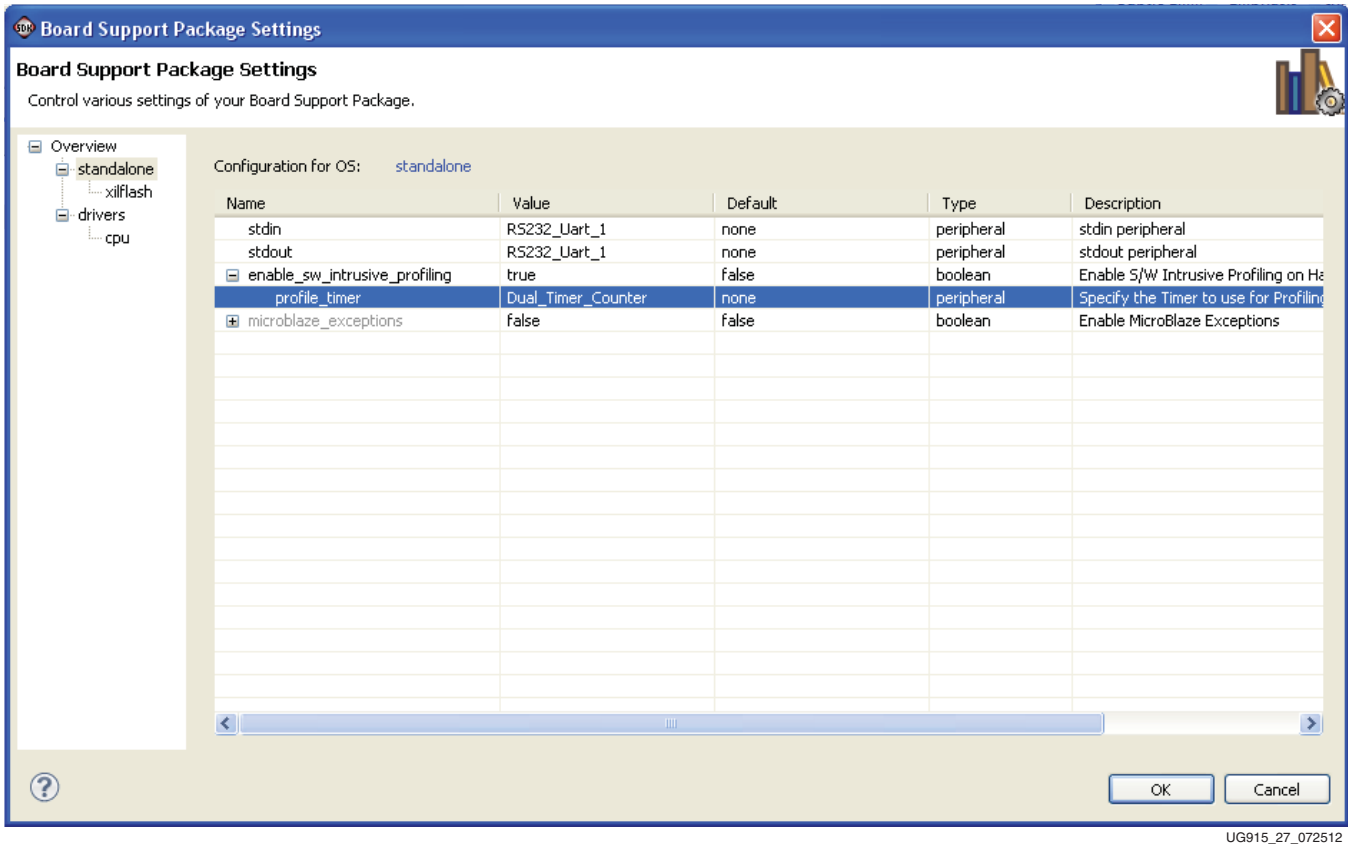


Figure 27: **Profiling Board Support Package Settings**

4. Click **OK** to save changes and exit the Board Support Package Settings window. SDK rebuilds the software platform.

Setting Up the Software Application

The software application project to be used for the profiling exercise is the `board_test_app_Console` application project provided with the Embedded Kit. Follow these steps to prepare the `board_test_app_Console` application for profiling:

1. The `board_test_app_Console` application is used, but must be modified. To preserve the original application, create a new software application project that is based off of `board_test_app_Console` by following these steps:
   a. Copy **board_test_app_Console** and paste it with name **board_test_app_Console_profile** (similar to step 1 under Creating a BSP for Profiling, page 40).

2. The `board_test_app_Console_profile` application needs to be associated with the BSP that was created for profiling. To do this, follow these steps:
   a. Right-click **board_test_app_Console_profile** and select **Change Referenced BSP.**
   b. Select **board_test_platform_app_Console_profile**.
   c. Click **OK**.

After compilation, the Project Explorer tab looks similar to Figure 28.

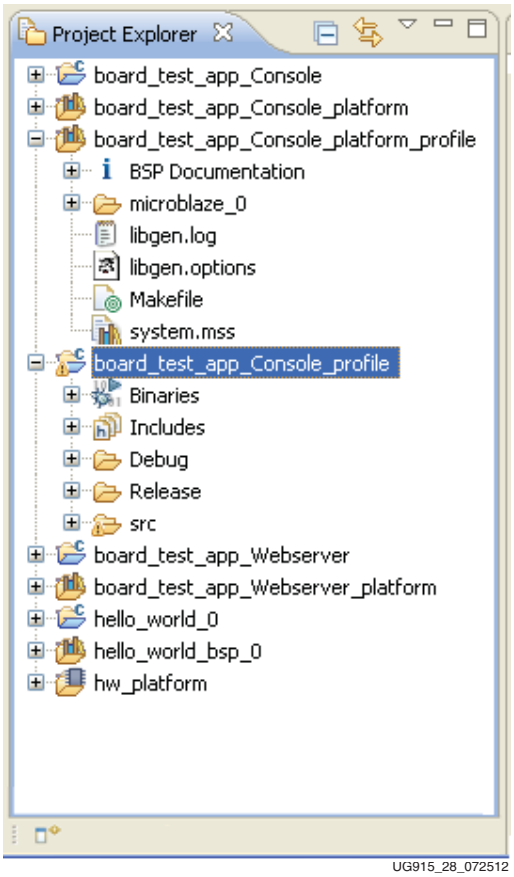


Figure 28: **Profiling Application Directory Structure**

3. The `board_test_app_Console_profile` software application must be compiled for profiling, with the `-pg` switch enabled. Follow these steps:
   a. Right-click **board_test_app_Console_profile** and select **Properties**.
   b. Expand **C/C++ Build** and click the **Settings** menu.
   c. Under **MicroBlaze gcc compiler**, click **Profiling**.
   d. In the **Profiling** menu, select the checkbox for **Enable Profiling (-pg)**.
   e. Click **OK** and the application is rebuilt.

