

Using VxWorks BSP with Zynq-7000 AP SoC

Authors: Uwe Gertheinrich, Simon George, Kester Aernoudt

Summary

VxWorks from Wind River:

- Is a Real Time Operating system (RTOS).
- Is a platform-based approach with configurable components that relate to different architecture support, network, file system, compiler and development tool chains.
- Supports the Zynq[®]-7000 All Programmable (AP) SoC architecture of multicore processor systems.
- Has support for asymmetric multiprocessing (AMP) and symmetric multiprocessing (SMP).

This application note is intended as a getting started guide for new users of VxWorks on the Zynq-7000 device. The document contains the following primary sections:

- Introduction: Explains the important elements of the Zynq-7000 software environment to provide a better understanding of BSP and application generation. This includes the:
 - ROM mechanism
 - Function of the first stage bootloader (FSBL)
 - Wind River bootloader
 - Explanation of the Zynq-7000 processor subsystem boot process
- Building VxWorks for Zynq-7000 AP SoC, page 5: Explains native flash (SD Card) and remote Ethernet (FTP) boot source options and the bootloader configurations for both options.
- Building and Debugging the Application, page 17: Explains how to create, build and remotely run a custom application with VxWorks on a Zynq-7000 device.

This document assumes familiarity with the Xilinx[®] ISE[®] Design Suite and Zynq-7000 AP SoC design methodology. This document includes a reference system for the Xilinx ZC702 board derived from the *Zynq-7000 AP SOC - Concepts, Tools and Techniques User Guide* [Ref 1], which provides the details on how to rebuild such a system.

- Accessing a Peripheral in the Processing System, page 20: Describes how to access peripheral systems in the Zynq-7000 Processing System (PS).
- Conclusion, page 25: Summarizes what this application note describes.
- Additional Resources, page 25: Provides additional resource links.

© Copyright 2013 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.

Hardware and Software Requirements

Software Requirements

- Xilinx[®] ISE[®] Design Suite: Embedded or System Edition 14.6 or Vivado[®] Design Suite, 2013.2.
- Wind River Workbench for VxWorks 6.9.3, which includes a Zynq-7000 AP SoC BSP
- Serial Communication utility program (such as Tera Term)

Hardware Requirements

- Xilinx ZC702 Development Board
- Ethernet Cable
- USB UART Cable

Introduction The Zynq-7000 AP SOC devices takes advantage of the on-chip CPU to facilitate configuration. Initial device configuration of the processing system (PS) and the programmable logic (PL) must take place through the PS when not using JTAG.

Two major blocks control the configuration:

- The first is the BootROM which is a static block of memory that is executed by the multiprocessor core after power-on reset and warm reset.
- The second major block is the device configuration unit which controls JTAG debug access and provides an interface to the AES, HMAC, and PCAP blocks for PL configuration and data decryption.

Both the PS and PL can be configured under PS control either securely or non-securely. Configuration under external host control is also possible using JTAG.

Unlike other Xilinx 7 series devices, Zynq-7000 AP SOC devices do not support initial PL controlled configuration. Configuration on the Zynq-7000 AP SOC devices is a multi-step process. The configuration process involves a minimum of two stages, but generally requires three stages.

The stages are:

- Stage 0: BootROM, page 3: Referred to as the BootROM, this stage controls initial device startup. The BootROM is non-modifiable code executed by the processor after power-on reset and warm reset.
- Stage 1: First Stage Bootloader, page 4: This is generally a first stage boot loader (FSBL), but it can be any user-controlled code. See the *Zynq-7000 AP SOC Software Developers Guide (UG821)* [Ref 1] for details about FSBL.
- Stage 2: VxWorks Bootloader, page 4: This is generally user-configurable software that can act as a second stage boot loader (SSBL). This stage is completely within user control. In the case of this document, it is part of the VxWorks bootloader.

Figure 1, page 3 illustrates a non-secure boot process for typical Linux system. Uboot is an example for higher-level boot loader and can be exchanged by VxWorks bootloader.



Figure 1: Boot Flow

Stage 0: BootROM

The Zynq-7000 AP SoC processor subsystem configuration starts after power-on reset. The ARM[®] CPU starts executing code from the on-chip BootROM with JTAG disabled. The BootROM contains code for base drivers for NAND, NOR, Quad-SPI, SD, and PCAP. DDR and other peripheral initializations are not performed from the BootROM and must be done in the Stage 1 image, First Stage Bootload (FSBL) or later.

For security, the CPU is always the first device out of reset among all master modules within the PS. When the BootROM is running the JTAG is disabled to ensure security.

The BootROM code is also responsible for loading the FSBL. Zynq-7000 AP SoC architecture supports multi-stage user boot image loading; any further user boot image loading after FSBL is the responsibility of the user. When the BootROM releases control to FSBL, user software assumes full control of entire system. The only way to execute the BootROM again is by performing a reset.

The PS boot source is selected using the mode-pin signals (indicated by a weak pull-up or pull-down applied to specific pins), which are sampled after during power-on reset. The sampled value is stored in the BOOT_MODE register.

The BootROM supports encrypted and unencrypted images referred to as secure boot and non-secure boot, respectively. Additionally, the BootROM supports beginning execution of the stage 1 image from OCM after copying the image or executing direct from linear flash (NOR or QSPI) when using the execute-in-place (XIP) feature.

- In secure boot the CPU, running from secure BootROM code, decrypts and authenticates the incoming user PS image, stores it in the OCM RAM, and then branches into that RAM.
- In non-secure boot the CPU, running from BootROM code, disables all secure boot features including the AES engine within the PL before branching to the user image in the OCM RAM or flash, if XIP is used. The Processor System (PS) boot image is limited to 192 KB unless booting with XIP.

Any subsequent boot stages for either the PS or the PL are the responsibility of the user and are under user control. The BootROM code is not accessible to the user.

• Following the stage 1 secure boot, you can proceed with either secure or non-secure subsequent boot stages.

 Following a non-secure first stage boot, only non-secure subsequent stage boots are possible.

For secure boot decryption and authentication, the PS uses the hard-wired AES-256 and SHA-256 modules within the PL. For this reason, the PL must be powered up during any secure boot, even if only the PS is configured. The device encryption key is user-selectable from either the on-chip eFUSE unit or the on-chip block RAM.

The possible boot sources are: NAND, NOR, SD, Quad-SPI, and JTAG. The first four boot sources are used in master boot methods in which the CPU loads the external boot image from nonvolatile memory into the PS.

Stage 1: First Stage Bootloader

The First Stage Bootloader (FSBL) starts after the execution of the BootROM. BootRom loads the FSBL into the OCM, or the FSBL executes in place (XIP) unencrypted from memory mapped flash (NOR or Quad-SPI), contingent upon the BootROM header description.

