ECEN 749: Microprocessor Systems Design

Lab 4: Linux boot-up on ZYBO Z7-10 board via SD Card

Manav Gurumoorthy

830000011

Section 603

Table of Contents

Introduction 3
Procedure 4
  Configuration 4
  U-boot 4
  Boot.bin 4
  Linux Kernel 4
  Device tree binary 4
  ramdisk 4
  Booting Linux on ZYBO Z7-10 5
Results 5
Conclusion 6
Questions 7

Introduction

The objective of this lab was to boot Linux on the ARM Cortex A9 Processor on the Zybo board. For booting Linux on the processor 4 files were generated. They are a BOOT binary file, the devicetree, the operating system image file and a ramdisk file. The BOOT is a first stage boot loader. The FSBL is responsible to do some initializations it also fetches the secondary boot loader. The secondary bootloader fetches the ramdisk onto memory. The ramdisk contains an initial root file system to which the operating system is mounted to. Control is passed on to the secondary bootloader (U-Boot) . U-boot gets the image file which boots into the operating system. In this case the FSBL was BOOT.bin, the second stage boot loader was U-boot and this loaded the ramdisk and fetches the kernel.

Procedure

Configuration

The initial configuration for this lab like the previous lab requires the creation of a block design of the Zynq system. The Zynq system had the following peripheral devices enabled: SD 0, UART 1 and TTC 0. The multiply IP created in lab 3 was copied into the active directory of this project.

Once the IP block was added and the block design was completed, connection automation was run.

U-boot

U-boot is the name of the boot loader used in the boot process of the Linux kernel. U-boot was extracted using the 'tar' command within the working directory for the lab. As u-boot was compiled on an x86 machine and the compiled version of u-boot was targeted to run on an ARM chip, a tool called a cross compiler was used. The cross-compiler and other tools were provided through the Vivado software. The gcc compiler had the prefix 'arm-xilinx-linux-gnueabi-'. Also since the tools are a part of Vivado, the settings were sourced before u-boot compilation.

u-boot was also configured to be run on the zybo-z7-10 board by using the make command.
make CROSS COMPILE= arm=Xilinx-linux-gnueabi- zynq_zybo_config

After the configuration of u-boot was completed, u-boot was compiled. A file gets generated called uboot, the extension '.elf' (executable and linkable file was added to this generated file.

Boot.bin

For the creation of BOOT.bin binary file the Vivado SDK was used. In the create new project window, ZYNQ FSBL was selected, where FSBL is First Stage Boot Loader. The project was then built. In the Xilinx Tools menu, 'Create Zynq Boot' was selected which was used to create the BOOT.bin file.

In the Zynq boot image window the path for the output file (output.bif) was set. The boot image consists of 3 files and the order in which they are listed in the file is very important. The files in the boot.bin file are the FSBL.elf, then the generated bit-stream and then the u-boot.elf that was previously created.

Linux Kernel

With the BOOT.bin file ready, the next step was to compile and create an image of the Linux kernel. Like the compilation of u-boot, the Linux kernel is also cross-compiled using the Vivado GCC toolchain.

The result of the successful compilation of the Linux kernel yielded a zImage file. This is the bootable file of the Linux kernel. In the case of this lab the kernel must be in an unzipped format, i.e. uImage.

Device tree binary

To boot the Linux OS on the Zybo-Z7-10 board a device tree binary is needed along with the boot.bin and uimage files. The device tree source file was present in the Linux source code. In this code details about the use of the AXI interconnect and it's use to interface with the custom multiply IP.

The last step before Linux boot up was to convert the .dts file into a .dtb file.

ramdisk

The ramdisk file was copied into the working directory. The file was used to create a image file for the ramdisk. The command used to create the image was using the following command:

