Lab #4:

Linux boot-up on ZYBO board via SD Card

Khanh Nguyen

UIN# 525000335

ECEN 449– 503

Date: February 16, 2017

INTRODUCTION:

The purpose of this lab is to learn how to boost Linux OS using Zybo Processing System. I created all the necessary booting files and loaded them into a flash card then boot the Linux through the fpga board.

PROCEDURE:

1/ Create a new project and choose Zybo as the board.

2/ Add the IP of ZynqQ7 Processing System then import ZYBO_zynq_def.xml into XPS settings in the Customize IP window.

3/ Enable UART1, SD0, TTC0 and disable all other peripheral pins

4/ Add multiply IP created from lab3 to the design.

5/ Create HDL wrapper and generate bitstream then export to SDK.

6/ Untar the u-boot.tar.gz and cross-compile the u-boot.

7/ Generate BOOT.bin by creating and build FSBL project in SDK.

8/ Untar the linux-3.14.tar.gz file from ECEN449 shared folder.

9/ Creating kernel image by running cross compiler linux configuration for ARM processor.

10/ Convert zImage into uImage.

11/ Modify the zynq-zybo.dts by adding 'multiplt' IP code.

12/ Convert the .dts file to .dtb file.

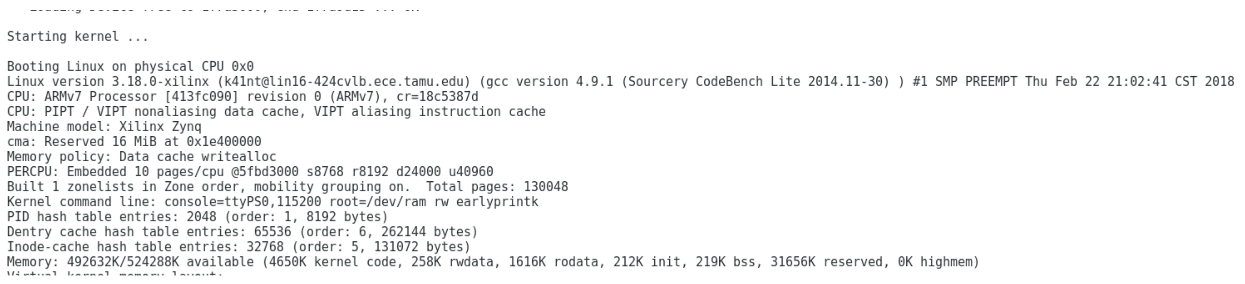
13/ Make the ramdisk by copying the ramdisk file from ECEN449 shared folder.

14/ Open picocom terminal, copy BOOT.bin, uImage, uramdisk.image.gz and devicetree.dtb on to SD card then plug it into the Zybo board.

15/ Hit reset button for the Zybo board to boot Linux.

RESULT:

The Linux OS successfully booted onto Zybo board.



```
Starting kernel ...

Booting Linux on physical CPU 0x0
Linux version 3.18.0-xilinx (k41nt@lin16-424cvlb.ece.tamu.edu) (gcc version 4.9.1 (Sourcery CodeBench Lite 2014.11-30) ) #1 SMP PREEMPT Thu Feb 22 21:02:41 CST 2018
CPU: ARMv7 Processor [413fc090] revision 0 (ARMv7), cr=18c5387d
CPU: PIPT / VIPT nonaliasing data cache, VIPT aliasing instruction cache
Machine model: Xilinx Zynq
cma: Reserved 16 MiB at 0x1e400000
Memory policy: Data cache writealloc
PERCPU: Embedded 10 pages/cpu @5fbd3000 s8768 r8192 d24000 u40960
Built 1 zonelists in Zone order, mobility grouping on.  Total pages: 130048
Kernel command line: console=ttyPS0,115200 root=/dev/ram rw earlyprintk
PID hash table entries: 2048 (order: 1, 8192 bytes)
Dentry cache hash table entries: 65536 (order: 6, 262144 bytes)
Inode-cache hash table entries: 32768 (order: 5, 131072 bytes)
Memory: 492632K/524288K available (4650K kernel code, 258K rwdata, 1616K rodata, 212K init, 219K bss, 31656K reserved, 0K highmem)
```

Result sreenshot

CONCLUSION:

I learned how to configurate and create all the necessary files to boot Linux onto a Zybo board.

The procedures are complex and the instructions are confusing sometimes (paths to necessary files are incorrect).

QUESTIONS:

1/ 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?

This local memory is used to store important start up instructions. These instructions determine what devices are connected to the board and what location it needs to boot the OS from. This local memory is in registers located on the microprocessor.

2/ 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 ' in each of the directories. If the file, , 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?

The file will be erased after the Zybo board restarts because it was saved on RAM.

3/ 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?

If another peripheral is added, all the above steps must be repeated because the new information of the hardware needs to be updated.