The FSBL is responsible for:

- Initialization using the PS configuration data provided by Xilinx Platform Studio (XPS) (see "Zynq-7000 PS Configuration" in the Zynq-7000 AP SOC Software Developers Guide (UG821) [Ref 2].
- Programming the PL using a bitstream
- Loading second stage bootloader or bare-metal application code into DDR memory
- Starting execution of the second stage bootloader or bare-metal application

Note: Before handoff to the second stage bootloader or bare-metal application, the FSBL invalidates the instruction cache and disables the cache and MMU, because Linux (and perhaps other operating systems) assume it is disabled upon start.

See the FSBL code provided with SDK for details on how the FSBL initializes the CPU and peripherals used by the FSBL, and how it uses a simple C run time library.

The bitstream for the PL and the second stage bootloader or bare-metal application data, as well as other code and data used by the second stage bootloader, Linux (or other operating system), or bare-metal application are grouped into partitions in the flash image.

Stage 2: VxWorks Bootloader

The VxWorks bootloader application loads a VxWorks image onto a target. Like VxWorks, you can configure the VxWorks bootloader with various facilities; such as a command line interface for dynamically setting boot parameters, a network loader, and a file system loader.

Uniprocessor (UP), symmetric multiprocessor (SMP), and asymmetric multiprocessor (AMP), configurations of VxWorks use the same bootloader.

In a development environment, a bootloader is useful for loading a VxWorks image from a host system, where you can modify and rebuild VxWorks. You can also use a VxWorks bootloader in production systems when the bootloader and operating system are stored on a disk or other media.

Self-booting (standalone) VxWorks images do not require a bootloader. These images are commonly used in production systems (stored in nonvolatile devices).

Usually, the bootloader is programmed in a nonvolatile device (usually flash memory or EEPROM) at an address such that it is the first code run by the processor when the target is powered on or rebooted. The procedure to get the boot loader programmed in a nonvolatile device or written to a disk is dependent on the target, and is described in following section using an SD card image.

The VxWorks product installation includes default bootloader images for each installed BSP. If they do not meet your needs, you can create a custom bootloader.

Host Environment Configuration

The following steps are one-time only:

- 1. Install VxWorks Tool chain 6.9.3.1.
 - a. Install Base Tools Package.
 - b. Invoke the Product Maintenance GUI.
 - Update the installer.
 - Configure online Content Update Network settings.
- 2. Apply updates based upon your license file, as shown in Figure 2.

Select Products
VxWorks 6.9.3.1 and VxWorks Edition 6.9.3.1 Platforms
▷ 📝 🍄 Board Support Packages for VxWorks 6.9.3.1 for ARM
VxWorks 6.9.3.1 and General Purpose Technologies
⊳ 📝 🍄 Wind River Diab Compiler 5.9.1 for VxWorks 6.9
📝 🍄 Wind River Setup 2.0.11 (shared)
a 📝 🗁 Wind River Workbench 3.3.4 Processor Pack 1
Image: Second Action of the
📝 💱 Wind River Setup 2.0.11 (shared)
Image: Wind River Workbench 3.3.4 for VxWorks 5.5 (Enabler Rev 6)
⊳ 📝 💱 Wind River Workbench 3.3.4 for VxWorks 6.3 - 6.9 (Enabler Rev 6)
⊳ 📝 🚯 Wind River Workbench 3.3.4 for Wind River Linux 4.0 - 5.0 (Enabler Rev 6)
V Wind River Workbench Core 3.3.4

Figure 2: Select Products Dialog Box

Building VxWorks for Zynq-7000 AP SoC



The Zynq-7000 BSP is a standard part of the 6.9.3.1 install, as shown in Figure 3:

- Because of the asynchronous nature of VxWorks BSPs, verify that you install the latest Xilinx BSP. The link is provided at [Ref 4].
- Install the BSP patches as described in the BSP download link: VxWorks 6.9.3.1 BSP Driver Source Patch for BSP, The link is provided at [Ref 5].
- 5. Complete all details of the build steps to apply the patch into the source tree.

Configure and Build a VxWorks BootROM and Kernel Image

The default BSP does not enable support for accessing an SD card. Because you use the SD card to store the VxWorks image, the first step is to modify the BSP configuration.

 In a text editor, open the <Install_Dir>/vxworks-6.9/target/config xlnx -zynq7k/config.h file, and modify line 197 from:

```
#undef DRV_STORAGE_SDHC/
to
#define DRV_STORAGE_SDHC
#define INCLUDE_DOSFS
#define INCLUDE_DOSFS_MAIN
#define INCLUDE_DOSFS_CHKDSK
#define INCLUDE_DOSFS_FAT
#define INCLUDE_DOSFS_SHOW
#define INCLUDE_DOSFS_DIR_VFAT
#define INCLUDE_DOSFS_DIR_FIXED
#define INCLUDE_FS_MONITOR
```

#define INCLUDE_FS_EVENT_UTIL
#define INCLUDE_ERF
#define INCLUDE_XBD
#define INCLUDE_XBD_BLKDEV
#define INCLUDE_DEVICE_MANAGER
#define INCLUDE_XBD_BLK_DEV
#define INCLUDE_XBD_PART_LIB
#define INCLUDE_DISK_UTIL

This enables the SDHC controller, as well as drivers for the FAT file system.

To use the VxWorks BSP with the Wind River Workbench to create a VxWorks Kernel Image, do the following:

2. Start the Wind River Workbench tool and select a workspace.

The Wind River SDK opens.

3. In the main context menu select File > New > Project.

The New Project Wizard opens.

4. Under VxWorks 6.x, select the VxWorks Image Project, as shown in Figure 4.

S New Project	
Select a wizard Creates a new VxWorks image project with all available kernel build specs	
Wizards:	
type filter text	
Wind River Workbench Project General CVS Desktop JavaScript VxWorks 6.x YxWorks Boot Loader / BSP Project YxWorks Downloadable Kernel Module Project YxWorks Image Project YxWorks Real Time Process Project YxWorks ROMFS File System Project Show All Wizards.	E F
Sack Next > Finish	Cancel

Figure 4: VxWorks Image Project

The New VxWorks Image Project multipage wizard opens.

- 5. Enter a project name, for example, zynq_vxworks_01, and click Next.
- 6. Select the **xlnx_zynq7k** BSP used for this project as highlighted in Figure 5.

