ZedBoard™ Linux Hardware Design Project Guide

1300 NE Henley Court, Suite 3
Pullman, WA 99163
(509) 334 6306 Voice | (509) 334 6300 Fax

Revision: January 8, 2013

Table of Contents

Introduction

Use this *ZedBoard Linux Hardware Design Project Guide* with the Digilent Linux Repository to run an embedded Linux system on the ZedBoard Zynq™-7000 Development Board. The package includes a pre-built u-boot binary and source files, which you can use to build your First Stage Boot Loader (FSBL) and hardware bitstream. The FSBL, bitstream and u-boot binaries allow you to create a Zynq boot image, or BOOT.BIN, that will boot the Xilinx Zynq-7000. Moreover, you can boot and run all of the programmable SoC features from an SD Card. Those interested in designing custom embedded Linux systems need this guide as a reference tool to make the requisite modifications needed to suit their target application.

This reference manual describes the contents of the ZedBoard Linux Hardware Design package and details how to build and test the various sources. We generated all Xilinx files using the Xilinx Embedded Development Kit (EDK) version 14.2.

Package Contents

Begin by extracting the package contents to your desired directory with no spaces in the file path. We will refer to this location as <pkg>. Here is a brief overview of the included files:

- **boot_image:**
  - system.bit – This bitstream configures the programmable logic
  - u-boot.elf – Second-Stage boot loader that loads Linux
  - zynq_fsbl.elf – First-Stage boot loader that configures the processing system
- **doc:**
  - ProjectGuide.pdf – A copy of the *ZedBoard Linux Hardware Design Project Guide*
  - DemoFeatures.txt – Contains a description of the features incorporated into the prebuilt Linux image included with this package
- **hw:**
  - XPS project designed in Xilinx EDK 14.2 that generates the hardware platform and bitstream
- **linux:**

  - devicetree.dts – The device tree source code for the prebuilt device tree
- **sd_image:**
  - BOOT.BIN – The Zynq boot image generated using the three files found in the boot_image folder
  - devicetree.dtb – A prebuilt device tree
  - ramdisk8M.image.gz – A prebuilt ramdisk file system
  - zImage – A prebuilt Linux kernel
- **sw:**
  - The hardware platform
  - The first-stage bootloader source

