ECEN 449: Microprocessor System Design
Department of Electrical and Computer Engineering
Texas A&M University

Prof. Sunil P. Khatri

Lab exercise created and tested by:
Kushagra Gupta, Cheng-Yen Lee, Abbas Fairouz, Ramu Endluri, He Zhou,
Andrew Douglass and Sunil P. Khatri

Laboratory Exercise #4

Linux boot-up on ZYBO Z7-10 board via SD Card

Jan2024

Objective

The purpose of lab this week is to get Linux up and running on the ZYBO Z7-10 board. There are many advantages to running an Operating System (OS) in an embedded processor environment, and Linux provides a nice open-source OS platform for us to build upon. This week, you will use Vivado to build a Zynq(ARM Cortex A9) based microprocessor system suitable for running Linux, and you will also use PetaLinux tools to compile the Linux kernel based on the specification of your custom microprocessor system. You will then combine the bit stream with FSBL(First Stage Boot Loader) and u-boot(Universal Boot Loader) to create Zynq Boot Image. FSBL initialize the Processing System(PS) with configuration data and initializes u-boot. u-boot is the boot loader that holds the instructions to boot the Linux Kernel. We will use all these files to boot Linux on ZYBO Z7-10 board using a SD card.

System Overview

The microprocessor system you will build in this lab is depicted in Figure 1. As in last lab, this system has a Zynq processor, an UART Peripheral, and a custom multiplication peripheral. Unlike last lab, however, this system has an SD Card, a timer and a DDR3 (Double Data Rate v3 Synchronous Dynamic Random Access Memory) controller. Additionally, the PS itself has a Memory Management Unit (MMU) and instruction and data cache. These additional peripherals along with the MMU within the PS are required to run Linux. The SD card controller provides SD card read/write access. The DDR3 SDRAM controller provides the system with 1GB of RAM where the Linux kernel can reside. The interrupt controller enables interrupt handling necessary for interaction with I/O. The MMU within the PS enables virtual memory, which is required to run a mainstream Linux kernel. The PS(ARM Cortex A9) cache is added to improve performance, as DDR SDRAM accesses have high latency. Interrupts to the processor are addressed by Generic Interrupt Controller(GIC). The timer is necessary for certain OS system calls.

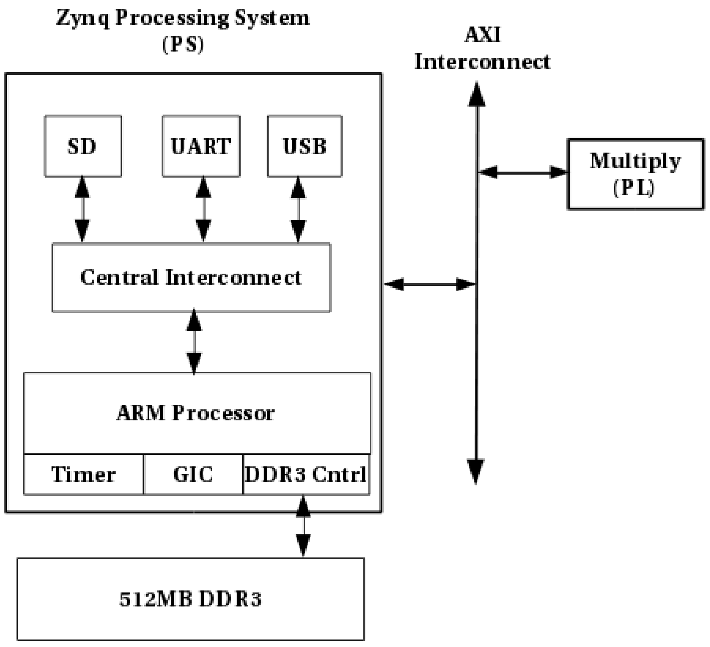


Figure 1: Zynq System Diagram

Procedure

1. To begin, we must create a base PS system which includes all the peripherals shown in Figure 1 .

   (a) Create a new directory by name lab4, open Vivado and create a new project as explained in Lab 3. Make sure to select ZYBO Z7-10 board in part window.

   (b) Next create block design, add ‘ZYNQ7 Processing System’ (PS) IP, run block automation and import ‘ZYBO_Z7_B2.tcl’ file’ as shown in Lab 3.