Creating a Profile Run Configuration

The next step to preparing the application project for profiling is to create a profile Debug configuration. This configuration enables and sets up profiling for the application. After profiling is enabled and set up, the application can be run and profiled.

1. Program the FPGA as described in Running the Board Test Software Application (Console Mode), page 22.
2. Right-click **board_test_app_Console_profile** and select **Run As > Run Configurations**.
3. In the Run configurations box, select **Xilinx C/C++ ELF** and click the **New** button icon to create a new Run configuration.
4. Populate the Main tab of the new debug configuration with these selections:
   - Name: **board_test_app_Console_profile Debug**
   - Project:**board_test_app_Console_profile**
   - Build Configuration: Leave it default (**Debug** or **Use Active**)
   - C/C++ Application: **Debug/board_test_app_Console_profile.elf** to select the executable that was built with the Debug configuration
5. Set the **STDIO** settings as done in Running the Hello World Software Project, page 15.
6. Click the **Profile Options** tab, which allows you to specify the profiling parameters. The profiling parameters are described in Table 4. Use these selections:
   a. Enable Profiling: Click the checkbox to select **Enable Profiling**
   b. Sampling Frequency (Hz): **10000**
   c. Histogram Bin Size (words): **4**
   d. Scratch memory address to collect profile data: **0x90000000.**

Table 4: **Profiling Parameters**

Parameter	Description	Setting
Sampling frequency (Hz)	The frequency in which timer interrupts are generated.	For this example, the default value is used. When a higher frequency is set, more data samples are obtained, which provides more accurate profiling results, but is more software-intrusive because of the number of interrupts resulting in interrupt calls to collect profiling data.
Histogram bin size (words)	The program text region is divided into multiple bins. The bin size determines how accurate the program counter location is in the data sample.	In this example, the default value is used. A smaller bin size divides the program text region into a large number of small bins. This smaller bin size allows a more accurate sample, but requires a large amount of memory space to store the profile data. A larger bin size is less accurate, but requires less memory space to store profile data.
Scratch memory address to collect profile data	The scratch memory address indicates where in memory the profile data is stored. This memory must lie outside the program memory area and should not be overwritten.	In this example, the scratch memory is set to an address in the internal block RAM.

7. Click **Run**. The application output appears in the Console.
8. The program waits for user input into the serial terminal console. Enter choice **C** to run the Ethernet test.
9. After the Ethernet test is complete, enter choice **0** to exit the application.

Profiling the Application

1. After the application completes, SDK confirms that the profiling results have been saved in `/board_test_app_Console_profile/Debug/gmon.out`. Click **OK** in the confirmation box. The `gmon.out` file appears in the Project Explorer tab, as shown in Figure 29. To view the profiling results, double-click the `gmon.out` file in the Project Explorer tab.

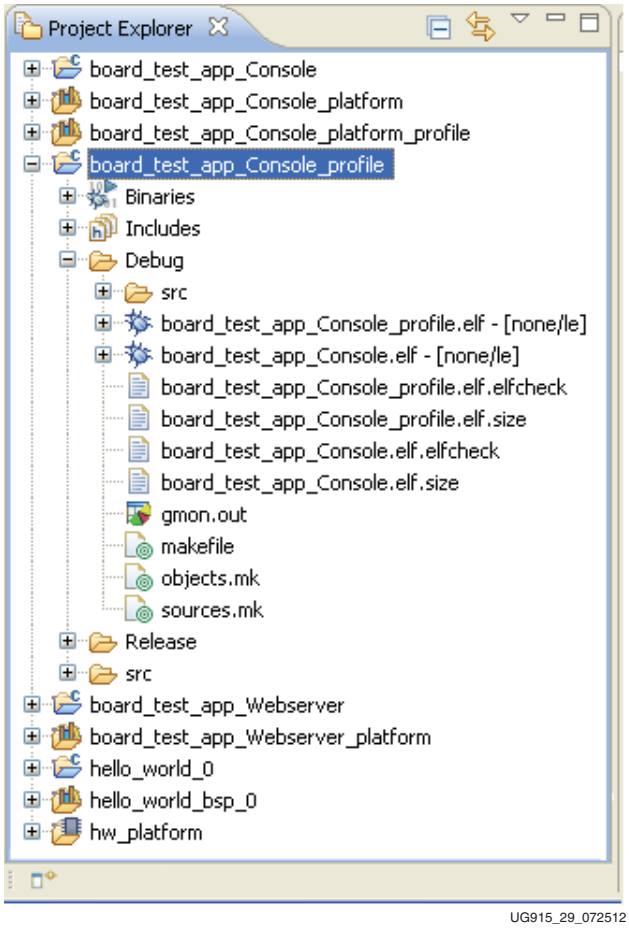
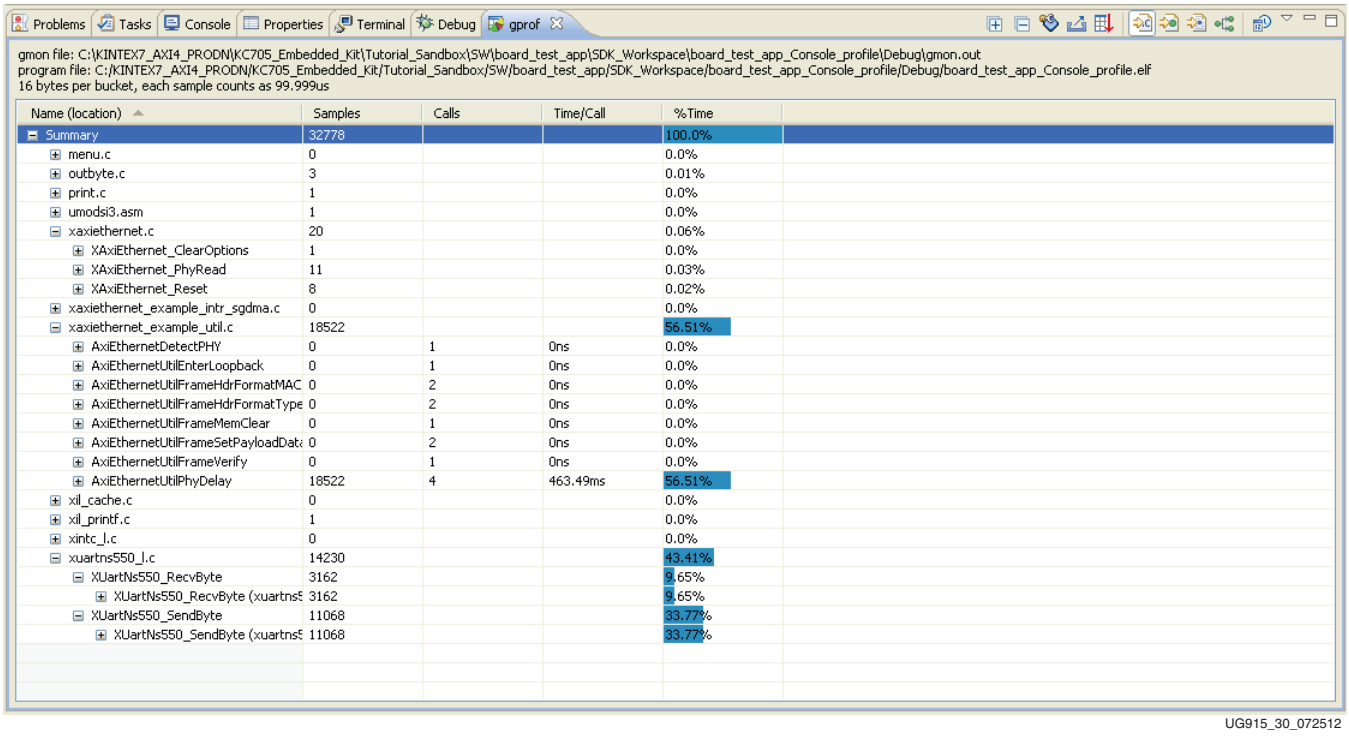