Hardware Configuration

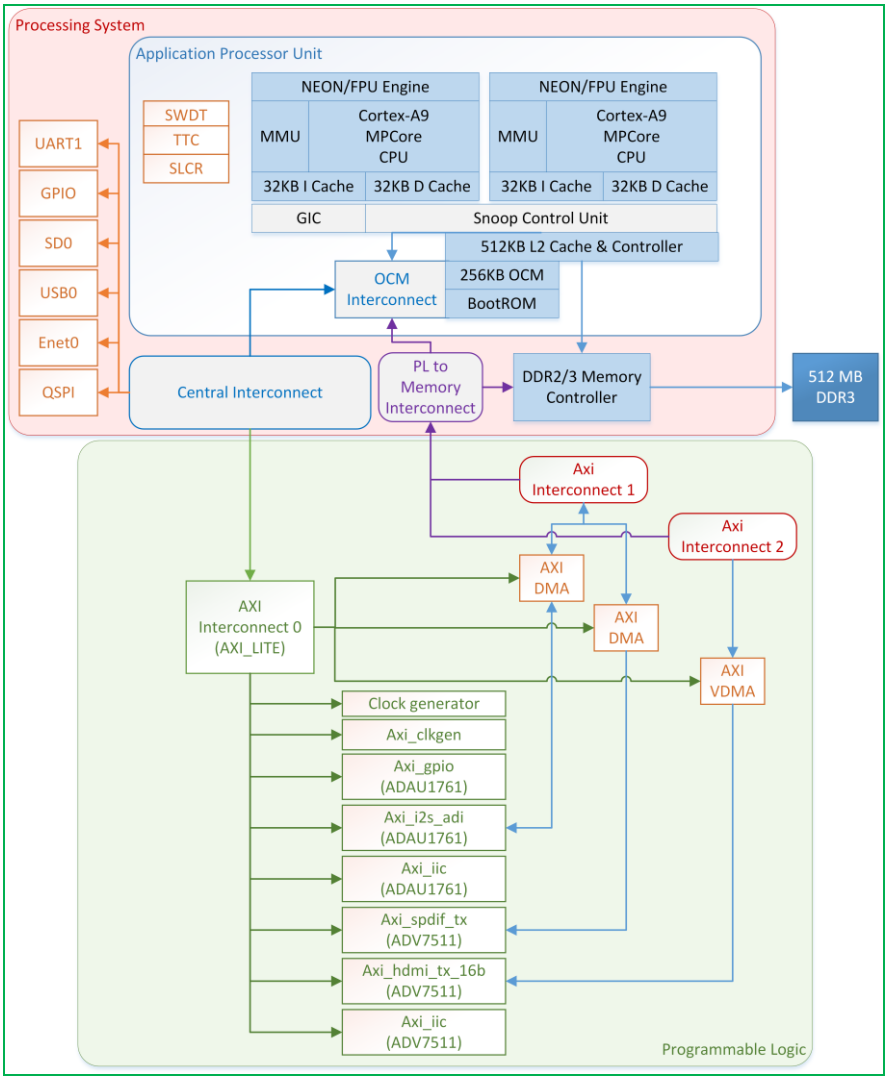

Figure 1. ***Basic Hardware System Architecture for ZedBoard***

Figure 1 is a block diagram of the ZedBoard hardware design. The ZedBoard Zynq Processing System (PS) cores support all of the attached hardware in the following list.

- **USB-OTG** - The USB-OTG PHY connects to USB controller 0 of the Zynq PS. The PHY reset signal operates through the PS GPIO controller. This reset signal is toggled within the FSBL to initialize the PHY. The over current detect signal from the USB-OTG PHY is inverted using an util_vector_logic core in the programmable logic and then connected to the proper pin on the PS.
- **UART** - The USB-UART bridge connects through UART controller 1 of the PS.
- **Pmod™ Ports** - The Pmod ports all connect to the PS GPIO controller. You can find drivers for most of the Digilent Pmods in the Digilent Linux Repository. See the available documentation for your hardware in the repository for more specific information on using the individual Pmod drivers. For specific GPIO pin mappings, see <pkg>\hw\xps_proj\data\system.ucf. (**Note:** When referencing these pins in Linux, you should add 54 to the number found in the .ucf.)
- **Switches/LEDs/Pushbuttons** - These devices connect to the PS GPIO controller.
- **OLED Display** - This device connects with the PS GPIO controller. The Pmod OLED driver controls this device in Linux.
- **SD Card Slot** - This card slot connects via the SD Controller 0 of the PS.
- **Ethernet** - The Ethernet port connects through the Ethernet Controller 0 of the PS.

The cores in the Zynq Programmable Logic (PL) support the following hardware:

- **HDMI Video** - A custom pcore in the PL (axi_hdmi_tx_16b) accesses video data stored in a framebuffer using an axi_vdma core. This data is encoded and sent to the ADV7511 chip.
- **HDMI Audio** - A custom core (axi_spdif_tx) allows access to sound data from DDR3 via an axi_dma core. This data is then encoded and sent to the ADV7511 chip.
- **HDMI Control** - An axi_iic core connects to the control port of the ADV7511 chip.
- **I^{2}S Audio** - A custom pcore in the PL (axi_i2s_adi) allows you to read and write sound data in DDR3 via an axi_dma core. This core implements an I^{2}S interface that can simultaneously receive and send audio data from the ADAU1761 audio codec. (**Note:** Linux does not yet support the ADAU1761 audio codec.)
- **I2S Control** - An axi_iic core is connected to the control port of the ADAU1761 chip. An axi_gpio core is also connected to the ADDR0 and ADDR1 lines of the ADAU1761 chip. When these two signals are left high (default value) then the resulting IIC address is 0b0111011. (**Note:** Linux does not yet support the ADAU1761 audio.)

Interrupts

The design package contains several signals from the programmable logic that trigger interrupts on the ARM core. To successfully connect these interrupts to drivers in the device tree, it is necessary to know the corresponding identifiers. (See Table 1 for the interrupt identifiers.)

