Summary

VxWorks® is a multicore-capable Real-Time Operating System (RTOS), commonly used in fully featured embedded subsystems where a commercial runtime software and tool chain solution is required. Supporting a variety of CPU architectures, most notably the ARM Cortex™-A9 MPCore as available in the All Programmable Zynq®-7000 SoC platform. This application note focusses on using the all new (circa 2014) VxWorks 7 offering on this platform, for guidance on using the maturing VxWorks 6.9 release refer to XAPP1158 [Ref 1].

Introduction

This application note comprises the following major sections:

- VxWorks Project Types
- Source Build Project – Creation, Configuration and Build
- Kernel Image Project – Creation, Configuration and Build
- Target Bring-Up – Local and Network Boot
- Software Application (Generic) – Creation, Development and Debug
- Software Application (Peripheral Access) – Creation, Development and Debug (ZC702 Only)
- Appendix A: Development Environment Installation
- Appendix B: Customizing uBoot for the Avnet Mini-ITX Development Kit

Hardware and Software Requirements

Hardware Requirements (Optional)

1. Xilinx ZC702 Development Kit (XC7Z020-1CLG484)
2. Avnet Mini-ITX Development Kit (XC7Z045-2FFG900)

Software Requirements

1. Wind River Workbench 4.0
   (See Appendix A: Development Environment Installation for installation instructions.)
2. TFTP server application, such as tftpd32
3. Serial communication utility, such as Tera Term
VxWorks Project Types

VxWorks Source Build (VSB)

These are the kernel libraries. VxWorks 6.9 and previous versions ship with standard prebuilt libraries, but it might be useful to change the configuration/sources of a VSB to rebuild them for a specific need.

You must build custom libraries if you want to vary from the default binaries shipped with your platform. For example, if you want to change the default settings for endianness, enable SMP support, or enable real-time process support, you must specify the appropriate options and build custom libraries. To build custom libraries, you must create, configure, and compile a VxWorks source build (VSB) project.

VxWorks Image Project (VIP)

This is the image that you build and run on the target. It contains the kernel, BSP, architecture support, and VxWorks middleware (stacks, standard programs, etc.). It might also contain a specific type of read-only file system that is linked with the image (romfs).

A VxWorks image project (VIP) is used to create a version of VxWorks using components based on the precompiled libraries provided in the VxWorks installation. A VIP can use the standard components that are delivered with binary libraries, or it can be based on a VxWorks source build (VSB) project.

Downloadable Kernel Module (DKM)

This is similar to kernel modules in Linux (binaries that you can load and unload on-the-fly to add symbols to the kernel). This can be used to dynamically load applications/drivers that run in kernel space.

You can use DKM projects to manage and build modules that exist in the kernel space. You can separately build the modules, run, and debug them on a target running VxWorks, loading, unloading, and reloading on-the-fly. Once your development work is complete, the modules can be statically linked into the kernel, or they can use a file system if one is present.
Source Build Project – Creation, Configuration and Build

This section explains how to create, configure and build a VxWorks Source Build (VSB) image from the Zynq-7000 BSP that is natively installed with the Wind River environment. Such must be created before creating a VxWorks kernel image. These steps explain how to achieve this:

1. Invoke Wind River Workbench and open a new or existing workspace.
2. Select File > New > Project. The new project wizard opens.
3. Under VxWorks 7, select the VxWorks Source Build Project, as shown in Figure 1.

![Figure 1: Selecting the VxWorks Source Build Project](image)
3. Click **Next**. The *New VxWorks Source Build Project* wizard dialog box opens, as shown in Figure 2.

4. Enter a project name, for example, *Xilinx_Zynq7k_BSP*, and click **Next**.

**RECOMMENDED:** It is highly recommended that this be located outside of your workspace to enable reuse across developments – it is a 'one off build stage' and build times can approach 10-20 minutes.
5. Select the appropriate Zynq BSP in the **BSP**: drop down menu, as shown in Figure 3. This would be `xlnx_zynq7k_?_?_?_? | avnet_mini_itx_7z_?_?_?_?` for the pre-installed BSPs. Click **Finish**.

