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 Ramu Endluri, He Zhou and Sunil P Khatri)

Laboratory Exercise #4

Linux boot-up on ZYBO board via SD Card

Objective

The purpose of lab this week is to get Linux up and running on the ZYBO 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 open source 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 need a RAMDISK, a temporary file system that is mounted during Kernel boot. The boot loader needs a device tree which has information about the physical devices in the system and this information is stored in 'device tree blob'. We will use all these files to boot Linux on ZYBO 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 512MB 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.

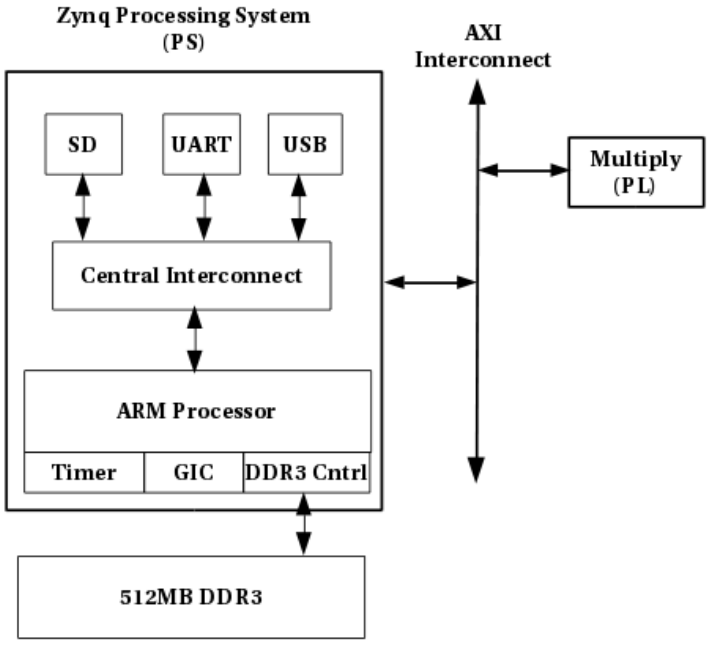

Figure 1: Zynq System Diagram

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 new project as explained in Lab 3. Make sure to select ZYBO board in part window.
   (b) Next create block design, add 'ZYNQ7 Processing System' (PS) IP, import 'ZYBO_zynq_def.xml' file' as shown in Lab 3.

(c) Run block automation.

(d) 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.

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

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

(g) Click on ’Project Settings’ under project Manager tab. Click on ’IP’ in the project setting window. Click on the green ’+’ 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.

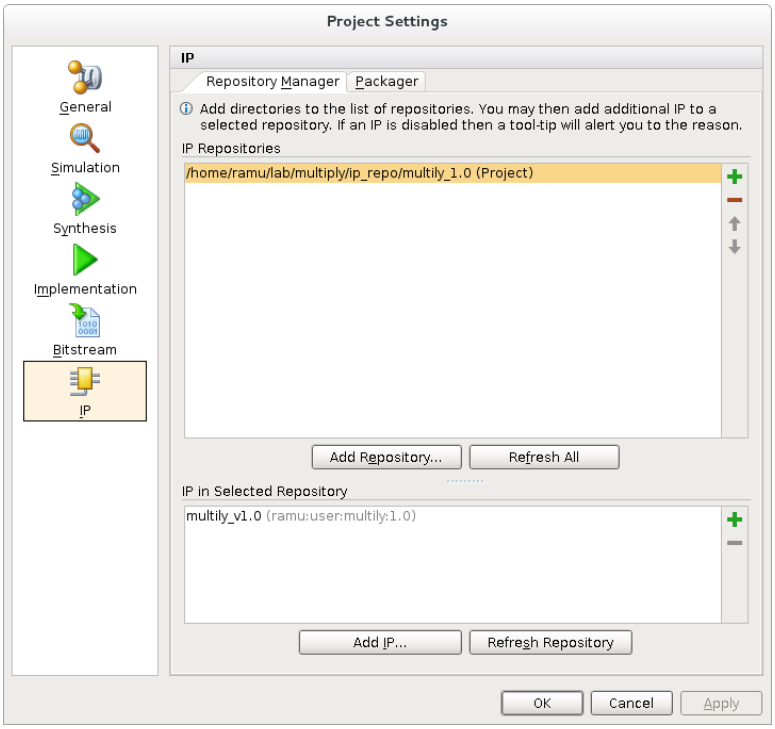

Figure 2: Add IP repository

(h) Add multiply IP to the base system.