Project Create a new WWorks image project with all available kernel build spec		
Project name:	New VxWorks Image Project	
Create project in workspace Create project at external location	Project Setup Base the new project either on an existing project, or on a board supp package and a tool chain.	ort
Overstary DESIGNERZYNQUZC_282:Designer/WWorkstworkspace -	Setup the project	
	Based on a board support package	
	Project) fin	NOR AL
	BSP: stns_zynq7k + Bro	wse
	Address model 32-bit kernel •	
	Tool chain: diab •	
	Target Agent	
	BSP validation test suite	
	Add support to project Options	
Kack Not> Finish	Setup information Base directory: C:/CAD/WindRiver/WR_692/vsworks-6.9/target/con	fig/xina_zynq7k
	(?) «Back Nee» Finish	Guncel

Figure 5: VxWorks Image Project Multipage Wizard

7. From the New VxWorks Image Project wizard, select **PROFILE_DEVELOPMENT** (Figure 6).

🔞 New V	Works Image Project	
Configur	ation Profile	
Select ke	rnel configuration profile.	
Profile:	(no profile)	
Profile	(no profile) PROFILE_BOOTAPP	
Descrip	PROFILE_COMPATIBLE	*
	PROFILE SMALL FOOTPRINT	
	PROFILE_STANDALONE_DEVELOPMENT	
		_
Synopsi	S: The kernel will be configured with its predefined defaults.	<u>^</u>
		~
?	< Back Next > Finish	Cancel

Figure 6: New VxWorks Image Project: PROFILE_DEVELOPMENT Option

- 8. Click Finish.
- 9. Open the Kernel configuration. Change the configuration to include the symbol table in the Kernel image (Figure 7, page 10).

Description	Name	Type V
a 🏭 development tool components (default)	FOLDER_TOOLS	
Cafe analysis system	FOLDER_CAFE	
E I Core Dump components	FOLDER_CORE_DUMP	
Downloadable kernel modules compiler	suf SELECT_COMPILER_INT	
Æ Kernel-write components	FOLDER_KERNEL_DEBUG	
Bystem Viewer components	FOLDER_WINDVIEW	
USB Debug	FOLDER_USB_DEBUG	
WDB Agent Proxy components	FOLDER_WDB_PROXY	
WDB agent components (default)	FOLDER_WDB	
boot application components	FOLDER_BOOT_APP	
kernel shell components	FOLDER_SHELL	
Ioader components	FOLDER_LOADER	
show routines	FOLDER_SHOW_ROUTI.	
symbol table components	FOLDER_SYMTBL	
symbol table initialization component	s FOLDER_SYM_TBL_INIT	
🕢 🕁 select symbol table initialization (lef SELECT_SYM_TBL_INIT	
🔓 built-in symbol table (default)	INCLUDE_STANDALONE	
🕞 downloaded symbol table	INCLUDE_NET_SYM_TBL	
System symbol table initialization	(de INCLUDE_SYM_TBL_INIT	
C++ symbol demangler	INCLUDE_CPLUS_DEMA.	
👂 💕 error status table	INCLUDE_STAT_SYM_TB	L

Figure 7: Component Configuration

You can now build the Kernel image.

Building the Kernel Image

1. In the Project Explorer window, mouse over to the Image project, right-click and select **Build Project**. The VxWorks image file is located in the ,.\<project_name>\default directory. Figure 8 shows the Build Project option.

Project Explorer 83		Getting Started 💮 creation.log 🕮
	∃ 🍇 £) = 🔛 🐺 = 🏹	Output of C:\CAD\WindRiver\WR_
Kernel Configuration VwWorks.bin (default) vwWorks.hex (default) vwWorks.hex (default)	New Edit Kernel Configuration Go Into	
Binaries	Open in New Window	
> 🔊 Includes	Сору	Ctrl+C
⊳ 🌌 default	D Paste	Ctrl+V
Egg xinc_zynq/k El linkSymt.c	🔀 Delete	Delete
R prjComps.h	Attributes	,
prjConfig.c	Source	•
B prjParams.h	Move	
is us Apploit.c	Rename	F2
e usrRtpAppInit.c	import	
CompObjs.lst	Export	
Makefile	Open Wind River VsWorks 6	9 Development Shell
Do Makefile.mk	2 Refresh	13
De prjObjs.lst	Close Project	
www.orks.makefile	Close Unrelated Projects	
Remote Systems 13	Project References	
Je 1 21	Build Project	Ctrl+B, Ctrl+P
E Local	Kebula Project	

Figure 8: Build Project Option

2. When the project build is complete, start a Wind River VxWorks development shell.

Use this shell to build a bootROM binary. The bootROM binary is the VxWorks bootloader (similar to Uboot); it is not within the Zynq-7000 device ROM.

3. Within the shell, type:

```
cd ..\<install_dir>\Wind
River\vxworks-6.9\target\config\xlnx_zynq7k\
make clean make bootROM
```

The commands generate a file with the name bootROM. Rename the file to bootROM.elf

4. Create a boot.bif and a zynq_fsbl_0.elf file, with the following format:

```
ZC702_bif_for_VxWorks:
    {
    [bootloader]zynq_fsbl_0.elf
    bootROM.elf
    }
```

5. Copy the bootROM.elf, zynq_fsbl_0.elf, and the boot.bif file to the /bootgen directory.

Alternatively, you can copy the bootgen.exe tool to the current installation directory.

```
C:\<install_dir>\Wind
River\vxworks-6.9\target\config\xlnx_zynq7k\
```

6. In the Wind River shell, type:

bootgen image boot.bif I BOOT.BIN

- 7. Copy the following files onto an SD card:
 - VxWorks from the .. <project_name>\default directory
 - BOOT.BIN from the ...<install_dir>\WindRiver\vxworks 6.9\xlnx_zynq7k\ directory.

This creates a system that can boot from an SD card.

The following subsection describes the required steps to boot from an SD card.

Booting From an Secure Digital Card

Use the Secure Digital (SD) card to boot the Zynq-7000 AP SoC Processor System (PS).

- 1. Connect a power cable, a Xilinx USB download cable, an Ethernet cable, and a USB UART cable to the board.
- 2. Put the SD Card in the SD card pole of the board. Ensure that the switches for booting from SD Card are in the right position. Ensure that the settings of Jumpers J27 and J28 are the same as shown. Move the DIP-Switches 3 and 4 of SW 16 to the left (this sets the switches to 1) (Figure 9, page 13).



Figure 9: Image of SD Card and USB Connection

These settings ensure an SD card boot.

- 3. Open a terminal session, and choose the right COM $\tt port$ () function, and set the Baud Rate to 115200.
- 4. Switch on the board.
- Stop the Autoboot process by pressing the keyboard Return key. The VxWorks bootROM prompt opens.
- 6. Type C at the boot prompt, and press **Return** to start the boot configuration.
- 7. Change the boot device to fs and press **Return** until you reach the file name.
- 8. Change to /sd0:1/vxWorks and press Return until you reach other (o).
- 9. If no entry exists, type gem0. Press Return and the boot prompt opens.
- 10. Type @ to proceed the boot process.
- 11. Type i to display all running tasks.

VxWorks boots and presents the output as shown in Figure 10.

