ECEN 749- Microprocessor System Design

Section 603

TA: Mr. Kunal Bharathi

LAB 4

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

Date of Performance: 09/26/2019

Student Name: Dhiraj Dinesh Kudva

UIN: 829009538

Introduction

The lab introduces the concept of loading the Linux operating system on the Zybo Z7-10 using a SD card. This will also focus on how a Linux Kernel is created and also the first stage and universal bootloader. This lab provides details about the files necessary for booting the operating system and exactly where it must be loaded on the chip.

Design:

The Zynq chip is divided into two parts: PL(Programming logic) and PS (Processing system) . The Programming system also has an SD card, timer and a DDR3 (Double Data Rate v3 Synchronous Dynamic Random Access Memory) controller. In addition to this, the PS has a Memory Management Unit, which is required to run the Linux. The below figure highlights the same:

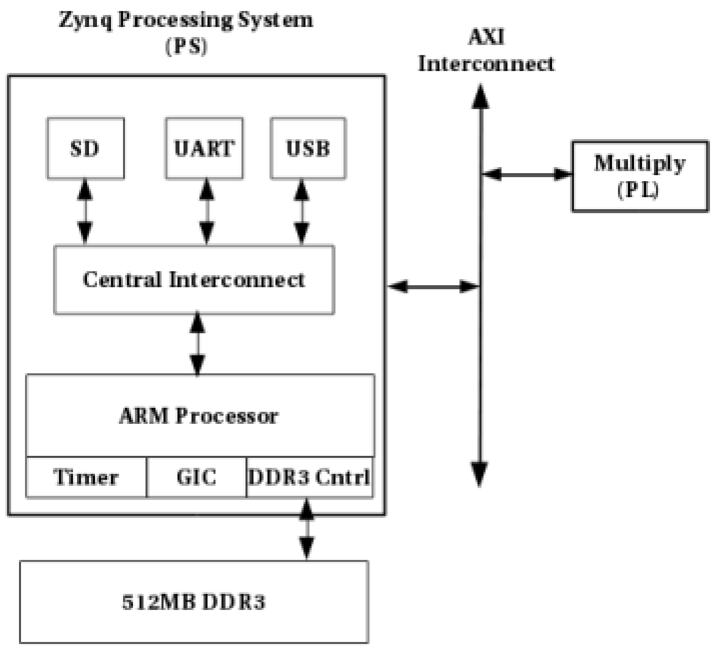
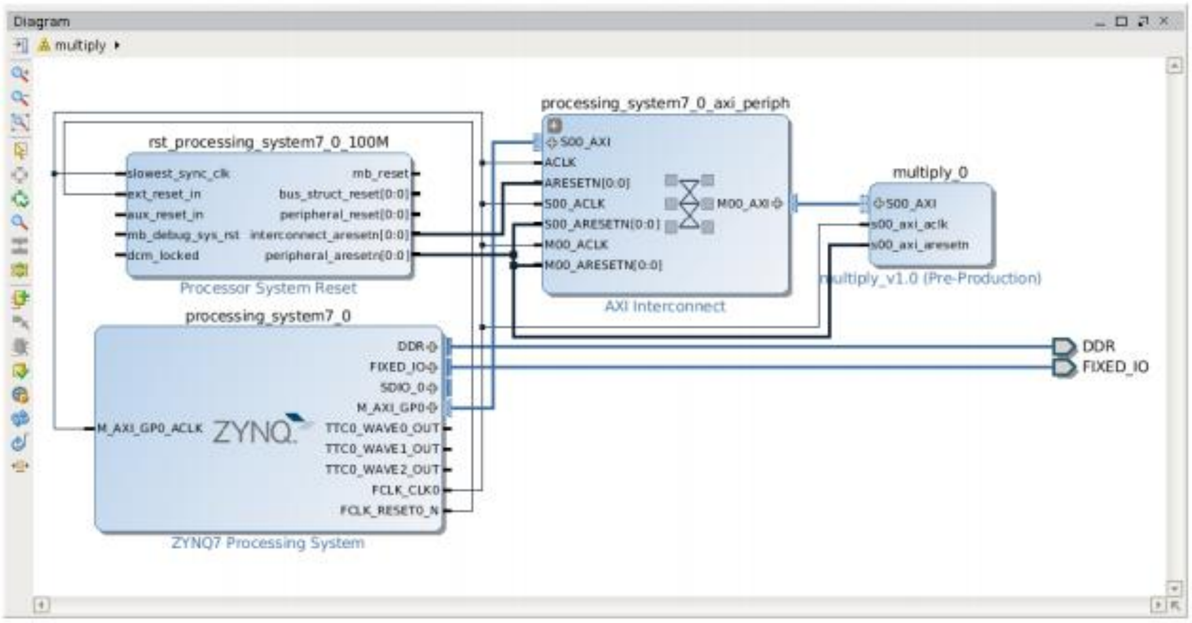