Figure 29: **Profiling Results File**

2. When the **Gmon File Viewer** selection appears, click **Workspace** and select **board_test_app_Console_profile.elf** under **board_test_app_Console_profile/Debug**. This was the ELF file that was used to create these profiling results. Click **OK** twice to complete the selection.
3. The Xilinx Profiling view opens at the bottom of the SDK window with the profiling results. Table 5 summarizes the different columns in the Xilinx Profiling View. The profiling results look similar to Figure 30.

Table 5: **Columns in the Xilinx Profiling View**

Column	Description
Name	Name of the function and the file in which it resides.
Samples	Number of times the profile timer interrupt handler detected that the program executed the corresponding function.
Calls	Number of calls made to the corresponding function.
Time/Call	Time usage per function call invocation.
% Time	Percentage of time spent in the function.



UG915_30_072512

Figure 30: **Profiling Results**

4. Expand **xaxiethernet_example_util.c**. For the Ethernet test, calls to functions in the **xaxiethernet_example_util.c** file take a large percentage of time. The largest percentage of time is taken by the four calls made to the `AxiEthernetUtilPhyDelay` function. This function gives the Ethernet PHY time to recover after resetting the PHY or updating the PHY modes.
5. Expand **xuartns550_l.c**. Because this is a menu-driven application, calls to functions in the **xuartns550_l.c** file take a large percentage of time for most tests in the `board_test_app` application. The percentage of time for the `XUartNs550_RecvByte` call depends on how long the application was waiting for user input.
6. Look through the profiling results. When finished, close the Xilinx Profiling view by clicking the **X** in the **gprof** tab.

Video Demonstration with xilkernel and lwip

The KC705 Embedded Kit provides a video demonstration that leverages Xilinx IPs capabilities (such as VDMA) and demonstrates the benchmarking of AXI-MM that uses xilkernel and lwip. The steps for building the video demonstration are summarized here and explained in detail in the rest of this section.

1. Creating a New SDK Workspace for the Video Demonstration, page 46

   Includes steps on how to create a new SDK workspace for the benchmarking demonstration. All other sections on the video demonstration use the SDK workspace created in this section.

2. Adding a Local Repository, page 47

   Describes the steps necessary to add a local repository to the SDK workspace. This is required because the video demonstration uses Xylon IP core for display engine (CVC) and Xilinx `xilmfs` drivers with additional utilities.

3. Importing the Hardware Platform Project, Video Demonstration Application, and BSP, page 48

   Details how to import the provided video demonstration projects into an SDK workspace.

4. Video Demonstration BSP, page 49

   Describes the BSP for the video demonstration software.

5. Video Demonstration Application, page 50

   Describes the video demonstration software application.

6. Video Demonstration Hardware Setup, page 50

   Details the hardware setup requirements to run the video demonstration.

7. Running the Video Demonstration, page 50

   Includes steps on how to run the video demonstration using the SDK.

8. Interacting with the Video Demonstration, page 51

   Describes the user interface for the video demonstration.

Creating a New SDK Workspace for the Video Demonstration

To avoid confusion with the files for stand-alone software development, this tutorial assumes that a new SDK workspace is used for the benchmarking demonstration files.

Completion of this section is required by all other benchmarking demonstration sections of this tutorial.

Follow these steps to create a new SDK workspace for the benchmarking demonstration sections of this tutorial:

1. Launch SDK, and create new workspace as described in Creating a New SDK Workspace for Software Development, page 12.
2. Specify the SDK workspace as **Tutorial_Sandbox/SW/Video_Demo/SDK_Workspace**.
3. Click **OK** in the Workspace Launcher.

Adding a Local Repository

The hardware platform for the video demonstration uses two local drivers:

- The `xilmfs` core supports additional utilities for MFS.
- The logiCVC core is third party IP core (logiCVC) from Xylon that supports the video display engine. The video demonstration software application uses a driver (from the same vendor) corresponding to the logiCVC to access the logiCVC core.

These two drivers are stored in a repository. The location of that repository must be added to the SDK workspace as follows:

1. In the SDK, select **Xilinx Tools > Repositories**.
2. Click **New...** by the Local Repositories box.
3. Browse to **Tutorial_Sandbox/SW/Video_Demo/repository** and click **OK** to add the directory as a repository.
4. The Repositories Preferences window is shown in Figure 31. Click **OK** to complete the selection.