**Note:** BSP versions evolve and might be updated from that which is available at time of publication of this application note. Select that which is most suitable to your development.

6. Right-click **Xilinx_Zynq7k_BSP** and select the **Properties** option.

7. The **Properties view for Xilinx_Zynq7k_BSP** wizard opens. Click **Build Properties** and change the **Build command** to **make -jN** where N is the number of cores in your host machine. This speeds up the build time. Give the N value with respect to the host machine configuration and click **Apply**.
8. A confirmation wizard then opens, select Yes and then click OK. This reduces the BSP build time.

9. Now build the VSB Project. Right-click the project (Xilinx_Zynq7k_BSP) in the Project Explorer window and select the Build Project option. A parallel build enable confirmation box is launched, select Yes.

10. It takes 10-20 minutes for the BSP build process to complete. After completing the build process, create the VxWorks kernel image by following the procedures in the following section.

   Note: At time of publication, there are some minor ignorable build errors. Wind River is working on clearing these issues.
This section explains how to create a VxWorks kernel image from the VxWorks Source Build Project (VSB) created in the previous stage. It also explains how to configure the VxWorks kernel for the specific implementation requirements.

1. In the Wind River Workbench main context menu, select **File > New > Project**. The *New Project Wizard* opens as shown in **Figure 4**.

   ![Image of New Project Wizard](image)

   **Figure 4**: *Creating the New VxWorks Kernel Image Project for the Zynq-7000 AP SoC*

2. Under *VxWorks 7*, select **VxWorks Image Project** and click **Next**. A New *VxWorks Image Project Wizard* opens (also shown in **Figure 4**).

3. Enter a project name (for example: *Zynq_VxWorks_image*) and click **Next**.
4. For "Project" browse to your prebuilt VSB project directory location. Once selected, a single BSP will be listed and that should match your choice made in the previous VSB build guidelines section. (See Figure 5.) Click Next.

5. From the New VxWorks Image Project wizard, select PROFILE_DEVELOPMENT (Figure 5).

6. Click Finish. This creates the kernel image project in the Project Explorer window.
Note: The reference file target.ref in the generated VxWorks Image Project (see Figure 6) describes all of the flow and configuration options.

7. Before building the kernel image, these symbols must be added into the kernel configuration:

- **INCLUDE_STANDALONE_SYM_TBL** – Includes the Zynq-7000 AP SoC symbol table
- **INCLUDE_IFCONFIG** – Configures the Ethernet (ifconfig)
- **INCLUDE_DEBUG_AGENT_START** – Includes debug services
- **INCLUDE_DEBUG_AGENT** – Includes debug services

For example, the following steps detail how to achieve this for the first symbol:

a. Open the Kernel configuration by expanding the *Zynq_VxWorks_Image Project* in the *Project Explorer* window and click on the kernel configuration. The *Component* wizard, which lists kernel components, opens in the workspace.
b. Hover the mouse over the Description headline in the Component wizard and then right-click as highlighted in the Figure 7.

![Figure 7: Finding the Kernel Symbol in the VxWorks Kernel Configuration](image1)

b. Hover the mouse over the Description headline in the Component wizard and then right-click as highlighted in the Figure 7.

c. Click the Find option. The Find window opens. Enter the string INCLUDE_STANDALONE_SYM_TBL in the text box as highlighted in Figure 8.

![Figure 8: Entering the Kernel Symbol in the VxWorks Kernel to Find Configuration](image2)
d. Click **Find**. The INCLUDE_STANDALONE_SYM_TBL symbol is found and displayed in the **Matching** window, as shown in **Figure 8**.

e. Right-click INCLUDE_STANDALONE_SYM_TBL and select the **Include (Quick Include)** option as highlighted in **Figure 9**.

f. Repeat these steps for each of the remaining symbols.

**Note:** The VxWorks ARM BSP uses a Device Tree Blob (DTB) to describe device specific information, and there are two methods to load the DTB:

- Embedded: The DTB is embedded in the VxWorks image
- Independent: An independent file is created for the DTB.