Figure 1: Zynq System Diagram

Procedure: Part 1. Creating Base System Design

1. To create base PS system, create a new directory, open Vivado and create new project.
2. Select the ZYBP Z7-10 from the board in part window.
3. Now add the ZYNQ Processing system (PS) IP in the block design.
4. Next step is to import the .tcl file and run block automation.
5. Then in Peripheral I/O pins only enable SD 0, UART 1 and TTC 0 peripherals by clicking the tick mark on the corresponding peripheral and disable the remaining peripherals.
6. Copy the ip_repo folder which contains multiply form lab3 to this lab folder.

7. Add the ip_repo in the IP setting in the Project setting window and then the multiply_ip would be available to be added to the base system. Add this multiply peripheral to the base system.
8. Run connection automation, then regenerate layout. It should look like the figure below.



9. Then create HDL Wrapper and 'Generate Bitstream'.

Part 2: u-Boot and creating BOOT.bin file

1. The first step is to extract the u-boot file in this project directory by using the following commands.
   Untar the '/mnt/lab_files/ECEN449/u-boot.tar.gz' file by executing the following command from
   within your lab directory
   >tar -xvzf /mnt/lab_files/ECEN449/u-boot.tar.gz
2. Cross compile tools are needed to compile the u-boot. As this is universal boot loader, it must be configured for the ZYBO-Z710 board. First we source the Vivado 2015.2 settings and then execute the following instructions:
   > make CROSS_COMPILE=arm-xilinx-linux-gnueabi- zynq_zybo_config
3. This will show the message 'Configuring for zynq_zybo board...' on the terminal.
4. Now the run command is given to the configured u-boot.
   > make CROSS_COMPILE=arm-xilinx-linux-gnueabi.
5. After the compilation, u-boot file is generated. Add the extension .elf by renaming the file.
6. The generated bitstream is now exported to SDK.
7. Create New Application Project in SDK and select 'Zynq FSBL' in the templates.
8. Then click on 'Build All' to build the projects.
9. Then click on 'Xilinx Tools' and create 'Zynq Boot Image'.
10. Add the path to this lab directory for output.bif.

11. 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'.
12. FSBL is the first stage bootloader and it should be the first file followed by bitstream and u-boot file. This particular order is necessary for creating the BOOT.bin file.
13. Add the bitstream and u-boot file and select partition type as datafile for both.
14. Then click Create image and the BOOT.bin file is created.

Part 3: Linux Kernel

1. Untar the file Linux kernel source code file 'linux-3.14.tar.gz' from ECEN449 shared folder (mnt/lab_files/ECEN449) into this lab 4 directory.
2. Navigate to linux source code folder 'linux-3.14' under the lab 4 directory and type this command

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

3. Then to configure linux,
   ```
   > make ARCH=arm CROSS_COMPILE=arm-xilinx-linux-gnueabi-
   ```
4. The kernel image is created by the above command. However we need an unzipped image instead of a zimage. To convert image, include u-boot tools in your PATH directory.
   ```
   > PATH=$PATH:<directory_to_u_boot>/tools
   ```
   And to make the uImage write the following command
5. The uImage is then created. It is necessary for rebooting the device.
6. The device tree file is also one of the files necessary for bootloading the Linux. The default device tree is located in the Linux Kernel at arch/arm/boot/dts/zynq-zybo.dts.
7. Since the multiply IP is added, we need to create an entry describing the 'multiply' IP in .dts file.
8. '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 334 (in this file: arch/arm/boot/dts/zynq-zybo.dts).