1)1)1)1)1)1)1)1)1)1)1)1)1)1)1)1)1)1)1)	11111111111	111111111111111111111111111111111111111	111111111	1111]]]]					
1000000000000000000000000000000000000	1111111111	111111111111111111111111111111111111111		1111	111					
1)1010101010101010101010101010101010101		,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,		IIII	11					
1 11111111 1111111 111111 11 1111111 111111 111111 111111 111 11111 111111 111111 111111 111111 1111 11111 111111 111111 111111 111111 111111 11111 11111 111111 111111 111111 111111 111111 111111 111111 111111 111111 111111 1111111 111111 111111 111111 111111 1111111 111111111111111111111111111111111111	11111	111111 1111	11111	1111]]]]]]]		(R)	
11 1111111 1111111 11111111 111 111111111111111111111111111111111111	1 1111	11111 111111	11111	1111]]]]]]			
111 111 111 111 111 111 1111	11 111	1111 11111111	1111]]]	1]]]]]]			
111 111 111 11111 111111 11111111 1111111 1111111 1111111 11111111111111111111111 111111111111111111111111111111111111	111 11]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]	11 1	11 11111111	11 1111]]]]]]	11 11	111	
11111 1 11111 111111 111111 111111 111111 111111 111111111111111111111111111111111111	1111 1]]]]]]]]]]]]]]]]]]]]]]	11 11111	11 1111 1]]]]]	
111111 111111 111111 111111 11111111 1111111 111111111111111111111111111111111111	11111]]]]]]]	111	111	11111 1111	11 1111	111111]]]]]	
1))))) 1)))) 1))))) 1))) 1))))) 1))))) 1)))))) 1))))))))))))))))))))))))))))))))))))	111111	11111 11	1111 1]	1111 1111]]]]]]	111111]]]	111	
1))))))) 1)))))) 1)))))) 1))))))))))))))))))))))))))))))))))))	1111111]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]	11111 1		111 1111	11 1111	1111 1]]]	1111	
1))))))))))))))))))))))))))))))))))))	11111111	11111 111	1111111		1 11111	11 1111]]]]	1111 11]]]	
1))))))))))))))))))))))))))))))))))))	11111111111	111111111111111111111111111111111111111]]]]]]							
1))))))))))))))))))))))))))))))))))))	1111111111	111111111111111111111111111111111111111]]]]])evelopπent S	System				
1)11111111111111111111111111111111111	1111111111	111111111111111111111111111111111111111]]]]							
1))))))))))))))))))))))))))))))))))))	11111111111	111111111111111111111111111111111111111]]]	٧x	lorks 6.9					
111111111111111111111111111111111111	11111111111	111111111111111111111111111111111111111]]	KERI	WEL: WIND ver	rsion 2.1	.3			
CPU: Xilinx Zunq-7000 ARHv7. Processor #0. Henory Size: Dxfb00000 (251Hb). BSP version 6.9/0. Created: Sep 26 2012, 22:28:47 ED& Policy Hode: Deployed HDB Connt Type: HDB_COHH_END HDB: Ready. → i NAHE ENTRY TID PRI STATUS PC SP ERRNO DELAY tJobTask 281464 1393bb0 0 PEND 2fe9cc 1393b24 0 0 tExcTask 280bc8 39b260 0 PEND 2fe9cc 39b19c 0 0 tLogTask logTask 13961b8 0 PEND 2fe9cc 39b19c 0 0 tLogTask logTask 13961b8 0 PEND 2fe9cc 1493698 0 0 tBhell0 shellTask 149e268 1 READY 306560 1494fd0 0 0 tHdbTask 218588 1489980 3 PEND 2fe9cc 14898ec 0 0 tErTTask 23641c 1398438 10 PEND 2ff290 13983c0 0 0 tErTTask 23641c 1398438 10 PEND 2ff290 13983c0 0 0 tErTTask 245798 14855a8 25 PEND 2fe9cc 1485538 0 0 tWatbgTask 245798 14855a8 25 PEND 2fe9cc 13936f68 0 0 tMetD ipconWatTask 139cdb0 50 PEND 2fe9cc 13936f68 0 0 tHetO ipconWatTask 139cdb0 50 PEND 2fe9cc 13936f68 0 0 tHetO ipconWatTask 139cdb0 50 PEND 2fe9cc 1485538 0 0 tHetO ipconWatTask 139cdb0 50 PEND 2fe9cc 1475b10 0 0	1111111111	111111111111111111111111111111111111111] ()opy:	right Hind R.	iver Syst	ens, Inc.	, 1984-	2012	
-> i NAME ENTRY TID PRI STATUS PC SP ERRNO DELAY tJobTask 281464 1393bb0 0 PENO 2fe9cc 1393b24 0 0 tExcTask 280bc8 39b260 0 PENO 2fe9cc 39b19c 0 0 tLogTask 10gTask 13961b8 0 PENO 2fd078 1396098 0 0 tShell0 shellTask 149e268 1 READY 306560 149dfd0 0 0 tHdbTask 276588 1489980 3 PENO 2fe9cc 14898ec 0 0 tHdbTask 23641c 1399438 10 PENO 2fe9cc 14898ec 0 0 ipcon_tick> 314904 1444e78 20 PENO 2fe9cc 14494e0 0 0 tWadbgTask 245798 14855a8 25 PENO 2fe9cc 1393628 0 tNet0 ipconWetTask 139cdb0 50 PENO 2fe9cc 139cd28 0 tNet0 ipconMetTask 139cdb0 50 PENO 2fe9cc 139cd28 0 tNet0 ipconMetTask 139cdb0 50 PENO 2fe9cc 139cd28 0 tNet0 ipconMetTask 139cdb0 50 PENO 2fe9cc 1475b10 0 tNetConf 27753c 1475cb0 50 PENO 2fe9cc 1475b10 0 tNetConf 27753c	СРО: XIIIn Meнory Size Created: So ED&R Policy HDB Conn Ty HDB: Readu	CPU: Xilinx Zynq-7000 ARMv7. Processor #0. Memory Size: Dxtb00000 (251Mb). BSP version 6.9/0. Created: Sep 26 2012, 22:28:47 ED8R Policy Mode: Deployed HDB Comm Type: HDB_COMM_END								