This application note uses the independent DTB method to load the DTB.
8. In the Project Explorer window, move the mouse over to the image project (Zynq_VxWorks_image), right-click and select Build Project as shown in Figure 10. It takes few seconds for the kernel image to be built.

![Figure 10: Building the VxWorks Kernel Image Project](image)

9. Expand the Build Targets (default) which is present under the Zynq_VxWorks_image in the Project Explorer window.
10. Right-click \texttt{uVxWorks} and select the \textbf{Build Target} option as shown in the Figure 11. On completion the \texttt{uVxWorks} and \texttt{<BSPIncludedDeviceTree>.dtb} images are created at directory location \texttt{<WorkspacePath>}</texttt{<KernelImageProjectName>}</texttt{default}.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{Building_the_Kernel_Image.png}
\caption{Building the Kernel Image}
\end{figure}

\section*{Target Bring-Up – Local and Network Boot}

This section explains how to run the VxWorks image on the Zynq-7000 AP SoC using different boot modes. A boot image (BOOT.BIN) must be created using the first stage boot loader (FSBL), the PL bit stream (optional and needed if there is any PL logic), and u-Boot. u-Boot subsequently loads the VxWorks image and hands-off execution control to the VxWorks image. Suitable image stitching is a feature of the “bootgen” utility present in Xilinx’s Software Development Kit (XSDK), however prebuilt images are provided in the Wind River Workbench install for the aforementioned development boards.

For custom boards, a customized FSBL that matches the base hardware must be used and thus a boot image (boot.bin) must be specifically created, including the FSBL and a suitable uBOOT second stage loader. Guidance on how to include this can be found in the \textit{Xilinx Software Development Kit (SDK) User Guide (UG1145)} [Ref 2].
Booting VxWorks (Common)

uBoot can source the VxWorks boot image and device tree from either a local nonvolatile storage device or from network-attached storage, but the initial device boot files MUST be local on the onboard SD card (or Quad-SPI).

1. Copy the files for your reference board from 
   `<InstallDirectory>\vxworks-7\pkgs\os\board\arm\<Board>\_bootloader\*\bin` and rename to “boot.bin”. See Local Boot (SD Card Example), page 15 for additional files to copy if planning a full local boot.

2. Connect a power cable and a USB UART cable to the board.

3. See target.ref as highlighted earlier for guidance on how to configure the board’s boot switches for SD/QSPI, as required. Full details and explanations can also be found in the corresponding board’s user guides (see the support page and the ZC702 Evaluation Board User Guide (UG850) [Ref 3]).

4. Open a serial terminal emulator, choose the appropriate COM port, and set the baud rate to 115,200.

5. Turn on the ZC702 board after inserting the SD card.

6. Stop the Auto boot process of u-boot by pressing the keyboard Return key.

7. The u-boot execution prompt is displayed, as shown in Figure 12.

![Figure 12: U-Boot Execution Prompt](image)
Local Boot (SD Card Example)

For a local boot, complete the steps in Booting VxWorks (Common), page 14 and then:

1. Ensure that the uVxWorks and <BSPIncludedDeviceTree>.dtb files are copied onto an SD card from the directory path \<workspace>\<KernelImageProjectName>\default.

2. Assuming u-boot has booted, enter fatls mmc 0 at the prompt. The list of files present in the SD card is displayed.

3. Enter the fatload mmc 0 0x5000000 uVxWorks command at the Zynq u-boot prompt. This command copies the uVxWorks image to address 0x5000000 of the target memory space.

4. Enter the fatload mmc 0 0x4000000 <BSPIncludedDeviceTree>.dtb command at the Zynq u-boot prompt. This command copies the DTB image to address 0x4000000 of the target memory space.
5. Enter the **bootm 0x5000000 - 0x4000000** command at the Zynq u-boot prompt. The VxWorks image boots and echoes the output as shown in Figure 13.