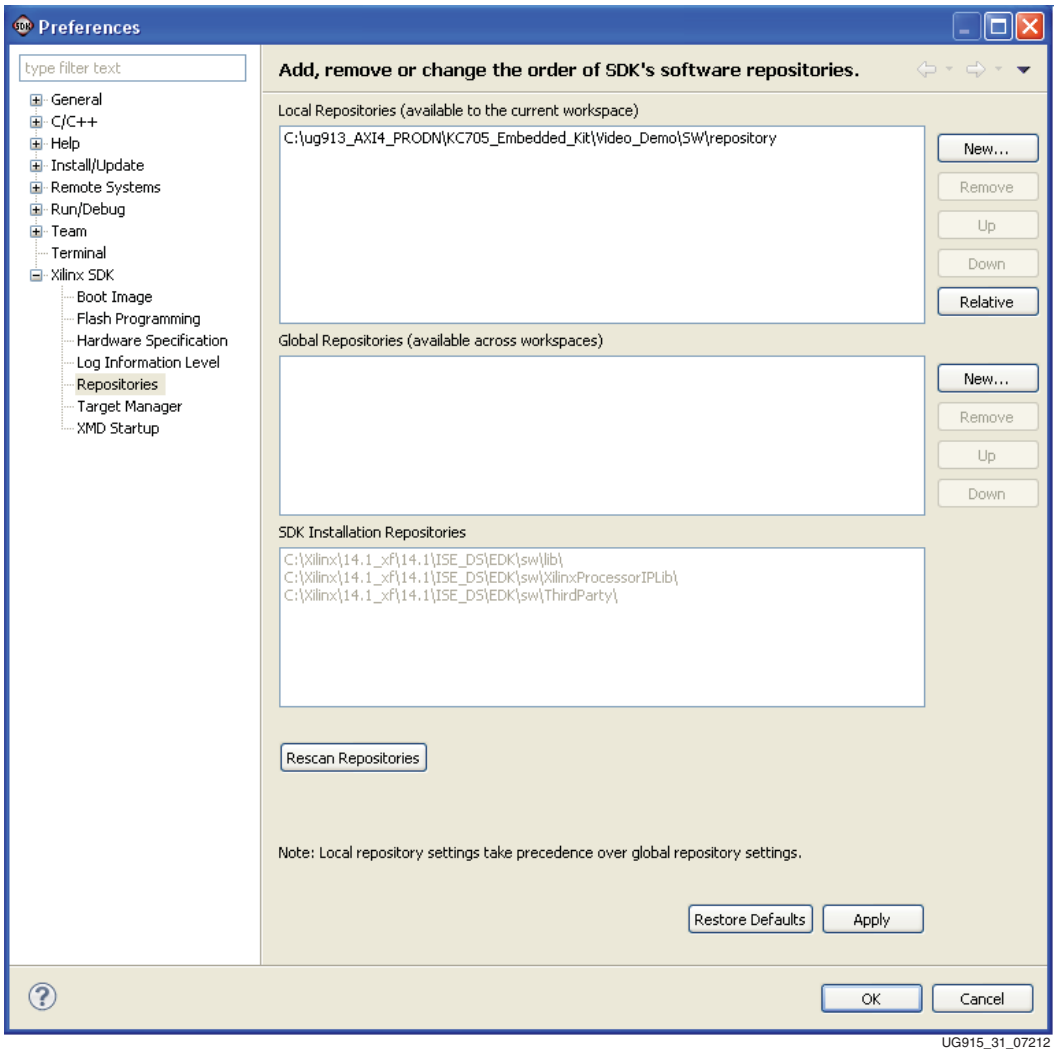


Figure 31: **Repositories Preferences**

Importing the Hardware Platform Project, Video Demonstration Application, and BSP

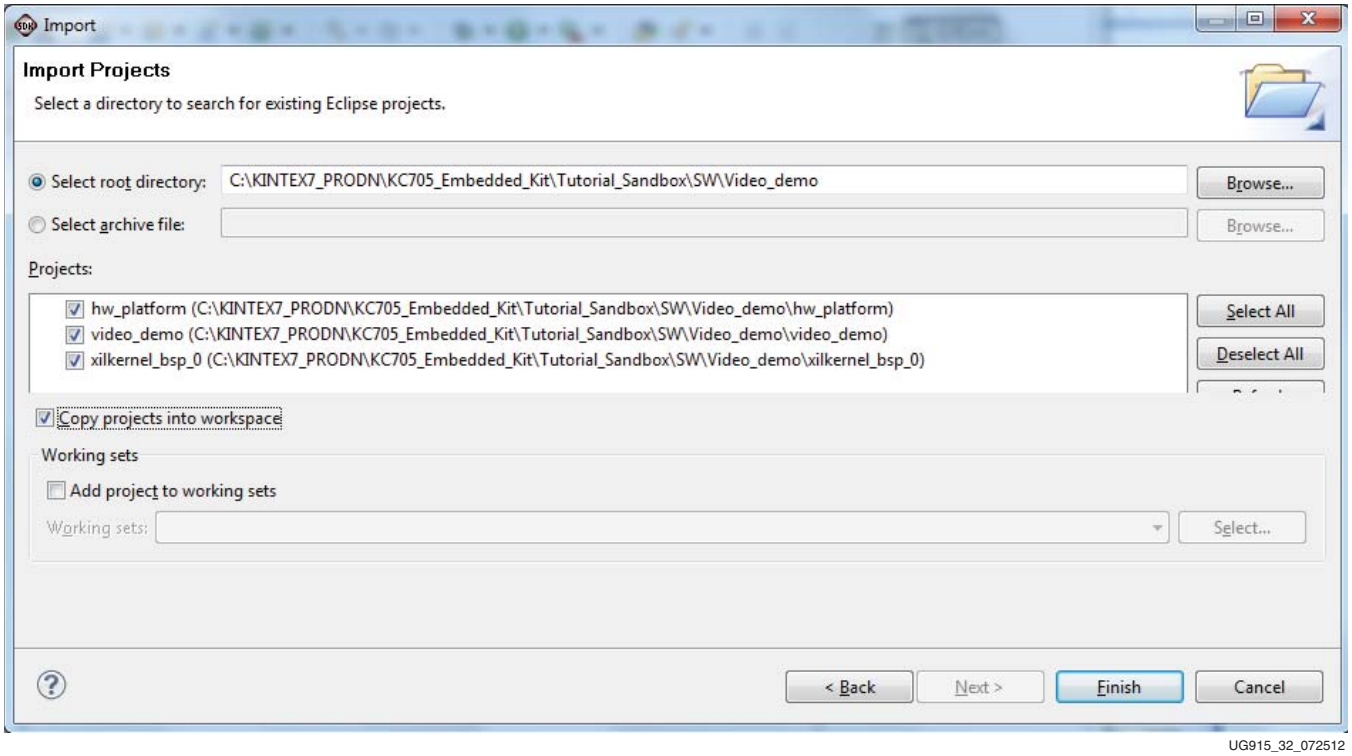
The hardware platform project, video demonstration application, and related BSP are provided with the Embedded Kit. For the video demonstration sections of this tutorial, the hardware platform project uses the hardware specification exported from the hardware system located in `Video_Demo/HW`.