NHRE ENTRY TID PRI STATUS PC SP ERRNO DELRY tJobTask 281464 1393bb0 0 PEND 2fe9cc 1393b24 0 0 tExcTask 280bc8 39b260 0 PEND 2fe9cc 39b19 0 0 tLogTask 10gTask 13961b8 0 PEND 2fd078 1396098 0 0 tBhell0 shellTask 149268 1 RENOV 306500 149dfd0 0 0 tHdbTask 248641 1398438 10 PEND 2fe9cc 14898ec 0 0 tErrTask 23641c 1398438 10 PEND 2fe9cc 14844e0 0 0 tVxdbgTask 245798 14855a8 25 PEND 2fe9cc 1485538 0 0 tWetb ipconMetTask 139cdb0 0 PEND 2fe9cc 1485538 0 0 tWetb	-> i									
tJobTask 281464 1393bb0 0 PEND 2fe9cc 1393b24 0 0 tExcTask 280bc8 39b260 0 PEND 2fe9cc 1393b24 0 0 tExcTask 280bc8 39b260 0 PEND 2fe9cc 1393b24 0 0 tLogTask logTask 13961b8 0 PEND 2fe9cc 1396098 0 0 tShe110 she11Task 1492c68 1 REHOY 306560 1494fd0 0 0 tHdbTask 276588 1489980 3 PEND 2fe9cc 149363c0 0 tExrTask 23641c 1398438 10 PEND 2fe9cc 1494640 0 0 tpcon_tick> 314904 1444e78 20 PEND 2fe9cc 1498538 0 0 twdtgTask 245798 1495548 25 PEND 2fe9cc 1494568 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	NAHE	ENTRY	TID	PRI	status	PC	SP	ERRNO	DELAY	
1300135K 201004 1395020 0 PEND 21e9cc 1395024 0 0 tExcTask 1296026 3950260 0 PEND 21e9cc 1395024 0 0 tStopTask 1396188 0 PEND 21e9cc 1395026 0 0 tStopTask 1396188 0 PEND 21e9cc 1395026 0 0 tStopTask 1492618 1 RENDY 306550 1494fd0 0 0 tHdbTask 2169027 1489882 0 0 0 0 tHdbTask 2169024 1398438 10 PEND 21e9cc 1494fd0 0 0 tpcrn_tick> 314904 1444e78 20 PEND 21e9cc 1494538 0 0 twdbgTask 235798 1495538 25 PEND 21e9cc 1495538 0 0 tNetD ipconMetTask 139610 50 PEND 21e9cc 13816c8 0 0 tpconf 27532 1475cb0 50 PEND 21e9cc 1475b10 0 0	+ Tob Tools	901 <i>464</i>	4000566	 0	DEND	24.0000	1000604	 ח	n	
Lick Task LogTask 13961b8 0 FEND 21620 396109 0 tShell0 shellTask 149268 1 REMDY 306560 14947d0 0 0 tHdbTask 276588 1 REMDY 306560 14947d0 0 0 tHdbTask 276588 1 REMDY 306560 14947d0 0 0 tHdbTask 276588 1 REMDY 306560 14947d0 0 0 tErtTask 23641c 1398438 10 PEND 21629cc 14898ac 0 0 tpcontitc>> 314904 1444a78 20 PEND 2169cc 14445d0 0 0 tWadbgTask 245798 14855a8 25 PEND 2169cc 1485538 0 0 tNetD ipconMetTask 1396810 50 PEND 21629cc 1395628 0 0 tipcon 214974 1381810 50 PEND 21629cc 1395628 0 0 tipcon 27532 1475cb0	+EveTask	201404	206260	о П	PEND	2fe9cc	2061024	0 0	0	
Hubbard 14950100 1600100 190010 1900100 0 Hubbard ShellTask 1490268 1 REHDY 306560 19047400 0 0 Hubbard 2169010 3 PEND 2169010 1908360 0 0 tHubTask 2168588 1489980 3 PEND 216901 3198360 0 0 tEvrTask 236410 1398438 10 PEND 216902 1398360 0 0 tpocnt_itck> 314904 144478 20 PEND 216902 1484538 0 0 tVxdbgTask 245798 1485538 25 PEND 216902 1484538 0 0 thetD ipconMetTask 1390405 50 PEND 216902 1392628 0 0 ipcons_ysi2 248074 1381810 50 PEND 2162902 1392668 0 0 thetConf277532 14752b0 50 PEND 216902 1475b10 0 0	ti ogTask	logTask	1206169	0 0	PEND	216900	1206002	0 N	0	
Contract Strept Contract Strept Contract Strept Contract Strept Contract Strept Strep Strept Strept <t< td=""><td>+SP0110</td><td>shollTask</td><td>140~269</td><td>1</td><td>PEANV</td><td>206560</td><td>1404F40</td><td>0 0</td><td>0</td><td></td></t<>	+SP0110	shollTask	140~269	1	PEANV	206560	1404F40	0 0	0	
Hub Tosk Libbool Libbool Libbool D Lipcon_tick> 314904 14944e78 10 PEND 2fe20c 149363cD 0 0 Lipcon_tick> 314904 14944e78 20 PEND 2fe9cc 1494538 0 0 tWxdbgTask 245798 1495548 25 PEND 2fe9cc 1485538 0 0 tNetdbgTask 248074 139610 50 PEND 2fe9cc 1397628 0 0 ipcon_sys1> 248074 138610 50 PEND 2fe9cc 1475b10 0 0 tNetConf 27753c 1475cb0 50 PEND 2fe9cc 1475b10 0 0	tlldbTask	249599	1490090	2	PEND	2foQcc	14908oc	0 N	0	
Christian Construct Construct <thconstruct< th=""> <thconstruct< th=""> <thc< td=""><td>tFofTack</td><td>23641</td><td>1209200</td><td>10</td><td>PEND</td><td>244200</td><td>13093-0</td><td>n</td><td>0 N</td><td></td></thc<></thconstruct<></thconstruct<>	tFofTack	23641	1209200	10	PEND	244200	13093-0	n	0 N	
Incon_class 245798 1418578 25 PEND 21e9cc 14185538 0 tVxdbgTask 245798 1485538 2 0	incon tick	314004	1444078	20	PEND	2fo9cc	1444da0	0	0	
Hubble Constraint 100000 50 FERD FERD FERD 0 ipcon_sys1> 248d74 138f810 50 PEND 2ff290 138f6c8 0 0 thetConf 27753c 1475cb0 50 PEND 2ff290 138f6c8 0 0	All all Track	01 100 1	44000.0	20	PEND	2fo0cc	140000	0	0	
thetConf27753c1475cb0_50_PEND21f290_13816c800 tNetConf27753c1475cb0_50_PEND21e9cc_1475b1000	THYDDOLASK	245798	19855.38				19000000			
tNetConf 27753c 1475cb0 50 PEND 2fe9cc 1475b10 0 0	tvxdDgTask tNotfl	245798 inconNetTask	1485588 139cdb0	50	PEND	2fe9cc	1400000 139ad28	.0	n.	
	t vxdbg i ask tNet0 incon sus15	245798 ipconNetTask 248d74	1465548 139cdb0 138f810	50 50	PEND	2fe9cc 2ff29cc	139cd28	0	Ŭ	
sdfmcXbdSv> 235198 14470e8 50 PENO 2fe9cc 1447084 0 0	tVxdDgTask tNet0 ipcon_sys1> tNetConf	245798 ірсонNet Task 248d74 27753с	1485548 139cdbD 138f81D 1475cbD	50 50 50	PEND PEND PEND	2fe9cc 2ff290 2ff290	139cd28 138f6c8 1475b10	0 0 0	0	