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

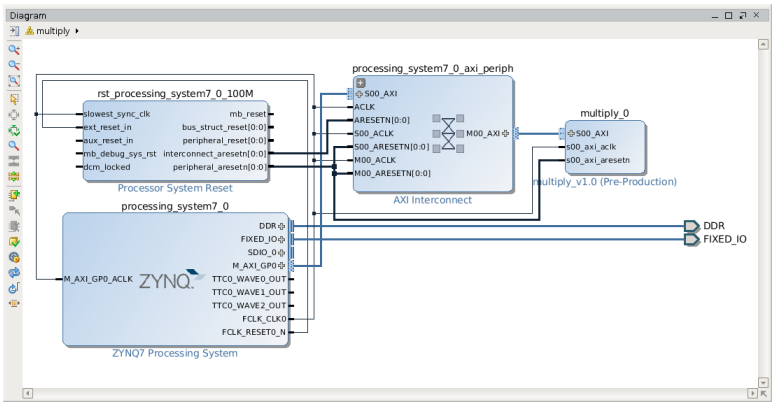

Figure 3: Base System Design

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

(k) Click on 'Generate Bitstream' to create the bit stream file. Now that we have bit stream, next is to compile u-Boot file.

2. u-boot

(a) Untar the '/homes/faculty/shared/ECEN449/u-boot.tar.gz' file by executing the following command from within your lab directory
>tar -xvzf /homes/faculty/shared/ECEN449/u-boot.tar.gz

(b) To compile U-Boot, we need cross-compile tools which are provided by Vivado . Those tools have a prefix arm-xilinx-linux-gnueabi- to the standard names for the GCC tool chain. The prefix references the platforms that are used. In order to use the cross-platform compilers, make sure the Vivado 2015.2 settings have been sourced.

(c) Since this is an universal boot loader we need to configure it to the target device, in our case our ZYBO board. To configure and build U-Boot for ZYBO, run the following commands in u-boot directory:
> make CROSS_COMPILE=arm-xilinx-linux-gnueabi- zynq_zybo_config
NOTE:make sure to source settings64.sh before executing command

(d) You should see the terminal message 'Configuring for zynq_zybo board...'

(e) Now we have configured u-Boot, to compile run the following command in terminal.
> make CROSS_COMPILE=arm-xilinx-linux-gnueabi-

(f) After the compilation, the ELF (Executable and Linkable File) generated is named u-boot. Add a .elf extension to the file name so that Xilinx SDK can read the file layout and generate boot.bin. The u-boot file can be found under u-boot folder.

(g) Next step is to generate boot.bin. Open Vivado and export bitstream to SDK as shown earlier in Lab 3. Make sure to select 'include bitstream' while exporting the design.

(h) After SDK launches, the hardware platform project is already present in Project Explorer on the left of the SDK main window. We now need to create a First Stage Bootloader (FSBL). Click File->New->Application Project.

(i) In the new project window, name the project as FSBL and click on Next, in the Templates window select 'Zynq FSBL' and click on Finish.

(j) Click on Project and select 'Build All' to build the project.

(k) Now we have all of the files ready to create BOOT.BIN. Click Xilinx Tools -> Create Zynq Boot Image.

(l) In the Create Zynq Boot Image window (as shown in Fig. 4), Click Browse to set the path for output.bif. Give the path to your lab4 directory.

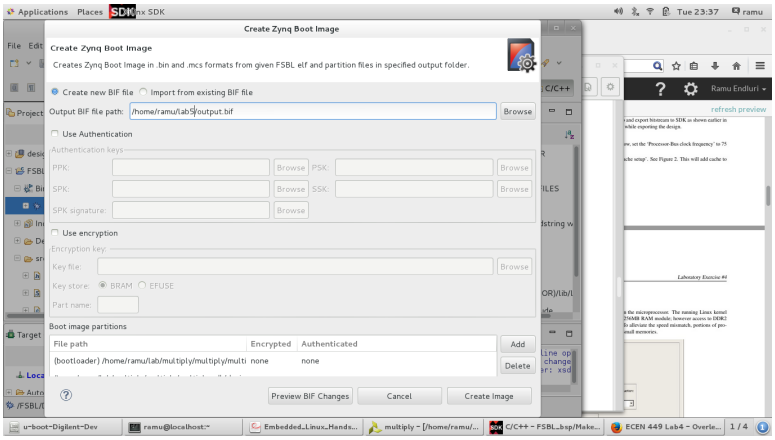

Figure 4: Zynq Boot Image

(m) FSBL as the name suggests it should be the first file in the boot image. Click add in the boot partitions section to add FSBL.elf. In the 'add partition' window, select 'Browse' and FSBL can be found under the directory lab4/project_name.sdk/FSBL/Debug/. Make sure the partition type is set as bootloader. This will indicate FSBL as the initial boot loader. Select 'OK'