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

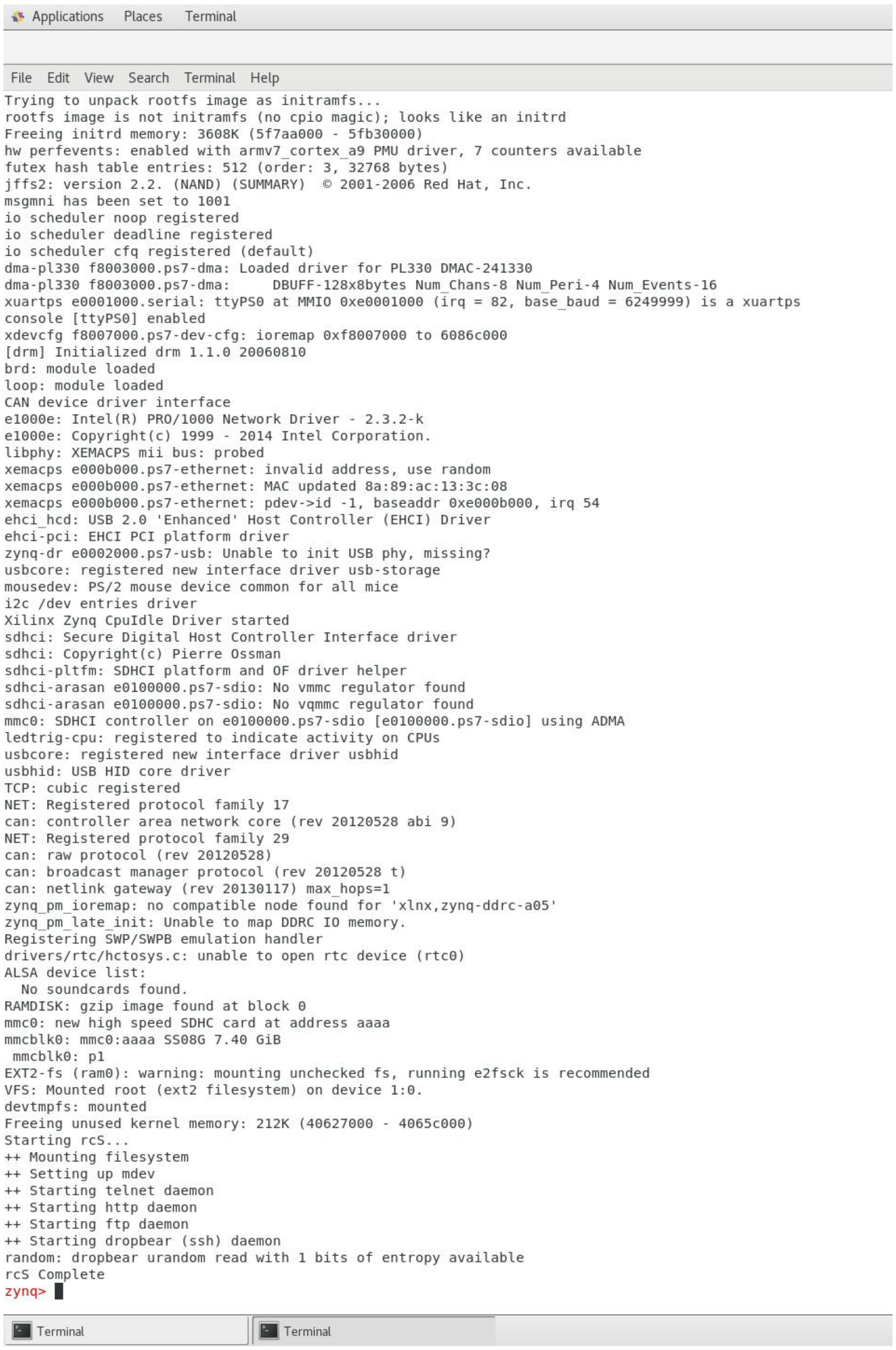
Booting Linux on ZYBO Z7-10

To view the output of the boot from the Zybo board, PICOCOM was used. The jumper on the ZYBO board was changed to the SD mode.

The SD card was inserted into the Zybo board and switched on. The linux boot up process was observed on the PICOCOM console.