Figure 10: SD Boot Terminal Transcript

Booting Using FTP

Repeat the first four steps shown in Booting From an Secure Digital Card, page 12, as follows:

- 1. Connect a power cable, a Xilinx USB download cable, an Ethernet cable, and a USB UART cable to the board.
- 2. Put the SD Card in the SD card pole of the board. Ensure that the switches for booting from SD Card are in the right position. Open a terminal session, and choose the right COM port () function, and set the **Baud Rate** to **115200**.
- 3. Change the local area network (LAN) connection of your system network settings to the IP address **192.168.1.1**.

🔋 Local Area C	Connection Properties			
Networking S	haring			
Connect using	р:			
🔮 Intel	Internet Protocol Version 4 (TCP/IPv	(4) Properties	9	23
This conne	General			
	You can get IP settings assigned au this capability. Otherwise, you need for the appropriate IP settings.	itomatically if your i d to ask your netwo	network supp ork administra	orts tor
	Obtain an IP address automat	ically		
i interest	Ouse the following IP address:			
🗹 🔟 Int	IP address:	192.168.	1.1	
•	Subnet mask:	255.255.2	255.0	
Insta	Default gateway:		•	
Transmis	Obtain DNS server address au	itomatically		
across d	Use the following DNS server	addresses:		
	Preferred DNS server:			
	Alternate DNS server:		•	
_	💹 Validate settings upon exit		Advance	d
		ОК	C	ancel

Figure 11: Host Ethernet MAC Configuration

- Start an FTP server. You can use the server delivered with the Wind River tool chain. If you use the Wind River FTP server, select Security > User/rights.
- 5. Click New User and type a name; for example, Zynq and a password.
- 6. Type the home directory of the VxWorks image, then click Done.
- 7. Switch on the board.
- Stop the Autoboot process by pressing Return. The VxWorks bootROM prompt opens.
- 9. At the boot prompt, type **c** then press **Return** to start the boot configuration.
- 10. Type gem0 to change boot device then press Return until you reach File Name.
- 11. Type VxWorks, then press Return until you reach inet on ethernet (e).
- 12. Change the **inet on ethernet (e)** to IP address **192.168.1.2:0xffffff00**, then press **Return** until you reach **host inet (h)**.
- 13. Change host inet (h) IP address 192.169.1.1, then press Return until you reach user.
- 14. Type the user name you choose for the server then press **Return** until you reach **password (pw)**.
- 15. Type password you choose for the server, then press **Return** until you reach **other (o)**.
- 21. If no entry exists, type gem0, then press Return.

The boot prompt opens.

22. Type @ to start the boot process.

VxWorks boots from the image using the terminal (Figure 12, page 16).

File	Edit	Setup	Control	Window	Help
other ())		: денО		A
[VxHork:	s Boot]: 0			
boot de unit nur processo host nam file nam inet on host inm gateway user (u ftp pass flags (f target) other (n	vice aber or num ne ether backp et (h) inet) suord f) name (o)	ber net (e) lane (b) (g) (рц) tn)	: fs : D : host : /d0:1/vxk : 192.168.1. : 192.168.1. : 192.168.1. : 192.168.1. : 90.0.0.1 : target : vxTarget : 0x0 : x1nx_zynq7 : gen0	lorks 2:0xfffff00 1 1	J
Loading Startin	/sdD: g at D	1/vxHork: x200000.	s1689104 		
Adding !	5489 s	ymbols fo	or standalor	e.	
100000 10000000 1000000 1000000 1000000 1000000 1000000 10000000 100000000	1999 199 1999 1	1111111 1111111 1111111 111111 111111	11111111111111111111111111111111111111	1))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))	11 1111 (R) 11 1111 1111 11111 11111 11111 11111 11111 11111 11111 11111 1111 11111 11111 1111 11111 11111 1111 11111 11111 1111 11111 1111 1111 11111 1111 1111 11111 1111 1111 111111 1111 1111 111111 1111 1111 111111 1111 1111 111111 1111 1111 111111 1111 1111 111111 1111 1111 111111 1111 1111 111111 1111 1111 1111111 1111 1111 1111111 1111 1111 11111111 1111 1111 11111111 1111 1111 111111111 1111 1111 11111111111111111111111111111111111 11111
->	auy.				

Figure 12: Boot Terminal Transcript

Building and Debugging the Application

Creating the Hello World Application

As short example, the following instructions describe how to build and download a small "Hello World" application to the remote target after you have set up and are running VxWorks.

The assumptions are:

- You have followed the prior stages of this document.
- VxWorks is executing on the target (with remote debug enabled).
- An Ethernet connection is present between the host and the target.

To create the "Hello World" application:

 Select File > New > Project > VxWorks Downloadable Kernel Module Project, as shown in Figure 13.

New Project	x
Select a wizard	>
Creates a new VxWorks downloadable kernel module project with all available kernel build specs	-
Wizards:	
type filter text	
Wind River Workbench Project	^
👂 🗁 General	
> 🧁 CVS	
> 🤄 Desktop	
b > JavaScript	E
VxWorks 6.x	
BY VXWorks Boot Loader / BSP Project	
H VXWorks Downloadable Kernel Module Project	
VXWorks Image Project	
VXWorks Real Time Process Project	
A A A A A A A A A A A A A A A A A A A	-
Show All Wizards.	
	_
(?) < Back Next > Finish Cancel	

Figure 13: New Project Wizard

2. Click **Next** and enter a project name; for example, **hello_world**, (Figure 14, page 18), then press **Finish**.

Project	
Creates a new kernel build sp	VxWorks downloadable kernel module project with all available pecs
Due is at a surrow	hells would
Project name:	nello_world
Location	
Oreate pro	ject in workspace
Create pro	ject at external location
Create pro	ject in workspace with content at external location
Directory: C	:\Xilinx_Working\vxworks_zynq_xapp\hello_world
2	

Figure 14: New Kernel Module Project

A hello_world project opens in the Project Explorer Window.

- 3. Move the mouse to the Project. Click the right button and select New > File.
- 4. Enter a file name; for example, hello.c, and click Finish.
- 5. Enter the following code into the file:

```
#include <stdio.h>
void hello()
{
printf("Hello Wind River\n");
}
```

6. In the Project Explorer window, select **Build Targets** (Figure 15), and click the right mouse button.



Figure 15: Build Target Selection

7. Select Build Options > Set Active Build Spec > ARMARCH7<gnu|diab>.

A message opens that asks if you want to set the active build spec to **ARMARCH7**<gnu|diab> (Figure 16).