(n) Next step is to add the bitstream generated. Bitstream(.bit file) can be found under project folder at /project_name.sdk/design_name_wrapper_hw_platform_0/. Hit 'Add' again to add the file and set the partition type as datafile and click 'OK'.

(o) Now add u-boot file created earlier in the lab. Hit 'Add' again to add the file and select partition type as datafile.

(p) It is very important that the 3 files are added in this order, or else the FSBL will not work properly . It is also very important that you set FSBL.elf as the bootloader and system.bit and u-boot.elf as data files. Click Create Image and the created BIN file is named as BOOT.bin and you should be able to find it in the lab4 directory.

3. Linux Kernel

(a) Untar the file Linux kernel source code file 'linux-3.14.tar.gz' from ECEN449 shared folder into your lab 4 directory.

(b) The Linux kernel can be configured to run on several devices. We need to configure the kernel with the default configuration for our board ZYBO. The configuration is located at arch/arm/-configs/xilinx_zynq_defconfig.

(c) We need a cross compiler to build linux for ARM processor on Zynq. Navigate to linux source code folder 'linux-3.14' under the lab 4 directory and to use the default configuration type the following command.

$ make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi- xilinx_zynq_defconfig

(d) Now that we have configured linux for our hardware we can compile it. Type the following command to compile linux.

> make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi-

(e) After successful linux compilation, you should see arch/arm/boot/zImage ready in the last few lines in terminal which is the kernel image.

(f) The kernel image is located at arch/arm/boot/zImage. However, in this case the kernel image has to be a uImage (unzipped) instead of a zimage. To convert image we need u-boot tools and we should include them in the PATH to use them. To include them in the path type the following command.

> PATH=$PATH:<directory_to_u_boot>/tools

(g) The tools are ready and to make the uimage type the following command from Linux directory.

> make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi- UIMAGE_LOADADDR=0x8000 uImage.

(h) Once the image is converted you should see uImage in lab4/linux-3.14/arch/arm/boot/

(i) To boot the Linux operating system on the ZYBO board, you need BOOT.BIN, a Linux kernel image (uImage), a device tree blob (.dtb file), and a file system. BOOT.BIN and uImage have

been created earlier in the lab. We will now compile the .dtb file. The default device tree source file is located in the Linux kernel source at arch/arm/boot/dts/zynq-zybo.dts.

(j) We have added a custom IP to the hardware system, hence we need to create an entry describing the 'multiply' IP in .dts file.

(k) 'multiply' IP is connected to the PS through an axi interconnect. Hence under 'ps7_axi_interconnect_0: amba@0' add the following lines after line 331.

```
multiply {
        compatible = "ecen449,multiply";
        reg = <0x43C00000 0x10000>;
};
```

(l) The reg entry holds the address of the multiply module which is the address allocated by Vivado in the 'Address Editor' tab to the multiply ip.

(m) Before we boot Linux we need to convert the .dts file to a .dtb(device tree binary) format which is the compatible format for arm. Execute the following command from the Linux source directory to convert the .dts file to .dtb.

```
> ./scripts/dtc/dtc -I dts -O dtb -o ./devicetree.dtb arch/arm/boot/dts/zynq-zybo.dts
```

(n) Copy the ramdisk file to the ECEN 449 directory. To wrap header file we need to use mkimage tools in u-boot directory. Run the following command from the ECEN 449 directory:

```
> ./u-boot/tools/mkimage -A arm -T ramdisk -c gzip -d ./ramdisk8M.image.gz uramdisk.image.gz
```

(o) The above command will create the uramdisk.image.gz file which we will use to boot linux

(p) Now we have all the files ready and its time to boot Linux on the ZYBO board. Copy the BOOT.bin, uImage, uramdisk.image.gz and devicetree.dtb file on to the SD card .

4. Boot Linux on ZYBO

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

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

```
$ picocom -b 115200 -r -l /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 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 board via the PICOCOM console. Demonstrate this to the TA.

Deliverables

1. [6 points.] Demo Linux booting on the ZYBO 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 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?