Import the hardware platform project (`hw_platform`), video demonstration application (`Video_Demo`) and platform (`xilkernel_bsp_0`) into the SDK workspace as follows:

1. Importing the `video_demo` project is similar to importing the `board_test_app_Console` project described in Importing the Board Test Software Application and BSP (Console Mode), page 19.
2. To specify the root directory, click **Browse...** and select **Tutorial_Sandbox/SW/Video_Demo**.

   When a BSP is imported into a workspace, the SDK looks for a hardware platform project in the system. When a software application is imported into a workspace, the SDK looks for a hardware platform project and a BSP in the workspace. This is why the `video_demo` project must be imported into the SDK workspace at the same time as the `xilkernel_bsp_0` BSP and `hw_platform` project.

   Note: The projects in the Import wizard might appear in a different order than shown in Figure 32.



UG915_32_072512

Figure 32: **Import the Video Demonstration Application, BSP, and Hardware Platform**

3. Click **Finish**. SDK imports the selected projects. The BSP and software application are compiled during the import process.

4. After the hardware platform project, software application, and BSP are imported and compiled, they appear in the Project Explorer tab, as shown in Figure 33.

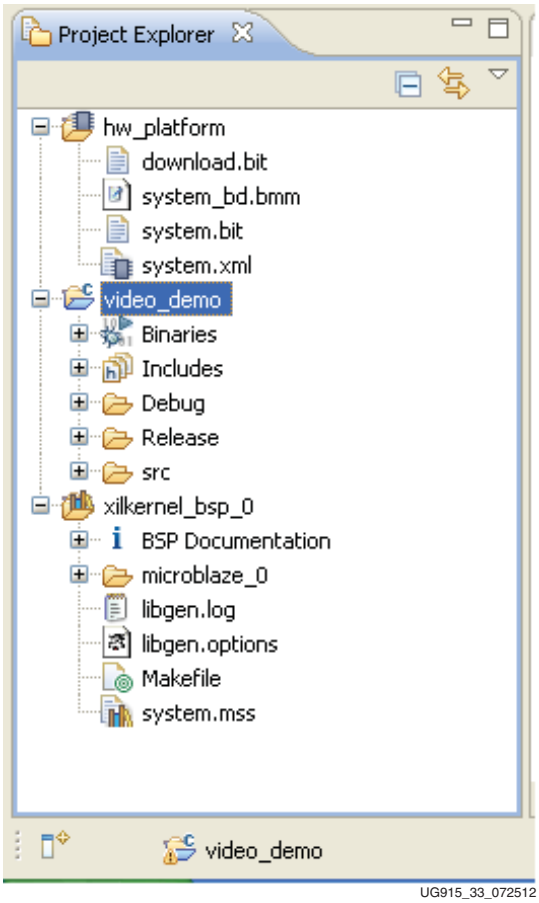


Figure 33: **Video Demonstration Application and BSP Directory Structure**

Video Demonstration BSP

The BSP for the video demonstration uses xilkernel. Xilkernel is a simple and lightweight kernel that provides services such as scheduling, threads, synchronization, and timers. Xilkernel requires a timer and the benchmarking demonstration hardware platform includes the `AXI_Timer` peripheral to suit this purpose.

The BSP also includes the `lwip TCP/IP` stack library. The Web server portion of the video demonstration application code uses the `lwip` socket API.

The Xilinx memory file system (`xilmfs`) library is included in the BSP. This library is used to store files in the memory of the KC705 board that can be accessed by the Web server.

The third party libraries for logiCVC and `xyl_oslib` (from Xylon) are also included in the BSP. These two libraries support the driver functionality of the logiCVC display IP core. The logiCVC carries out actual driver functionality of the logiCVC IP core, and `xyl_oslib` contain the OS wrappers that are needed by the logiCVC driver. To view the BSP settings, right-click **xilkernel_bsp_0** and select **Board Support Package Settings**. After browsing the settings, click **Cancel** to exit the Software Platform Settings.

Video Demonstration Application

The video demonstration application uses a Web server to display the video selection menu, video stream controls, and plot various graphs such as throughput data, on-chip temperature, and on-chip voltages (V_{CCINT} and V_{CCAUX}). The Web server uses the `lwip` socket API.

- There is one main thread that listens on the HTTP port (80) for connections. A new thread is spawned to process a request from a connection. Requests can be GET or POST requests.
- A performance monitor IP and its associated driver are integrated into this application. This IP and driver probe the AXI-MM interface periodically and collect the performance (read and write throughput) numbers. A separate thread is spawned that interfaces to this driver and stores the performance data.
- One more thread is spawned that handles the various video patterns (blend, move around) on a regular basis.

For GET requests, the Web server looks for the requested file on the memory file system and if present, the file is sent to the Web browser making the request. If the file is not present on the memory file system, an HTTP 404 error page is sent to the browser. Contents of the memory file system for the benchmarking demonstration can be viewed in `Tutorial_Sandbox\SW\Video_Demo\memfs\memfs_files`.

On the Web server, after the video streams are selected and the **start video** button is clicked, the selected video streams start to display on the monitor. Also, the performance monitor thread starts periodically collecting the performance numbers.

The AXI-MM throughput data for read and write and other on-chip parameters are requested by means of the POST request. When AXI-MM throughput data is requested by the Web server, the software application posts the current read and write performance throughput numbers that are probed at the AXI-MM interface. The video streams are continuously performing the frame buffer transactions by means of various VDMA channels. Hence the performance numbers reflect the traffic at the AXI-MM interface. Then, the `video_demo` application sends the throughput number (in Gb/s) and the appropriate HTTP header to the requesting Web browser.

Video Demonstration Hardware Setup

The video demonstration uses a Web server to serve a Web page that graphs the AXI-MM data throughput and on-chip parameters (temperature, V_{CCINT}, and V_{CCAUX}).

The video demonstration setup has two variants:

- With external (live) video
- Without external (live) video

The hardware setup for video demonstration is done as described in UG913, *Getting Started with the Kintex-7 FPGA KC705 Embedded Kit* [Ref 2].