Device tree identifier	Connected port in PL	Signal Description
59	axi_vdma_0 :: mm2s_introut	“Memory map to stream” VDMA interrupt for HDMI video data
58	axi_dma_spdif :: mm2s_introut	“Memory map to stream” DMA interrupt for HDMI sound data
57	External Port :: hdmi_int	External interrupt from the ADV7511 HDMI controller
56	axi_iic_hdmi :: IIC2INTC_irpt	IIC interrupt for HDMI control port
55	axi_dma_i2s :: mm2s_introut	“Memory map to stream” DMA interrupt for I2S sound data
54	axi_dma_i2s :: s2mm_introut	“Stream to memory map” DMA interrupt for I2S sound data
53	axi_iic_i2s :: IIC2INTC_irpt	IIC interrupt for I2S control port

Table 1. *Interrupt Identifiers*

Building the Hardware Bitstream and Platform

Follow steps 1-4 to build the Xilinx Platform Studio (XPS) generated files that describe your hardware platform.

1. Open the project found at <pkg>\hw\xps_proj\system.xmp in XPS 14.2.
2. Click “Export Hardware Design to SDK…” in the Project menu to open a dialog box.
3. Check “include bitstream and BMM file” in the open dialog box. Then press Export Only.
4. Wait for the process to finish building, this can take a while. Once the process is complete, you can locate the hardware platform files at <pkg>\hw\xps_proj\SDK\SDK_Export\hw\ and the bitstream at <pkg>\hw\xps_proj\SDK\SDK_Export\hw\system.bit.

Building the First Stage Boot Loader

Follow steps 1-6 to build the FSBL in the Xilinx Software Development Kit (SDK).

1. Open SDK 14.2 and create a new workspace at a location of your choosing. We will refer to this location as <wrk>.
2. Import the Hardware Profile
    a. Under the File menu, click New and then “Xilinx Hardware Platform Specification” to open a dialog window
    b. Click the browse button in this dialog window seen circled in figure 2. You must now select the hardware platform specification file. If you are using the hardware platform that you built in XPS, then it will be located at <pkg>\hw\xps_proj\SDK\SDK_Export\hw\system.xml. If you are using the prebuilt hardware platform included with this package, then select <pkg>\sw\hw_platform\system.xml.

Note: The prebuilt hardware platform does not contain a bitstream. We omitted it to reduce the overall size of this package. If you need to add the bitstream to the hardware profile, it may be copied from <pkg>\boot_image\system.bit.

c. Click “Finish”