Links, wante served from the original from the served formation of the se	Anno Radi Tangat, Anno, A	R Francisco Contraction of Contracti	iver(s*); — Chan Poper	
autoparal appCompth ppCompth ppCompth ppCompth ppCompth ppCompth autoparate autoparate autoparate autoparate autoparate autoparate ppComptm	Debug viewer Gener Tak Debug viewer Gener Tak Demonipaet Voltools Kenner Tak	€ Tatis P Terman come d'Amporta come d'Amporta come d'Amporta m d'Applicaté Semeng Applicaté come d'Amporta	Manage Active Band Spece. Manage Active Band Spece. Banded Center. Build Center. Build Center. Build Center. Build Center. Build Center. Build Center. The second second part of the same of reflect showship and data mark of the reflect showship and data mark of reflect showship and da	ADDRESS OF TABLE ADDRESS OF TABLE
Annote System: 12 Deal South System: 12 Deal South South Registries South Files S	a () () () () () () () () () (ansina - M det- cases - d' - elap-acti- eratoria - grand allegare - eratoria - grand allegare - eratoria - grand allegare - tera - traditionaria - tera - traditionaria - tera - delarizaria - tera - delarizaria - delarizaria - delarizaria - tera - delarizaria - tera - delarizaria - delarizaria - delarizaria - tera - delarizaria - delarizaria - delarizaria - delarizaria - delarizaria - delarizaria - delarizaria - delarizaria - delarizaria - delarizaria - delarizaria - delarizaria - delarizaria - delarizaria - delarizaria - delar	prop) (al.Worldness reservice A.Stargett Net inform downling, may doe, provide profiliant starting, may doe, provide profiliant starting, and profiliant does not Winned Network of the Start (reservice militant, starting, starting, and the Start maps, profiliant starting, starting, starting, starting, does and the starting of the Starting Starting, starting, start, starting, starting, starting, starting, starting, starting, starting, starting, starting, starting, starting, starting, starting, starting, starting, starting, starting, starting, starti	ancial SMAT(pu) SMAT(pu) SMAT(pu),PAL,JADE(SMAT(pu),JAE,JADE(SMAT(pu),JAE,JADE(SMAT(pu),JAE,JADE(SMAT(pu),JAE,JADE(SMAT(pu),JAE,JADE(SMAT(pu),JAE) SCALLipula SCALLipula SCALLipula More

Figure 16: Build Target Configuration

- 8. Select YES.
- 9. Build the project. A message opens.
- 10. Click Continue.
- 11. In the bottom-left corner of the Wind River Workbench, go to Remote Systems.
- 12. In the window, right-click and select New > Connection.
- 13. Select Wind River VxWorks 6.x Target Server Connection, and click Next.
- 14. Type the target IP address of **192.168.1.2**, and the path to the kernel image (in this case type: \..\<project_name>\default). Click Finish.
- 15. Right-click the new connection, and click **Connect**.
- 16. In the Project Explorer window, right-click **Build Targets**, then select **Debug VxWorks Kernel Task**.
- 14. The Debug Configuration window opens, as shown in Figure 17, page 20.

Debug Configurations	Street a street street street	x
Create, manage, and run confi Select the launch context for exec	gurations ting the launch operations.	ř.
	Name: hello - hello_world.out - VxWorks6x_192.168.1.2 (1)	
type filter text	🚰 Launch Context 🔄 🖞 Downloads) 😁 Projects to Build 💱 Source 🔲 Common	
ach Target Context C++ Application	VxWorks Kernel Task Options	ń
2++ Attach to Application 2++ Postmortem Debugger	Launch Context: 🧟 🕅 🕅 🗐 🌒 🏇	
-++ Remote Application inch Control get Communication Framework Vorks Kernel Task helio - helio_world.out - VxWork noEntryPoint - helio_world.out . Vorks Real Time Process	Acfault (localhost) A	ш
	▼ General	
	Entry Point: hello Browse	
	Arguments:	
	Advanced	
	▼ Debug	
	✓ Break on Entry	•
Filter matched 11 of 11 items	Apply Revert	
?	Debug	

Figure 17: Debug Configuration Window

17. From General > Entry, type hello to locate the hello world project.

Alternatively, you can use the Browse button to find the hello world project.

18. Click Debug.

The terminal window that is connected to the board issue the following message:

>> Break at 0x0144accc: hello +0x4 Task: 0x14cdfc0 (Hello)

Note: The message can differ slightly from yours as it depends on your Kernel settings. It just gives you the hint that a task was downloaded and stopped for debug.

19. In the debug window, click the Run button.

The terminal window displays: Hello Wind River.

- 20. Type i to see the tasks in the VxWorks task list.
- 21. Type **repeat 10, hello** to repeat the task 10 times.

Accessing a Peripheral in the Processing System

In the ARM[®] Cortex[™] A9 processor, every peripheral is memory mapped. The address map for the Zynq-7000 AP SoC processor, for example, is listed in the *Zynq-7000 AP SoC Technical Reference Manual (UG585)* [Ref 2].

Modifying the Hello World Application

Modify the "Hello World" application to access the GPIO peripheral. On the ZC702 board, MIO pin 10 is connected to an LED (DS12).

From the Zynq-7000 AP SoC Technical Reference Manual, (UG585) [Ref 6], you see that the base address of the GPIO peripheral is 0XE000A000. To access pin 10 as an output, you must configure this peripheral first:

- 1. Set the direction to output by writing a 1 to bit 10 of the gpio.DIRM_0 register.
- 2. Enable the output by writing a 1 to bit 10 of the gpio.OEN_0 register.
- 3. Write to bit 10 of the gpio.DATA_0 register to control the LED.

The updated source code of the hello world application now looks like the following:

#include <stdio.h>

```
#include <sys/mman.h>
#define GPIO_BASE 0xE000A000
#define GPIO_DIRM_0 0x00000204
#define GPIO OEN 0 0x00000208
#define GPIO_DATA_0 0x0000040
int main(void)
{
     printf("Hello World!\n");
int val = 0xfffffff;
sysOutLong(GPIO_BASE + GPIO_DIRM_0, 0x00000400);
sysOutLong(GPIO_BASE + GPIO_OEN_0, 0x00000400);
while (1) {
sysOutLong(GPIO_BASE + GPIO_DATA_0, val);
sleep(1);
val ^= 0xfffffff;
}
return 0;
}
```

4. Save this file, then build and debug this application following steps 9 to 19 of the previous section. The result is that the LED toggles every second.

```
Accessing a
Peripheral in
the
Programmable
Logic
```

Accessing a peripheral in the Programmable Logic is very similar to accessing a peripheral in the processing system: both master GP AXI interfaces have an address space of 1GB, as can be seen in the Address Map table in the *Zynq-7000 AP SoC Technical Reference Manual* (UG585) [Ref 6]. The differences are:

- You must first program the PL with a BIT file containing the AXI_GPIO peripheral.
- You must modify the VxWorks BSP to allow access to the address range that you configured for that peripheral.