Running the Video Demonstration

After the video demonstration files are imported and built, the application can be run on the KC705 board. Follow these steps to set up the serial terminal console, program the FPGA, and run the video demonstration:

1. Program the FPGA as described in Running the Board Test Software Application (Console Mode), page 22. Ensure that the bitstream from **Tutorial_Sandbox\SW\video_demo\SDK_Workspace\hw_platform** is used.
2. Right-click **video_demo** and select **Run As > Run Configurations.**
3. In the Run configurations box, select **Xilinx C/C++ ELF** and click the **New** button icon to create a new Run configuration.
4. Populate the Main tab of the new debug configuration with these selections:
   - Name: **video_demo Release**
   - Project: **video_demo**
   - Build Configuration: **Release**
   - C/C++ Application: **Release/video_demo.elf** to select the executable that was built with the Release configuration
5. Set the **STDIO** settings as done in Running the Hello World Software Project, page 15.
6. Click **Run**. An XMD console appears indicating that the `Video_Demo` application is downloaded into memory and run.
7. The `Video_Demo` application displays information about the Web server on the terminal screen, as described in UG913, *Getting Started with the Kintex-7 FPGA KC705 Embedded Kit* [Ref 2].

Interacting with the Video Demonstration

When the Web server is running, the Web browser on the connected host computer displays the video demonstration Web page when the URL is set to http://192.168.1.10. The initial Web page that is displayed is shown in Figure 34. The Web page uses JavaScript, so the browser must have JavaScript enabled.

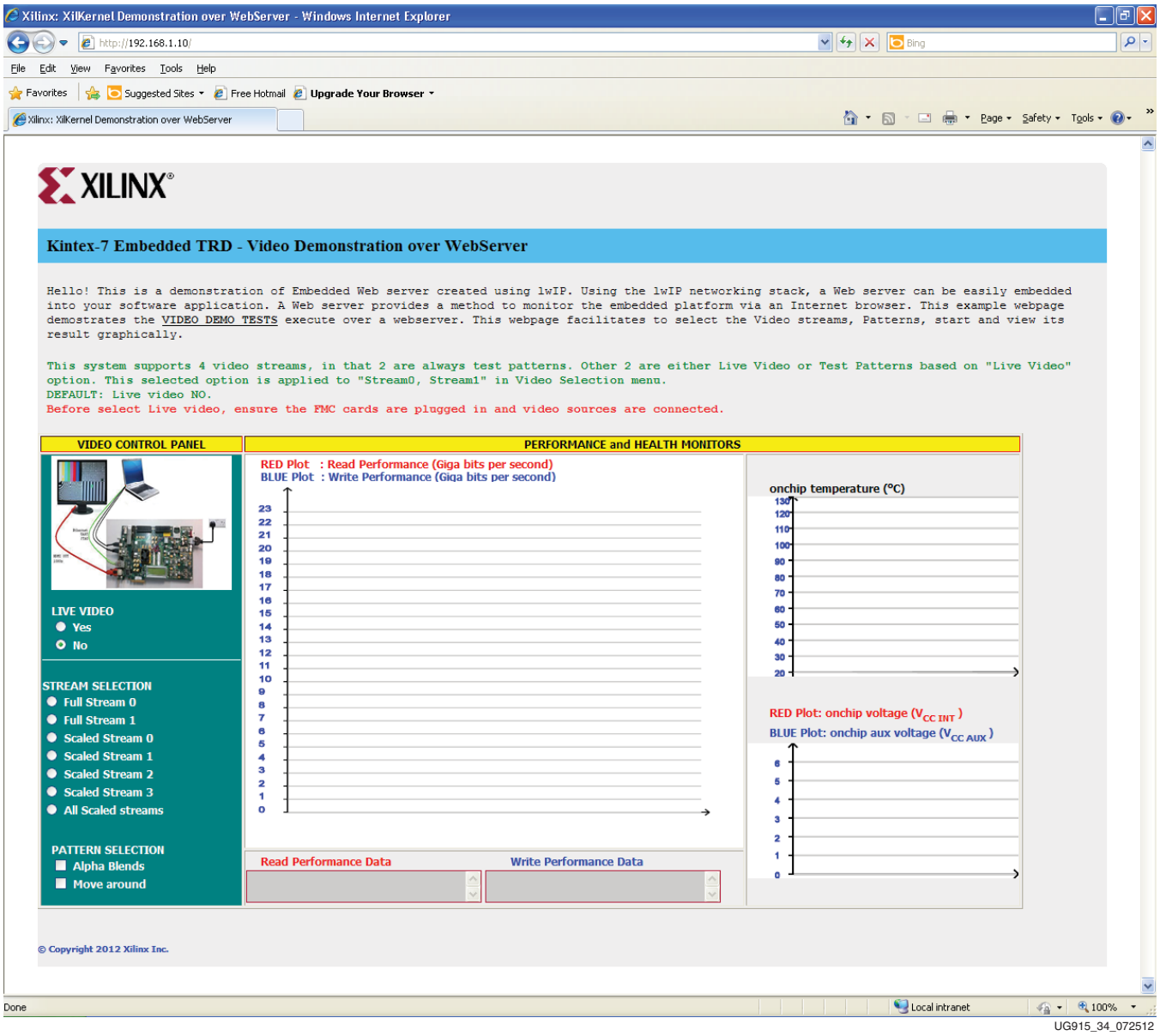


Figure 34: **Initial Video Demonstration Web Page**

The execution of video demonstration and the selection of various options are described in UG913, *Getting Started with the Kintex-7 FPGA KC705 Embedded Kit* in detail [Ref 2].

However, a quick start for the video demonstration is as follows. On a web page, perform the following:

- Click the **NO** radio button under **LIVE VIDEO.**
- Click the **All Scaled streams** radio button under **STREAM SELECTION.**
- Click the **Move around** checkbox under **PATTERN SELECTION.**

These settings start all 4 streams in scaled mode and also apply the *move around* feature. The monitor starts to display the streams and graphs on the web page start being plotted.

The Web browser receives one packet of results at a time. This packet contain 5 results:

- Read throughput (Gb/s)
- Write throughput (Gb/s)
- On-chip temperature (degrees Celsius)
- On-chip V_{CCINT} voltage
- On-chip V_{CCAUX} voltage

JavaScript converts these results into coordinates and updates the graphs. The graphs update every time a new data packet is received. The graphs hold a maximum of 12 data points at a time. After the maximum has been reached, the oldest data point is dropped and the newest data point is added.

This application continues to run infinitely. To terminate the test, close the web page and turn off the board.

Next Steps

The video demonstration software design blocks are shown in Figure 35 and Figure 36.