   (c) Now double click on PS IP, it will open ‘Re-customize IP’ window. Go to ‘Peripheral I/O pins’ tab and observe the peripherals that are enabled in the system.

   (d) Enable SD 0, UART 1 and TTC 0 peripherals by clicking the tick mark on the corresponding peripheral and disable the remaining peripherals.

   (e) Copy ’ip_repo’ folder which contains ‘multiply’ IP from Lab 3 and paste into Lab 4 directory.

   (f) Click on ’Settings’ under the ‘PROJECT MANAGER’ section in Vivado. Expand ’IP’ in the project setting window and click on ‘Repository’. Click on the blue ’+’ sign and add the ’ip_repo’ directory as shown in Figure 2. Click apply to add multiply IP to the current project IP catalog. Now you can add multiply IP to the base system.

   (g) Add multiply IP to the base system.

   (h) Run connection automation as shown in Lab 3. Now the base system should look like the one as in Figure 3

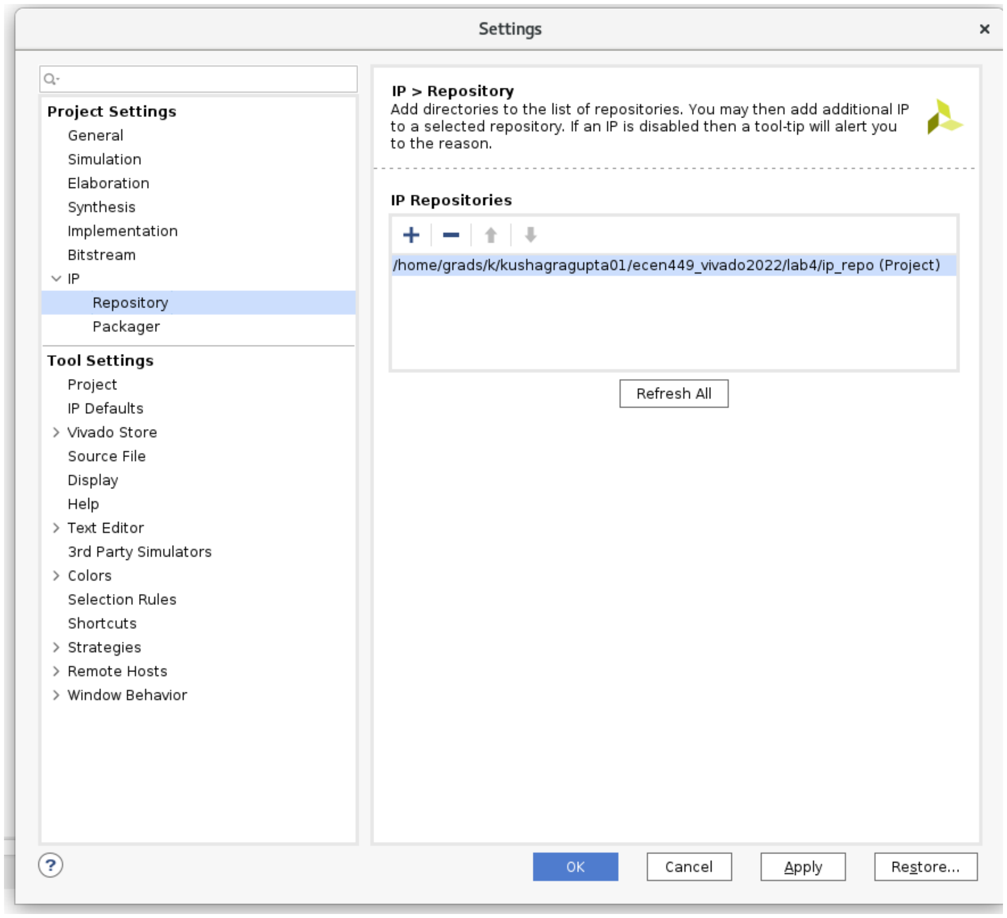
   (i) Create HDL wrapper to the base system as shown in previous lab.

   (j) Click on ’Generate Bitstream’ to create the bit stream file. Export the design (.xsa file) including bitstream similar to previous labs. Now that we have bitstream, next is to generate files needed for the Linux boot-up.