Results

The results of the Linux boot up was observed from PICOCOM, a screenshot of the same is attached below.



```
Trying to unpack rootfs image as initramfs...
rootfs image is not initramfs (no cpio magic); looks like an initrd
Freeing initrd memory: 3608K (5f7aa000 - 5fb30000)
hw perfevents: enabled with armv7_cortex_a9 PMU driver, 7 counters available
futex hash table entries: 512 (order: 3, 32768 bytes)
jffs2: version 2.2. (NAND) (SUMMARY)  © 2001-2006 Red Hat, Inc.
msgmni has been set to 1001
io scheduler noop registered
io scheduler deadline registered
io scheduler cfq registered (default)
dma-pl330 f8003000.ps7-dma: Loaded driver for PL330 DMAC-241330
dma-pl330 f8003000.ps7-dma:     DBUFF-128x8bytes Num_Chans-8 Num_Peri-4 Num_Events-16
xuartps e0001000.serial: ttyPS0 at MMIO 0xe0001000 (irq = 82, base_baud = 6249999) is a xuartps
console [ttyPS0] enabled
xdevcfg f8007000.ps7-dev-cfg: ioremap 0xf8007000 to 6086c000
[drm] Initialized drm 1.1.0 20060810
brd: module loaded
loop: module loaded
CAN device driver interface
e1000e: Intel(R) PRO/1000 Network Driver - 2.3.2-k
e1000e: Copyright(c) 1999 - 2014 Intel Corporation.
libphy: XEMACPS mii bus: probed
xemacps e000b000.ps7-ethernet: invalid address, use random
xemacps e000b000.ps7-ethernet: MAC updated 8a:89:ac:13:3c:08
xemacps e000b000.ps7-ethernet: pdev->id -1, baseaddr 0xe000b000, irq 54
ehci_hcd: USB 2.0 'Enhanced' Host Controller (EHCI) Driver
ehci-pci: EHCI PCI platform driver
zynq-dr e0002000.ps7-usb: Unable to init USB phy, missing?
usbcore: registered new interface driver usb-storage
mousedev: PS/2 mouse device common for all mice
i2c /dev entries driver
Xilinx Zynq CpuIdle Driver started
sdhci: Secure Digital Host Controller Interface driver
sdhci: Copyright(c) Pierre Ossman
sdhci-pltfm: SDHCI platform and OF driver helper
sdhci-arasan e0100000.ps7-sdio: No vmmc regulator found
sdhci-arasan e0100000.ps7-sdio: No vqmmc regulator found
mmc0: SDHCI controller on e0100000.ps7-sdio [e0100000.ps7-sdio] using ADMA
ledtrig-cpu: registered to indicate activity on CPUs
usbcore: registered new interface driver usbhid
usbhid: USB HID core driver
TCP: cubic registered
NET: Registered protocol family 17
can: controller area network core (rev 20120528 abi 9)
NET: Registered protocol family 29
can: raw protocol (rev 20120528)
can: broadcast manager protocol (rev 20120528 t)
can: netlink gateway (rev 20130117) max_hops=1
zynq_pm_ioremap: no compatible node found for 'xlnx,zynq-ddrc-a05'
zynq_pm_late_init: Unable to map DDRC IO memory.
Registering SWP/SWPB emulation handler
drivers/rtc/hctosys.c: unable to open rtc device (rtc0)
ALSA device list:
  No soundcards found.
RAMDISK: gzip image found at block 0
mmc0: new high speed SDHC card at address aaaa
mmcblk0: mmc0:aaaa SS08G 7.40 GiB
 mmcblk0: p1
EXT2-fs (ram0): warning: mounting unchecked fs, running e2fsck is recommended
VFS: Mounted root (ext2 filesystem) on device 1:0.
devtmpfs: mounted
Freeing unused kernel memory: 212K (40627000 - 4065c000)
Starting rcS...
++ Mounting filesystem
++ Setting up mdev
++ Starting telnet daemon
++ Starting http daemon
++ Starting ftp daemon
++ Starting dropbear (ssh) daemon
random: dropbear urandom read with 1 bits of entropy available
rcS Complete
zynq> 
```

Conclusion

Lab 4 described the steps to compile and boot the Linux kernel on the Zybo-Z7-10 board. The lab introduced the steps in compiling Linux for an embedded system scenario.

Questions

1. The function of this small local memory is to fetch and store the BOOT files during the FPGAs initialisation. This memory is used to initialise the system based on the mode selected using jumper 5. This is called the BootROM and is used to fetch files from the SD card onto memory for execution. Yes, modern motherboards do have such a local memory that holds the BIOS. The BIOS is highly specific to the motherboard hardware and like the case in this lab is for initialising the system.
2. The following directories can be written to
   a. Bin
   b. Dev
   c. Etc
   d. Lib
   e. Licenses
   f. Lost+found
   g. Mnt
   h. Opt
   i. Bin
   j. Sbin
   k. Tmp
   l. Var

   When the system is reset all the created files are lost. This is because there is no storage device mounted to the kernel. Though the SD card is used to store the kernel image and the other required boot files, the kernel doesn't have an interface / port connected to it to write changes back to the SD card. The system only has a volatile DDR3 RAM and no way of writing to storage. By using the mount command the kernel would be able to access and save files on to the SD card.
3. We would have to add details about the peripheral and its interface to the dts file and create a new dtb file. This is required because the DTB contains the address of the new peripheral device. We would also have to create a new boot.bin file with the updated bitstream (hardware wrapper generated by vivado).