![VxWorks SD Boot Mode Execution on the Zynq-7000 AP SoC](image)

**Figure 13**: VxWorks SD Boot Mode Execution on the Zynq-7000 AP SoC
Network Boot (TFTP Example)

To boot from the network, complete the steps in Booting VxWorks (Common), page 14 and then:

1. Identify the IP address of your host machine ensuring connection to the same ethernet subnet as your target board, or establish a point-to-point connection.
   
   Note: The IP addresses used in this application note are just used as an example. You need to change the network parameters according to your actual situation. An incorrect setting for the bootargs parameter can cause the procedure to fail.

2. Start the TFTP server with administrator permissions (right-click the tftp application > run as administrator) on the host machine. Store the path `\<workspace>\<KernelImageProjectName>\default` where the uVxWorks, `<BSPIncludedDeviceTree>` images are present. Ensure that a local firewall allows tftp to be used on your machine (or disable this firewall temporarily).

3. At the u-boot prompt, enter these commands to set the network IP addresses in the u-boot environment:

   setenv ipaddr 192.168.88.169  
   setenv serverip 192.168.88.170  
   setenv gatewayip 192.168.88.1  
   setenv blocksize 512  
   saveenv  

   Network parameters are saved in the u-boot environment.

4. Update and save the bootargs to the u-boot environment by entering these commands at the u-boot prompt:

   setenv bootargs 'gem(0,0)host:vxworks h=192.168.88.170 e=192.168.88.169:fffffe00 g=192.168.88.1 u=target pw=vxTarget f=0x0'  
   
   Note: This must be written/pasted on one line for the terminal CLI.

   saveenv  

5. Download the uVxWorks image from your host machine to address `0x5000000` of the target memory location using this command:

   tftpb 0x5000000 uVxWorks

6. Download the Device Tree Blob image from the host machine to address `0x4000000` of the target memory location using this command:

   tftpb 0x4000000 <BSPIncludedDeviceTree>.dtb
7. When the downloading process has completed run the image by entering this command at
the u-boot prompt:

    bootm 0x5000000 - 0x4000000

This results in the VxWorks kernel booting and you can see the boot logs on the serial
console.

8. At the end of the VxWorks kernel boot process the serial console display should be similar
to the example shown in Figure 14.

```
    synq-uboot> bootm 0x50000000 - 0x40000000
    ## Booting kernel from Legacy Image at 05000000 ...
    Image Name: vxWorks
    Image Type: ARM VXWorks Kernel Image (uncompressed)
    Data Size: 3015584 Bytes = 2.9 MB
    Load Address: 0x0200000
    Entry Point: 0x0200000
    Verifying Checksum ... OK
    ## Flattened Device Tree blob at 04000000
    Booting using the Fdt blob at 0x40000000
    Loading Kernel Image ... OK
    ## Starting vxWorks at 0x00200000, device tree at 0x04000000 ...
    Target Name: vxTarget
    
    VxWorks 7
    Core Kernel version: 1.0.8.0
    Build date: Feb 11 2015 16:00:52
    Copyright Wind River Systems, Inc.
    1984-2015
    
    Board: TIM 7000 Development Board
    OS Memory Size: 1024MB
    ED&F Policy Mode: Deployed
    Debug Agent: Started (always)
    Stop Mode Agent: Not started
    
    Adding 8963 symbols for standalone.
    >>
```

*Figure 14: VxWorks TFTP Boot Completion on the Zynq-7000 AP SoC*

9. This display indicates that the VxWorks kernel is up and running and the VxWorks shell can
be used to issue commands. Enter `help` for an overview of the available commands that are
supported by this kernel.
Establishing a Network Connection to VxWorks

1. Enter this command to set the IP address of the target, again substituting as appropriate to your environment:

   \texttt{ifconfig "gem0 192.168.88.169"}

   The actual IP address can be shown by entering:

   \texttt{ifconfig}

   The IP address configuration is displayed, an example is shown in \textit{Figure 15}.