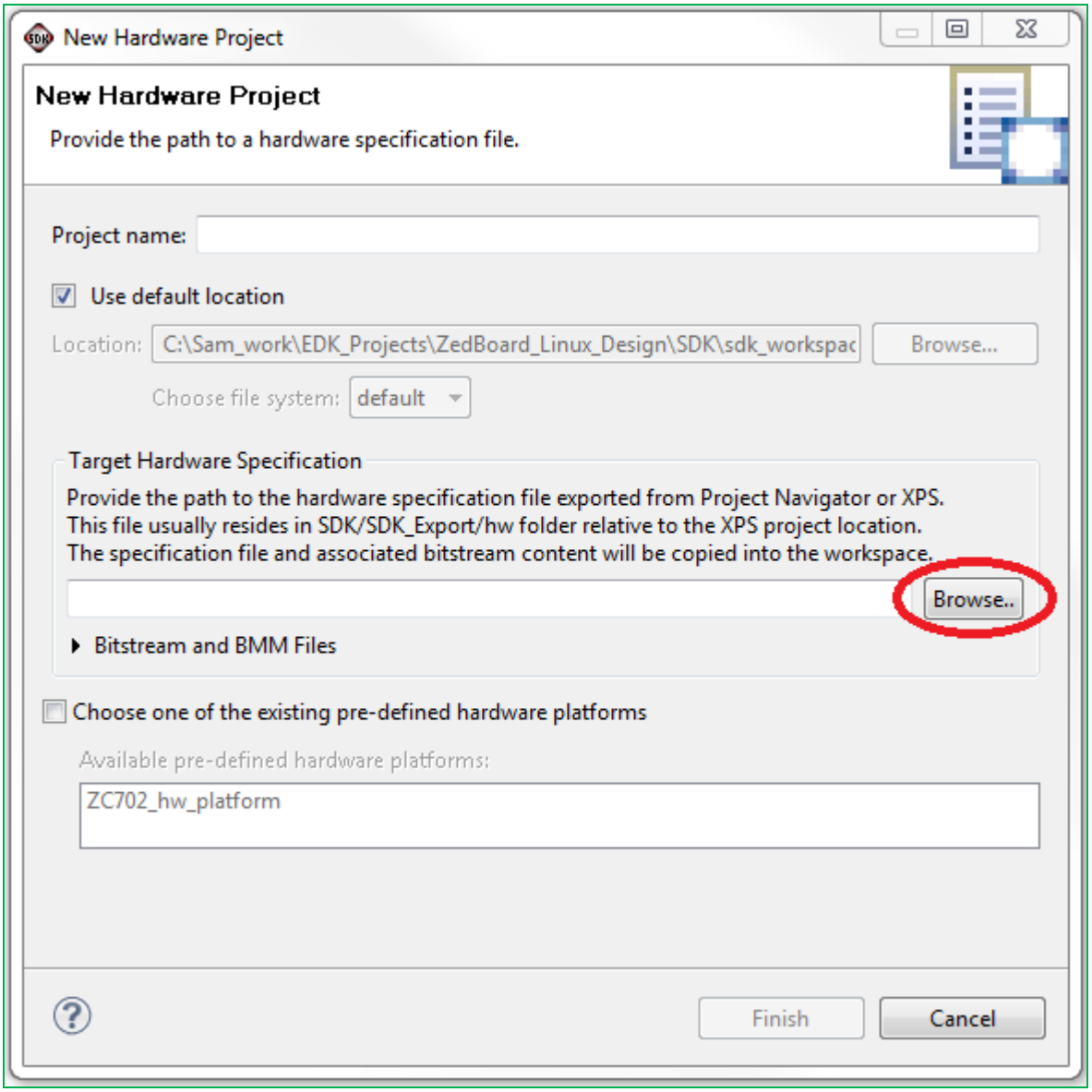

Figure 2. ***Importing HW Specification***

3. Generate a new Zynq FSBL

   a. Under the File menu, click New and then “Xilinx C Project” to open the dialog window.

   b. In the dialog window, choose “Zynq FSBL” under the “Select Project Template” header. Leave all other options as defaults and then press Next.

c. Leave all options as defaults again and click Finish.

4. Replace <wrk>\zynq_fsbl_0\src\main.c with <pkg>\sw\zynq_fsbl\src\main.c.

   Note: The main.c file included with this package is identical to the automatically generated one, except that it includes a small portion of code that resets the USB PHY.

5. Under the Project menu, click Clean, select “Clean all projects,” and then press OK. Pressing OK should clean the project and then immediately rebuild it.

6. After the rebuild has completed, you can find the FSBL binary at <wrk>\zynq_fsbl_0\Debug\zynq_fsbl_0.elf.

U-boot Source

The u-boot binary included in this package originates from the Digilent u-boot source code available on GitHub. Please refer to the Embedded Linux Development Guide, available on the Digilent website, for instructions on obtaining and building this source code. The process is not difficult, but it does require a computer running Linux.

Check out the git repository with the “v2012.04-digilent-13.01” tag and then run “make zynq_zed_config” to configure the source as it was when the u-boot binary was built.

Building a Zynq Boot Image

Follow steps 1-4 to generate an image that you can use to boot a Zynq device from an SD card. This image will execute the FSBL, which is responsible for configuring the processing system, the programmable logic, and then handing off execution to u-boot.