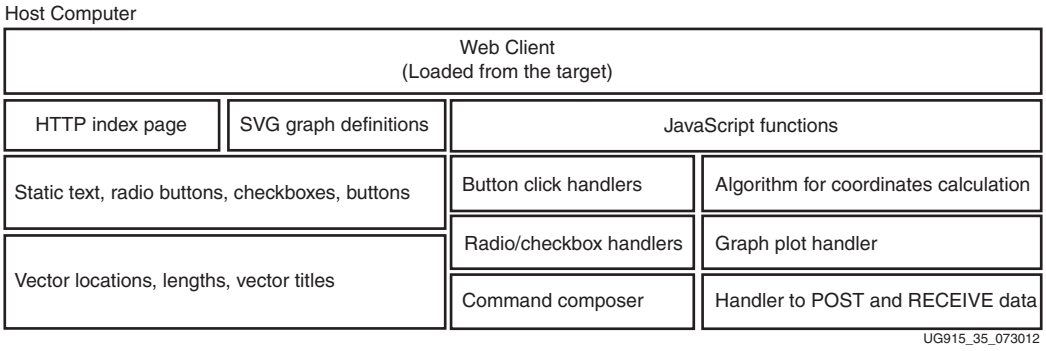


Figure 35: **HTML Design Blocks**

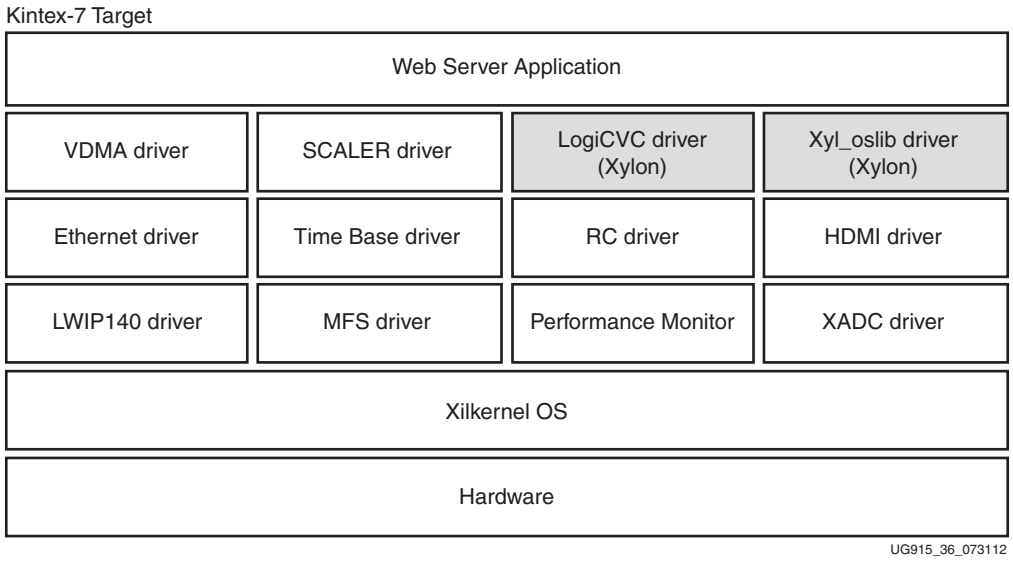


Figure 36: **Kintex-7 Video Demonstration Software Design Blocks**

Because the design is modular and the blocks are plug-in in nature, the next steps would be to replace blocks with different functionality, or to add new components to the existing software design.

An example design is shown in Figure 37.

Replace the Scaler driver block with a JPEG driver that handles different compression algorithms. Similarly, replace the output driver from HDMI to DVI. (Shaded boxes are changes proposed).

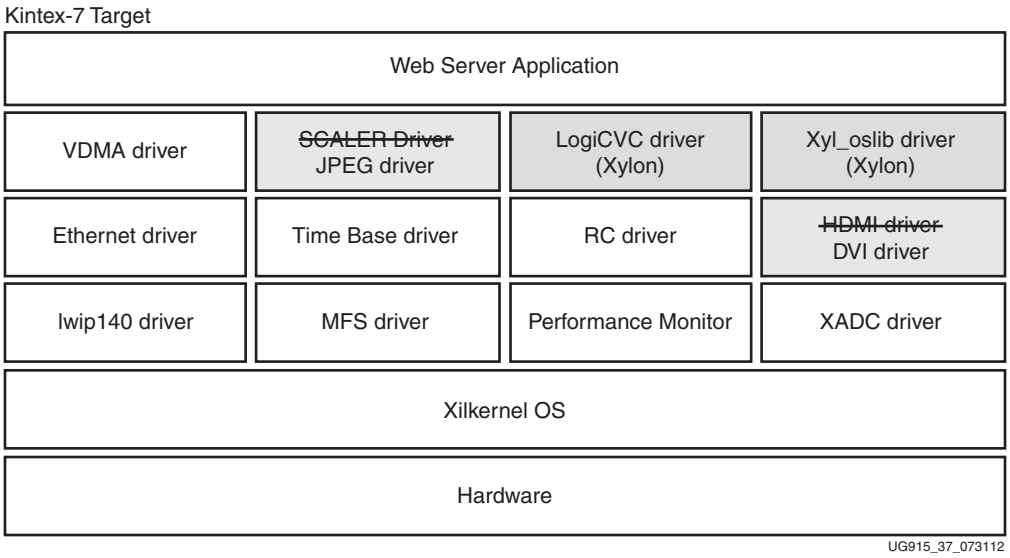


Figure 37: **Proposed Changes to the Kintex-7 Video Demonstration Software Design Blocks**

Similarly, the host-side HTML/JavaScript/SVG scripts can be designed to add new components. An example is shown in Figure 38. In this example, we can introduce the video display on the web page as an additional feature. (Shaded boxes are proposed changes). Table 6 lists additional possible next steps.