2. Linux boot files generation using PetaLinux

   (a) PetaLinux is an embedded Linux Software Development Kit (SDK) targeting FPGA-based system-on-chip (SOC) designs. For PetaLinux files, you need to arrange a fresh USB 3.2 thumb-drive of size at-least 64 GB and format [1] it as "ext4". It is highly recommended to reserve this USB drive for the PetaLinux usage during the remaining labs of ECEN499. Do not use it for other purpose.

[1] In the CentOS desktop machine, go to Applications → Utilities → Disks → <select your thumb-drive>. Then click on the gear icon and select ‘Format Partition’ option. In the ‘Format Volume’ window, type the name you want to set for your drive, enable ‘Erase’ option, and select the ‘Type’ as ‘Ext4’.



IP > Repository

Add directories to the list of repositories. You may then add additional IP to a selected repository. If an IP is disabled then a tool-tip will alert you to the reason.

IP Repositories

/home/grads/k/kushagragupta01/ecen449_vivado2022/lab4/ip_repo (Project)

Figure 2: Add IP repository

(b) Insert the USB drive and copy PetaLinux installer from the shared path to the <USB-drive-directory>. On the desktop machine, <USB-drive-directory> will be /run/media/<Net-ID>/<drive-name>.

```
$ cd <USB-drive-directory>
$ cp /mnt/lab_files/ECEN449/petalinux-v2022.1-04191534-installer.run .
```

(c) Set the required GCC version:

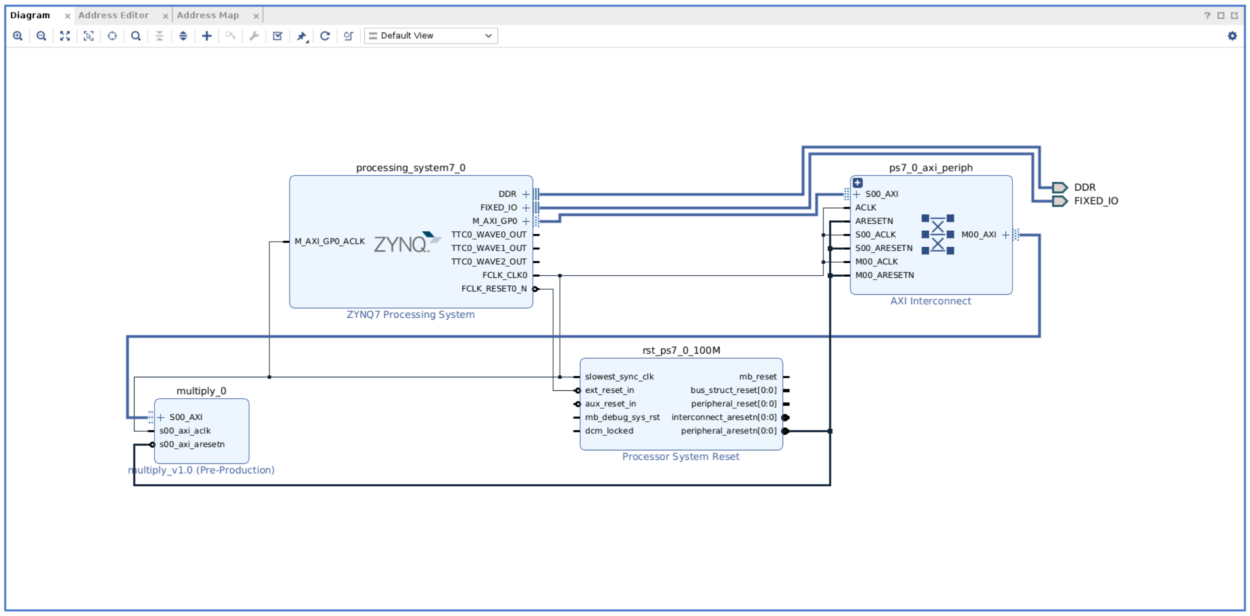


Figure 3: Base System Design

$ scl enable devtoolset-9 bash

(d) Install PetaLinux in a directory (for example petalinux_install) within the <USB-drive-directory> Please format your drive to 'ext4' (step 2a) before the installation.
$ cd <USB-drive-directory>
$ mkdir petalinux_install
$ chmod 755 ./petalinux-v2022.1-04191534-installer.run
$ ./petalinux-v2022.1-04191534-installer.run - -dir petalinux_install - -platform "arm"
Note: You can safely ignore the 'No tftp server found' warning
Note: This step can take up to 30 minutes to complete

(e) Once the installation has succeeded, source PetaLinux environment settings:

$ source petalinux_install/settings.sh

Once settings are sourced successfully, you can start using PetaLinux commands. Note: You need to source PetaLinux environment settings (step 2.d) every time you open a new terminal.