   ```
   multiply {
   compatible = "ecen449,multiply";
   reg = <0x43C00000 0x10000>;
   };
   ```
9. The reg entry holds the address of the multiply module which is the address allocated by the Vivado in the 'Address Editor' tab to the multiply ip.
10. The .dts file created needs to be converted into device tree binary (.dtb) format, as it is compatible for arm.
    This conversion is done using the following instruction.
    ```
    > ./scripts/dtc/dtc -I dts -O dtb -o ./devicetree.dtb arch/arm/boot/dts/zynq-zybo.dts
    ```

11. Copy the ramdisk image from /mnt/lab_files/ECEN449/ramdisk8M.image.gz to the local directory. Then to wrap the header file we make use of the mkimage tools in u-boot directory and run the following command:
    > ./u-boot/tools/mkimage -A arm -T ramdisk -c gzip -d ./ramdisk8M.image.gz uramdisk.image.gz
12. This will create the uramdisk.image.gz file.
13. Along with this, copy BOOT.bin, devicetree.dtb and uImage in the SD card. In this lab we will be booting the Linux on Zybo Z7-10 using SD card.

Part 4: Boot Linux Zybo Z7-10

1. The picocom terminal will be used to show the booting operation of Linux. The Vivado settings must be sourced before running the picocom terminal.
2. Change the mode from JTAG to JP5 on the ZYBO board. This is done to boot the Linux. Connect the power and USB cable.
   Run the following command to start picocom:
   $ picocom -b 115200 -r -l /dev/ttyUSB1
3. Now insert the SD card which has the four files necessary for loading the Linux on the ZYBO board and then press the reset button.
   If the earlier configurations are correct, then the linux will start booting as shown below.