   \begin{verbatim}
   > ifconfig gem0
   Link type:Ethernet HWaddr 00:0a:35:11:22:33
   capabilities: TXCSUM VLAN_MTU
   inet 192.168.88.169 mask 255.255.254.0 broadcast 192.168.89.255
   inet6 fe80::20a:35ff:fe11:2233%gem0 prefixlen 64 automatic
   UP RUNNING SIMPLEX BROADCAST MULTICAST
   MTU:1500 metric:1 VM:0 ifindex:2
   RX packets:116 mcast:0 errors:0 dropped:0
   TX packets:255 multicast:0 errors:0
   RX bytes:13k TX bytes:25k
   \end{verbatim}

   \textit{Figure 15: IP Address Configuration for the ZC702}
This section describes how to create, build, download, and debug a simple *Hello_World* application using the Wind River Workbench. Before executing the follow steps ensure that the VxWorks kernel image is executing on the target and that an Ethernet connection is present between the host and the target.

Create and Build an Application

The following steps describe how to build and download a small *Hello_World* application to the remote target after you have set up and are running VxWorks:

1. Select File > New > Project > VxWorks Downloadable Kernel Module Project, as shown in Figure 16.

![Wizard for Selecting the Downloadable Kernel Module Project](image)

*Figure 16:* Wizard for Selecting the Downloadable Kernel Module Project
2. Click **Next** and enter a project name; for example, *Hello_world* (see Figure 17).

![Figure 17: Naming the Kernel Downloadable Module](image1)

3. Click **Next** and select an **image project** in the *Project context/Based on* drop down menu, then click **Finish** (see Figure 18). A *Hello_world* project opens in the *Project Explorer* window.

![Figure 18: Selecting The Image Project](image2)
4. Expand the *Hello_world* project and click the `dkm.c` file. This is the C application file. It can be renamed for your application by right-clicking on it, selecting the *rename* option, and entering a new name; for example *hello_world.c*. Enter the following code in this file:

```c
#include <stdio.h>
void hello()
{
    while(1)
    {
        printf("Hello Wind River\n");
    }
}
```

5. To build the application, right-click *hello_world.c* and select the *Build Project* option.
Debug an Application using Wind River Workbench

Perform the following steps to debug the Hello_World application built in Create and Build an Application.

1. Click **New Connection**, (located on the top left side, under the menu option of the Wind River Workbench) to launch and configure the connection wizard. A New Connection wizard opens.

2. Select **Target Type: Running Target** and **Connection Mode: Application Mode**, as show in the **Figure 19**.

![Figure 19: Configuring the Debug Setup](UG570_c1_01_120913)
3. Enter the target board's IP address and port number (192.168.88.169:1534 for example) as the Target Address and browse to the kernel image which is present under the path \<KernelImageProjectName>\default\uVxWorks in the in kernel symbol file option, as shown in Figure 19. Click Finish.

4. Connect the board by clicking the connect symbol located beside the New Connection option:

You now have a properly established connection between the target and the Workbench, and are ready to debug the application.

5. Right-click on the Hello_world project and select the Run/Debug Kernel Task option.

6. Enter the first C function name that would get executed at running time of your application in the Entry Point option in the Run/Debug Kernel Task wizard. For the hello_world.c application, hello is the starting function. Also select the Attach Debugger and Break at Entry Point check boxes as shown in Figure 20.

![Figure 20: Launching the Debug Kernel Task](image-url)
7. Click **OK**. Application execution is stopped at the entry function hello(). The serial terminal connected to the board displays the following message:

```
Break at 0x005060e0: hello   Task: 0x201fe5e8 (tHello)
```

*Note:* The message can differ slightly as it depends on your Kernel settings. It just indicates that a task was downloaded and stopped for debug.

8. In the debug window, click **Run**. The terminal window displays **Hello Wind River**.

---

**Software Application (Peripheral Access) – Creation, Development and Debug (ZC702 Only)**

This section explains how to modify the *Hello_World* application to access the GPIO peripheral on the ZC702 board. MIO pin 10 is connected to an LED (DS23) on the ZC702 board. All of the peripheral address space is listed in the *Zynq-7000 All Programmable SoC Technical Reference Manual* (UG585) [Ref 4]. From that list, the base address of the GPIO peripheral is **0XE000A000**.