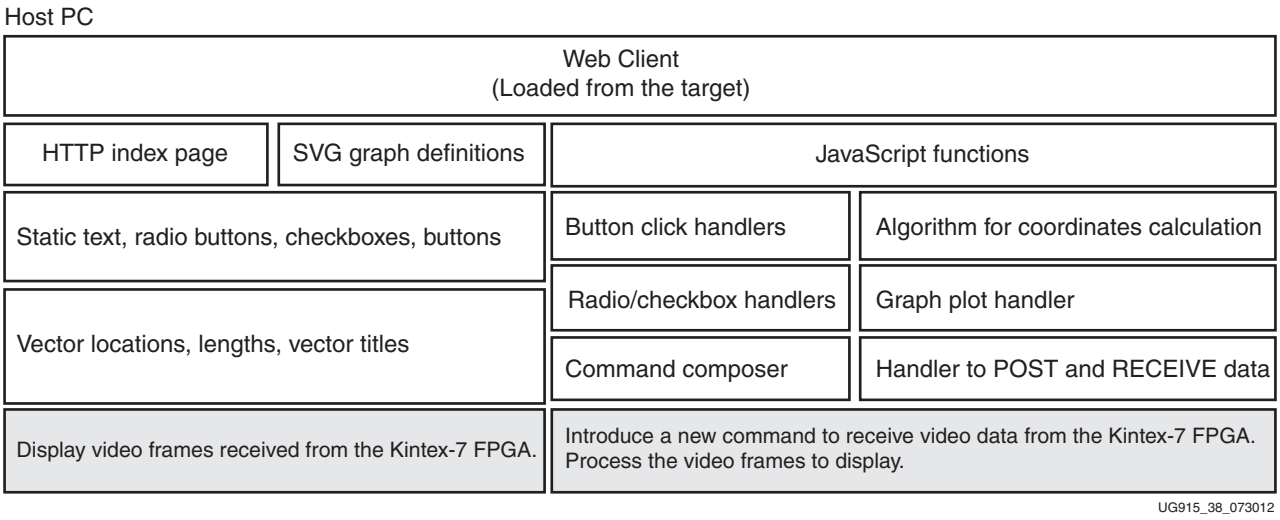


Figure 38: **Proposed Changes to the HTML Design Blocks**

Table 6: **Suggested Next Steps**

Next Step	Refer To
Develop with xilkernel	UG643, *OS and Libraries Document Collection* [Ref 6]
Further configure a stand-alone platform	SDK documentation (**Help > Help Contents** in the SDK)
Advanced profiling	SDK documentation (**Help > Help Contents** in the SDK)

Running the Pre-Built Software Applications

The `ready_for_download` directory under the `KC705_System` directory contains the pre-built hardware bitstream and software executable files for the `board_test_app_Console` stand-alone software application and `board_test_app_Webserver` web-based software application. The pre-built files for the `Video_Demo` software application are stored under the `Video_Demo/ready_for_download` directory. This section details steps to run the pre-built `board_test_app_Console` stand-alone software application, the pre-built `board_test_app_Webserver` web-based software application and the pre-built `Video_Demo` software application. This section is optional and only provides a quick way to run the software applications included with the Embedded Kit. The instructions are in these subsections:

- Running the Pre-Built Board_Test_App_Console Stand-Alone Software Application
- Running the Pre-Built Board_Test_App_Webserver Software Application
- Running the Pre-Built Video_Demo Software Application

Each of those three subsections require the following setups:

- The board setup and UART setup are described in UG913, *Getting Started with the Kintex-7 FPGA KC705 Embedded Kit* [Ref 2].
- Open an ISE® Design Suite command prompt. From Windows, select **Start > All Programs > Xilinx Design Tools > ISE Design Suite > Accessories > ISE Design Suite Command Prompt.**

Running the Pre-Built Board_Test_App_Console Stand-Alone Software Application

To run the pre-built `board_test_app_Console` stand-alone software application, the hardware bitstream must be programmed to the FPGA and the application must be downloaded into memory. To run the `board_test_app_Console` using the files in the ready_for_download directory under `KC705_System`, follow these steps:

1. Open an ISE Design Suite command prompt.

   $ **cd KC705_Embedded_Kit\KC705_System\ready_for_download**

   $ **xmd**

   XMD% **fpga -f download.bit**

   XMD% **connect mb mdm**

   XMD% **dow board_test_app_Console.elf**

   XMD% **con**

2. The resulting output on the serial terminal is as shown in Figure 11. Further details on the included tests in the board_test_app software application can be found in Table 3.
3. When software execution is complete, enter these commands:

   XMD% **stop**

   XMD% **rst**

   XMD% **exit**

Running the Pre-Built Board_Test_App_Webserver Software Application

To run the pre-built board_test_app_Webserver web-based software application, the hardware bitstream must be programmed to the FPGA and the application must be downloaded into memory. To run the board_test_app_Webserver using the files in the ready_for_download directory under KC705_System, follow these steps:

1. Connect an Ethernet cable between the KC705 board and the host computer. Setup the IP address of the host computer Ethernet connection to static address: **192.168.1.100**.
2. Open an ISE Design Suite command prompt as described above.

   $ **cd KC705_Embedded_Kit\KC705_System\ready_for_download**

   $ **xmd**

   XMD% **fpga -f download.bit**

   XMD% **connect mb mdm**

   XMD% **dow board_test_app_Webserver.elf**

   XMD% **con**
3. On the host computer, start Internet browser (Internet Explorer version 8) and browse the target. In the address bar, type **http://192.168.1.10** and **Enter**. This brings up the web page as shown in Figure 18.
4. Figure 17 shows the resulting output on the serial terminal. Further details on the included tests in the board_test_app software application can be found in Table 3.
5. When software execution is complete, enter these commands:

   XMD% **stop**

   XMD% **rst**

   XMD% **exit**

Running the Pre-Built Video_Demo Software Application

To run the pre-built Video_Demo software application, the hardware bitstream must be programmed into the FPGA and the application must be downloaded into memory. To run the Video_Demo using the files in the ready_for_download directory under Video_Demo, follow these steps:

1. Connect the KC705 board to an Ethernet port on a computer via an Ethernet cable.
2. Assign an IP address to the PC of **192.168.1.100**.

   Note: The demonstration uses a hard-coded MAC address and a fixed IP address of 192.168.1.10 and does not connect to the regular LAN network using DHCP. Do not connect more than one board to the same network segment.

3. Open an ISE Design Suite command prompt as described above.

   $ **cd KC705_Embedded_Kit\Video_Demo\ready_for_download**

   $ **xmd**

   XMD% **fpga -f download.bit**

   XMD% **connect mb mdm**

   XMD% **dow video_demo.elf**

   XMD% **con**

   Further details on the using the `Video_Demo` software application when running can be found in Video Demonstration with xilkernel and lwip, page 46.

4. When software execution has completed, enter these commands:

   XMD% **stop**

   XMD% **rst**

   XMD% **exit**

XILINX®

Appendix A

Additional Resources

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see the Xilinx Support website at:

http://www.xilinx.com/support.

For a glossary of technical terms used in Xilinx documentation, see:

http://www.xilinx.com/support/documentation/sw_manuals/glossary.pdf.

References

1. Xylon IP Cores web page: http://www.logicbricks.com/Products/logiCVC-ML.aspx
2. UG913, *Getting Started with the Kintex-7 FPGA KC705 Embedded Kit*
3. UG914, *AXI Interface Based KC705 MicroBlaze Processor Subsystem Hardware Tutorial*
4. UG683, *EDK Concepts, Tools, and Techniques*
5. DS669, *AXI Interface Based KC705 Embedded Kit MicroBlaze Processor Subsystem Data Sheet*
6. UG643, *OS and Libraries Document Collection*
7. UG081, *MicroBlaze Processor Reference Guide*