(f) Create PetaLinux project with a name (for example linux_boot) using the zynq template:

```
$ petalinux-create -t project - -template zynq -n linux_boot - -tmpdir /tmp/<Net-ID>
```

This step creates the project configured for Zynq processor-based systems. Note: The tmp directory provided as - -tmpdir should be unique to every new project you create.

(g) Move to PetaLinux project directory and reconfigure PetaLinux project with the hardware (.xsa file) you exported in step 1:

```
$ cd linux_boot
$ petalinux-config - -get-hw-description=<XSA-file-path> - -silentconfig
```

(h) Build PetaLinux project:

```
$ petalinux-build
```

After the above statement executes successfully, verify that the files exist and their timestamps are current (i.e. the timestamps match the completion time of the latest PetaLinux build). These files are located in the images directory in the PetaLinux project directory. List the files using the following commands:

```
$ cd images/linux
$ ls -al
```

The two files we are looking for are boot.scr and image.ub. Check that they exist and that their timestamp is current. boot.scr is the script that U-Boot reads during boot time to load the kernel and rootfs. image.ub contains kernel image, device tree and rootfs.

(i) Generate the zynq boot image (BOOT.bin) using the following command:

```
$ petalinux-package - -boot - -fsbl zynq_fsbl.elf - -fpga system.bit - -u-boot u-boot.elf
```

This step combines zynq_fsbl (first stage boot loader), u-boot (second stage boot loader), and system.bit (FPGA bitstream) to create BOOT.bin (zynq boot image) which is used for the system boot-up.

(j) We will boot Linux on the Zybo board using a SD card. Copy the BOOT.BIN, image.ub, and boot.scr files to the SD card.

3. Boot Linux on ZYBO Z7-10

(a) Open a new terminal and source settings64.sh. We will use PICOCOM to watch the output from the ZYBO Z7-10 board.

(b) Change JP5 into SD card mode to boot from the SD card and power on the ZYBO Z7-10 board. Connect the USB cable to ZYBO Z7-10 board. Run the following command to start PICOCOM.

```
$ picocom -b 115200 /dev/ttyUSB1
```

(To exit press Ctrl-A and then Ctrl-x to exit picocom.)

(c) Disconnect the SD card from the PC and plug it into the ZYBO Z7-10 board. Press the reset button(PS-SRST) to start booting Linux. If the process is successful you should see Linux booting up on the ZYBO Z7-10 board via the PICOCOM console. Use the username as "petalinux" and set the password as "root". By default, the Linux will boot for a non-root user, you can switch to the root user using the following command:

```
$ sudo su - (password is root)
```

Demonstrate this to the TA.

Deliverables

1. [6 points.] Demo Linux booting on the ZYBO Z7-10 board to the TA.

   Submit a lab report with the following items:

2. [8 points.] Correct format including an Introduction, Procedure, Results, and Conclusion. Be sure to summarize the process required to build the hardware and compile the Linux kernel.

   Warning: Missing information will result in missing points.

3. [2 points.] The output of the terminal (picocom) showing the Linux boot.

4. [4 points.] Answers to the following questions:

   (a) Compared to lab 3, the lab 4 microprocessor system shown in Figure 1 has 512 MB of SDRAM. However, our system still includes a small amount of local memory. What is the function of the local memory? Does this ‘local memory” exist on a standard motherboard? If so, where?

   (b) After your Linux system boots, navigate through the various directories. Determine which of these directories are writable. (Note that the man page for ’ls’ may be helpful).

   Test the permissions by typing ‘touch <filename>’ in each of the directories. If the file, <filename>, is created, that directory is writable. Suppose you are able to create a file in one of these directories. What happens to this file when you restart the ZYBO Z7-10 board? Why?

   (c) If you were to add another peripheral to your system after compiling the kernel, which of the above steps would you have to repeat? Why?