To access MIO pin 10 as an output, the following configuration must be implemented:

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 a **1** to bit 10 of the *gpio.DATA_0* register to control the LED.

The default configuration of the VxWorks BSP MMU table allows access to a limited set of addresses space in the Zynq-7000 AP SoC. You can access these limited set of addresses with their physical addresses directly from the user space. If you need to access the addresses which are not in this set, then that particular physical address needs to mapped to the addresses of the VxWorks BSP MMU table. This mapping is done using the API `pmapGlobalMap();`. You can check the list of default access addresses provided by entering the command `vmContextShow` at the VxWorks command prompt.
The following code snippet illustrates the configuration procedure:

```c
#include <stdio.h>
#include <vxWorks.h>
#include <vmLibCommon.h>
#include <pmapLib.h>

// Zynq-7000 GPIO Peripheral Details
#define GPIO_BASE 0xE000A000
#define GPIO_DIRM_0 0x00000204/sizeof(UINT32)
#define GPIO_OEN_0 0x00000208/sizeof(UINT32)
#define GPIO_DATA_0 0x00000040/sizeof(UINT32)

// Physical Address Mapping Library Wrapper Function
UINT32 *phyTovirt (UINT32 phyaddr, size_t size)
{
    UINT32 *virtAddr;
    virtAddr = (UINT32 *)pmapGlobalMap ((PHYS_ADDR) phyaddr, (size_t) size,
                                         MMU_ATTR_SUP_RW | MMU_ATTR_CACHE_OFF | MMU_ATTR_CACHE_GUARDED);
    if (virtAddr == PMAP_FAILED) {
        printf("pmapGlobalMap returned ERROR.\n");
        return 0;
    } else {
        printf("virtual address is %p.\n", virtAddr);
    }
    return virtAddr;
}

main()
{
    // Zynq-7000 GPIO Peripheral Configuration through Virtual Address Space
    UINT32 *GPIO_BASE_virtaddr = phyTovirt(GPIO_BASE, 0x1000);
    UINT32 val=0xffffffff;
    // Set GPIO Bit10 Direction to Out
    sysOutLong(GPIO_BASE_virtaddr + GPIO_DIRM_0, 0x00000400);
    // Set GPIO Bit10 OEn to Enable Output Driver
    sysOutLong(GPIO_BASE_virtaddr + GPIO_OEN_0, 0x00000040);
    while (1) {
        // Toggle GPIO content by XOR'ing value
        sysOutLong(GPIO_BASE_virtaddr + GPIO_DATA_0, val);
        sleep(1);
        val ^= 0xffffffff;
    }
    return OK;
}
```

Save this file, then build and debug this application following steps 7 to 10 of the previous section, **Debug an Application using Wind River Workbench**. The result is that the LED toggles every second.
Conclusion

This application note describes the procedures to use the Wind River Workbench to configure, build, compile, and debug the VxWorks BSP, kernel image, user space VxWorks RTOS application, and access the Zynq-7000 AP SoC processing system peripherals from user space by using the kernel services. These procedures support the building of a real-time system solution using a Zynq-7000 AP SoC and VxWorks RTOS. Example source code is provided to enable you to quickly start software development using the Wind River Workbench.

References

1. *Using VxWorks BSP with Zynq-7000 AP SoC (VxWorks 6.9)* (XAPP1158)
2. *Xilinx Software Development Kit (SDK) User Guide* (UG1145)
7. *Vivado Design Suite Tutorial - Embedded Processor Hardware Design* (UG940)
   - Lab 1: Building a Zynq-7000 Processor Design
   - Chapter 3: Boot and Configuration
   - Appendix A: Using Bootgen
   - VxWorks Device Driver Developer's User Guide, 6.0
   - VxWorks Application Programmer's User Guide 6.7
   - Wind River Workbench User's Guide, 3.1
Appendix A: Development Environment Installation

VxWorks is provided as an integral part of a comprehensive Eclipse-based development environment, namely “Wind River Workbench”. Offering project management, development and debug capabilities – all of which are leveraged throughout the remainder of this application note. Follow the these steps to install the environment:

2. Create a new account (or log in if you already have a Wind River Online support account).
3. Once logged in, click the download button and follow the instructions for installing your Wind River products. Make sure that the Xilinx Zynq-7000 Board Support Package (BSP) is selected at the time of Wind River product installation. For further details refer to Wind River support at http://www.windriver.com/company/contact/.