1. Open Xilinx SDK with a workspace of your choosing.

2. Under the Xilinx Tools menu, click “Create Boot Image” to open a dialog window.

3. Create u-boot.bin

   a. In the dialog window, set the Bif file to “Create a new Bif file…”

   b. Browse to select the desired FSBL elf file. This can either be one you generate or the one included with this package at <pkg>\boot_image\zynq_fsbl.elf

   c. Click the Add button to add a partition to the boot image and then select your preferred bitstream. This can either be one you generate or the one included with this package at <pkg>\boot_image\system.bit

   d. Click the Add button again and select the u-boot image that you would like to use. The one included with this package may be found at <pkg>\boot_image\u-boot.elf.

   e. Set the Output Folder to the directory where you would like the boot image placed.

   f. Click Create Image to place a file named u-boot.bin in the selected output folder.

4. In order for the Zynq device to recognize the boot image, you must rename it from u-boot.bin to BOOT.BIN.

Testing a Zynq Boot Image

This package includes a prebuilt Linux kernel, file system, device tree, and Zynq boot image. Operators can use these four files to boot a fully functional Linux system on the ZedBoard. This capability allows you to verify that a boot image operates according to your desired specifications. Perform steps 1-11 to boot Linux on the ZedBoard using these files.

1. Obtain an SD card and card reader. Format the SD card as FAT32.

2. Copy devicetree.dtb, zImage, and ramdisk8M.image.gz to the SD card from <pkg>\sd_image\.

3. Copy either a Zynq boot image that you have generated or the prebuilt boot image located at <pkg>\sd_image\BOOT.BIN to the SD card.

4. Eject the SD card from the computer and insert it into the ZedBoard.

5. Set the jumpers on the ZedBoard as follows:

    - MIO 6: set to GND
    - MIO 5: set to 3V3
    - MIO 4: set to 3V3
    - MIO 3: set to GND
    - MIO 2: set to GND
    - VADJ Select: Set to 1V8
    - JP6: shorted
    - JP2: shorted
    - Leave all other jumpers unshorted

6. Attach a computer running a terminal emulator to the UART port with a USB micro cable. Configure the terminal emulator as follows:

    - Baud: 115200
    - 8 data bits
    - 1 stop bit
    - no parity

7. Attach a 12V power supply to the ZedBoard and power it on.

8. Connect to the appropriate port in the terminal emulator. Once connected, you should begin to see feedback from the boot process within a few seconds. Feedback response times depend on the speed of the SD card.

9. Wait for the boot process to complete. You will know boot-up has completed when pressing return at the terminal presents you with a red "zynq>" prompt.

10. Verify the various features of the Linux demo. For a list of the available features, refer to <pkg>\doc\DemoFeatures.txt.

11. Once you are done, run the command “poweroff” at the terminal. You can then switch the ZedBoard off.

Additional Resources

Consult the following documents for additional information on designing embedded Linux systems for Digilent system boards.

- *Getting Started with Embedded Linux – ZedBoard*
  This document describes how to obtain the Linux Hardware Design and use it with the Digilent Linux repository to build and run a fully functional Linux system on the ZedBoard. You can obtain this document from the ZedBoard product page on the Digilent website.

- *Embedded Linux Development Guide*
  This document describes the differences between conventional Linux Development and Linux Development for Digilent system boards. Anyone who plans on tweaking the kernel or adding device drivers should read this guide. You can obtain the *Embedded Linux Development Guide* from the embedded Linux product page on the Digilent website.

- *Embedded Linux Hands-on Tutorial – ZedBoard*
  This document walks the reader through the process of modifying the ZedBoard Linux Hardware Design to include additional hardware. The guide shows how to make this hardware accessible to Linux by modifying the device tree and designing a custom driver that brings the hardware’s functionality up to the Linux operator. You can obtain this guide from the ZedBoard product page on the Digilent website.

- *Linux Developer’s Wiki*
  This web page contains an up to date list of hardware that is supported by the Digilent Linux repository and an FAQ section that addresses some issues you may run into while using the current version of the kernel. It also contains information on submitting patches for those who are interested in contributing code. You can find the Linux Developer’s Wiki at: www.github.com/Digilent/linux-digilent/wiki.

The structure of this design package stems from the Xilinx Zynq-7000 Base Targeted Reference Design (TRD). For more information on this design, see the Xilinx website at www.xilinx.com.