The design created for this section contains an AXI_GPIO peripheral connected to the M_AXI_GPO port of the PS.

The four GPIO pins of the $\tt AXI_GPIO$ peripheral are connected to the DS15 to DS18 LEDs on the ZC702 board.



Figure 18: Zynq-7000 Processor System and Peripherals

The Address Editor shows the base address where this peripheral is mapped, as shown in Figure 19.

Cell	Base Name	Offset Address	Range	High Addr
- J /processing_system7_1				
E Data				
/axi_gpio_1	Reg	0x41200000	64K	0x4120FF

Figure 19: Address Editor

After implementing this design, you generate a new FSBL, and use this FSBL, together with the generated BIT file, to create a new boot.bin file to download to the SD card.

5. Follow the same steps as you used but use a slightly modified boot.bif file, as follows:

```
//ZC702_bif_for_VxWorks:
{
  [bootloader]fsbl.elf
  bitfile.bit
  bootROM.elf
```

}

Where:

- fsbl.elf is the new FSBL.
- bitfile.bit is the BIT created by the hardware design.

This produces a new boot.bin file to boot the ZC702 board.

6. To access the peripheral from within a VxWorks kernel module, first modify the BSP.

The default configuration of the VxWorks BSP configures the MMU to allow access to a limited set of addresses, listed in the documentation of the BSP.

Start	Size	End	Access to
0x0000_0000	1M	0x000F_FFFF	OCM
0x0010_0000	1019M	0x3FBF_FFFF	DDR3 SDRAM
0x0FC0_0000	4MB	0x0FFF_FFFF	ROM
0xE000_0000	32MB	0xE1FF_FFFF	I/O PERIPHERALS
0xF800_0000	6000B	0xF800_0BFF	SLCR
0xF8F0_0000	16KB	0xF800_0BFF	SCU
0xFC00_0000	16MB	0xFCFF_FFFF	QSPI FLASH
0xFFFF_0000	4KB	0xF800_0BFF	OCM

Adding the Address Range to the MMU Configuration

The M_AXI_GP ports are not mapped.

To add the address range to the MMU configuration, modify the VxWorks BSP:

- 1. Open the VxWorks image project, then browse to the /xlns_zynq7k folder.
- 2. Double-click the sysLib.c file to open the file in the Text Editor.



Figure 21: MMU Configuration

3. In the syslib.c file, scroll down to line 109, find the struct containing the MMU configuration.

For example, on line 225, find the mapping for the GPIO peripheral used in the previous example:

```
{
    ZYNQ7K_GPIO_BASE, /* Zynq-7000 gpio */ ZYNQ7K_GPIO_BASE,
    PAGE_SIZE,
        MMU_ATTR_VALID_MSK | MMU_ATTR_PROT_MSK |
    MMU_ATTR_DEVICE_SHARED_MSK,
    MMU_ATTR_VALID |MMU_ATTR_SUP_RWX|
    MMU_ATTR_DEVICE_SHARED
    },
4. Add the mapping for the AXI_GPIO peripheral by adding the following line:
```

- 5. Save the file, and rebuild the VxWorks project.
- 6. Use this VxWorks image, and either:
 - Put the image on the SD card to boot from SD, or
 - When fetching the VxWorks image over FTP, boot the board with the updated boot.bin file.
- 7. After booting the ZC702 board with this SD card, the FSBL configures the PL before launching the VxWorks BootROM.

The BootROM then loads the updated VxWorks image.

Updating the Hello World Project

Update the "hello world" project to access the AXI peripheral. The AXI_GPIO peripheral is slightly different from the hardened GPIO peripheral: you need only to set the direction, not enable the output driver.

Change the contents of the file to:

```
#include <stdio.h>
#include <stdio.h>
#include <sys/mman.h>
#define AXI_GPIO_BASE0x41200000
#define AXI_GPIO_TRI??????0x04
#define AXI_GPIO_DATA??????0x00
int main(void)
{
printf("Hello World!\n");
int val = 0;
sysOutLong(AXI_GPIO_BASE + AXI_GPIO_TRI, 0);
while (1) {
```

	SYSOUTLC	ong (AXI_G	PIO_BASE + AXI_GPIO_DATA, val);
	printf("	%d\n", v	al);
	sleep(1)	;	
	val++;	0100	(0) (m) = 0
	ll (Val	== 0X100	(val = 0;
	roturn (•	
	}	' <i>i</i>	
		مماداه ا	t and run it on the 70700 beard, it to raise the LEDs supervises and
		ia this projec	t, and run it on the 20702 board, it toggles the LEDs every second.
Conclusion	This application BSP on the Zyn overview of the	note has pro q-7000 SoC boot process	ovided step-by-step instructions for running the VxWorks 6.9.3.1 All Programmable device platform, and additionally provided an s for the Zynq-7000 AP SoC platform.
	You now know t	he steps for	using VxWorks RTOS on the Zynq-7000 AP SoC platform.
Additional	The following lir	nks are to ad	ditional resources referenced in this document:
Resources	1 Zung Z000		anaanta Taala and Taalaniguaa Ulaar Cuida (UC827)
	1. Zyriq-7000	AF 300 - 0	
	2. Zynq-7000	All Programi	nable SoC Software Developers Guide (UG821)
	3. Zynq-7000	AP SoC Tec	hnical Reference Manual, (<u>UG585</u>)
	4. Xilinx BSP:	gppve_6_9	_xlnx_zynq7k_6_9_2:
	https://porta	al.windriver.c	om/cgi-bin/windsuff/bsp/infoBSP.cgi?id=12020
	5. VxWorks 6.	9.3.1 BSP D	river Source Patch:
	https://suppo	rt.windriver.co	m/olsPortal/faces/maintenance/downloadDetails.jspx?contentId=041654
	6. VxWorks 6.	9.3.1 USB L	2 Cache Source Patch:
	https://suppo	rt.windriver.co	m/olsPortal/faces/maintenance/downloadDetails.jspx?contentId=041575
Revision	The following ta	ble shows th	e revision history for this document.
Revision History	The following ta	ble shows th	e revision history for this document.
Revision History	The following ta	ble shows th Version 1 0	e revision history for this document. Description of Revisions Initial Xilinx release
Revision History	The following ta Date 09/27/2013	ble shows th Version 1.0	e revision history for this document. Description of Revisions Initial Xilinx release.
Revision History	The following ta Date 09/27/2013	ble shows th Version 1.0	e revision history for this document. Description of Revisions Initial Xilinx release.

XAPP1158 (v1.0) September 27, 2013