Appendix B: Customizing uBoot for the Avnet Mini-ITX Development Kit

This appendix explains how to make the BOOT.BIN with a u-boot source. Wind River modified the vendor-supplied u-boot source code to support booting up the uVxWorks image.

1. You can retrieve the U-boot source code from the Xilinx repository by using this command:
   `git clone git://git.xilinx.com/u-boot-xilinx.git`
3. Checkout the u-boot source code and switch the git branch to a particular version of u-boot source by using this command:
   `cd u-boot-xilinx and git checkout -b xilinx-v2013.4`
   **Note:** Refer to document Ubuntu_on_Zynq_Tutorial_03.pdf located in the Zynq_MiniITX_z7045_Ubuntu_FAT_v2013_4 folder for detailed information and follow the same steps to configure any version of u-boot to support the Zynq-Mini-ITX development board.
4. Unzip the file Zynq_Mini-ITX_z7045_Ubuntu_FAT_v2013_4.zip and find files boards.cfg and zynq_mitx.h in the Zynq_Mini-ITX_z7045_Ubuntu_FAT_v2013_4 directory and copy boards.cfg to the top directory and zynq_mitx.h to the /include/configs directory of the u-boot source code.
5. Compile the u-boot source code to generate the u-boot executable. Before this install the proper cross compiler, assuming your cross compiler is called, for example, arm-xilinx-linux-gnueabi-gcc. You can compile u-boot source code by entering the following series of commands:`
export CROSS_COMPILE=arm-xilinx-linux-gnueabihf-
make distclean
make zynq_mitx_config
make

6. After executing this series of commands, a file with the name u-boot is created. Rename the u-boot file u-boot.elf.

7. Use the zynq_fsbl.elf file from the Zynq_Mini-ITX_z7045_Ubuntu_FAT_v2013_4 directory and the u-boot.elf file created in above steps to create the BOOT.BIN image. The BOOT.BIN image needs to be packaged by the Xilinx tool bootgen, available in the Xilinx SDK tool.

### Revision History

The following table shows the revision history for this document.

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Revision</th>
</tr>
</thead>
<tbody>
<tr>
<td>05/08/2015</td>
<td>1.0</td>
<td>Initial Xilinx release</td>
</tr>
</tbody>
</table>

### Please Read: Important Legal Notices

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available “AS IS” and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of Xilinx’s limited warranty, please refer to Xilinx’s Terms of Sale which can be viewed at http://www.xilinx.com/legal.html#tos; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx’s Terms of Sale which can be viewed at http://www.xilinx.com/legal.html#tos.

**Automotive Applications Disclaimer**

XILINX PRODUCTS ARE NOT DESIGNED OR INTENDED TO BE FAIL-SAFE, OR FOR USE IN ANY APPLICATION REQUIRING FAIL-SAFE PERFORMANCE, SUCH AS APPLICATIONS RELATED TO: (I) THE DEPLOYMENT OF AIRBAGS, (II) CONTROL OF A VEHICLE, UNLESS THERE IS A FAIL-SAFE OR REDUNDANCY FEATURE (WHICH DOES NOT INCLUDE USE OF SOFTWARE IN THE XILINX DEVICE TO IMPLEMENT THE REDUNDANCY) AND A WARNING SIGNAL UPON FAILURE TO THE OPERATOR, OR (III) USES THAT COULD LEAD TO DEATH OR PERSONAL INJURY. CUSTOMER ASSUMES THE SOLE RISK AND LIABILITY OF ANY USE OF XILINX PRODUCTS IN SUCH APPLICATIONS.

© Copyright 2015 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.