Output:

```
zynq>

U-Boot 2014.01 (Sep 26 2019 - 14:43:28)

I2C:   ready
Memory: ECC disabled
DRAM:  512 MiB
MMC:   zynq_sdhci: 0
spi_setup_slave: No QSPI device detected based on MIO settings
SF: Failed to set up slave
*** Warning - spi_flash_probe() failed, using default environment

In:    serial
Out:   serial
Err:   serial
Net:   Gem.e000b000
Hit any key to stop autoboot:  0
Device: zynq_sdhci
Manufacturer ID: 3
OEM: 5344
Name: SS08G
Tran Speed: 50000000
Rd Block Len: 512
SD version 3.0
High Capacity: Yes
Capacity: 7.4 GiB
Bus Width: 4-bit
reading uEnv.txt
** Unable to read file uEnv.txt **
Copying Linux from SD to RAM...
reading uImage
3447904 bytes read in 306 ms (10.7 MiB/s)
reading devicetree.dtb
7490 bytes read in 15 ms (487.3 KiB/s)
reading uramdisk.image.gz
3693174 bytes read in 323 ms (10.9 MiB/s)
## Booting kernel from Legacy Image at 03000000 ...
   Image Name:   Linux-3.18.0-xilinx
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    3447840 Bytes = 3.3 MiB
   Load Address: 00008000
   Entry Point:  00008000
   Verifying Checksum ... OK
## Loading init Ramdisk from Legacy Image at 02000000 ...
   Image Name:
   Image Type:   ARM Linux RAMDisk Image (gzip compressed)
   Data Size:    3693110 Bytes = 3.5 MiB
   Load Address: 00000000
   Entry Point:  00000000
   Verifying Checksum ... OK
## Flattened Device Tree blob at 02a00000
   Booting using the fdt blob at 0x2a00000
   Loading Kernel Image ... OK
   Loading Ramdisk to 1f7aa000, end 1fb2fa36 ... OK
   Loading Device Tree to 1f7a5000, end 1f7a9d41 ... OK

Starting kernel ...

Booting Linux on physical CPU 0x0
Linux version 3.18.0-xilinx (dhirajkudva@zach-333-em3-22.engr.tamu.edu) (gcc version 4.9.1 (Sourcery CodeBench Lite 2014.11-30) ) #3 SMP PREEMPT Thu Sep 26 15:56:33 CDT 2019
CPU: ARMv7 Processor [413fc090] revision 0 (ARMv7), cr=18c5387d
CPU: PIPT / VIPT nonaliasing data cache, VIPT aliasing instruction cache
Machine model: Zynq ZYBO Z7 Development Board
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)
```

```
Starting kernel ...

Booting Linux on physical CPU 0x0
Linux version 3.18.0-xilinx (dhirajkudva@zach-333-em3-22.engr.tamu.edu) (gcc version 4.9.1 (Sourcery CodeBench Lite 2014.11-30) ) #3 SMP PREEMPT Thu Sep 26 15:56:33 CDT 2019
CPU: ARMv7 Processor [413fc090] revision 0 (ARMv7), cr=18c5387d
CPU: PIPT / VIPT nonaliasing data cache, VIPT aliasing instruction cache
Machine model: Zynq ZYBO Z7 Development Board
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)
Virtual kernel memory layout:
    vector  : 0xffff0000 - 0xffff1000   (   4 kB)
    fixmap  : 0xffc00000 - 0xffe00000   (2048 kB)
    vmalloc : 0x60800000 - 0xff000000   (2536 MB)
    lowmem  : 0x40000000 - 0x60000000   ( 512 MB)
    pkmap   : 0x3fe00000 - 0x40000000   (   2 MB)
    modules : 0x3f000000 - 0x3fe00000   (  14 MB)
      .text : 0x40008000 - 0x40626b1c   (6267 kB)
      .init : 0x40627000 - 0x4065c000   ( 212 kB)
      .data : 0x4065c000 - 0x4069cb60   ( 259 kB)
       .bss : 0x4069cb60 - 0x406d3a78   ( 220 kB)
Preemptible hierarchical RCU implementation.
        Dump stacks of tasks blocking RCU-preempt GP.
        RCU restricting CPUs from NR_CPUS=4 to nr_cpu_ids=2.
RCU: Adjusting geometry for rcu_fanout_leaf=16, nr_cpu_ids=2
NR_IRQS:16 nr_irqs:16 16
L2C-310 erratum 769419 enabled
L2C-310 enabling early BRESP for Cortex-A9
L2C-310 full line of zeros enabled for Cortex-A9
L2C-310 ID prefetch enabled, offset 1 lines
L2C-310 dynamic clock gating enabled, standby mode enabled
L2C-310 cache controller enabled, 8 ways, 512 kB
L2C-310: CACHE_ID 0x410000c8, AUX_CTRL 0x76360001
ps7-slcr mapped to 60804000
zynq_clock_init: clkc starts at 60804100
Zynq clock init
sched_clock: 64 bits at 333MHz, resolution 3ns, wraps every 3298534883328ns
ps7-ttc #0 at 60806000, irq=43
Console: colour dummy device 80x30
Calibrating delay loop... 1332.01 BogoMIPS (lpj=6660096)
pid_max: default: 32768 minimum: 301
Mount-cache hash table entries: 1024 (order: 0, 4096 bytes)
Mountpoint-cache hash table entries: 1024 (order: 0, 4096 bytes)
CPU: Testing write buffer coherency: ok
CPU0: thread -1, cpu 0, socket 0, mpidr 80000000
Setting up static identity map for 0x467598 - 0x4675f0
CPU1: Booted secondary processor
CPU1: thread -1, cpu 1, socket 0, mpidr 80000001
Brought up 2 CPUs
SMP: Total of 2 processors activated.
CPU: All CPU(s) started in SVC mode.
devtmpfs: initialized
VFP support v0.3: implementor 41 architecture 3 part 30 variant 9 rev 4
regulator-dummy: no parameters
NET: Registered protocol family 16
DMA: preallocated 256 KiB pool for atomic coherent allocations
cpuidle: using governor ladder
cpuidle: using governor menu
hw-breakpoint: found 5 (+1 reserved) breakpoint and 1 watchpoint registers.
hw-breakpoint: maximum watchpoint size is 4 bytes.
zynq-ocm f800c000.ps7-ocmc: ZYNQ OCM pool: 256 KiB @ 0x60880000
vgaarb: loaded
SCSI subsystem initialized
```

Result:

The concept of kernel and bootloading operations were explained with the help of this lab. The major task in this lab was creating the four files: BOOT.bin, devicetree.dtb, uImage and uramdisk.iamge.gz. The configuration for BOOT.bin file was one of the crucial task, as the FSBL and the corresponding datafile must be in that particular order. Any alteration in the same would have resulted in generation of a wrong BOOT.bin.

Conclusion:

The lab helped in understanding the basics of how an operating system is loaded and what all files are needed for the same. It also helped in understanding the Linux Kernel and how to upload the operating system on ZYBO Z7-10 board.

Questions:

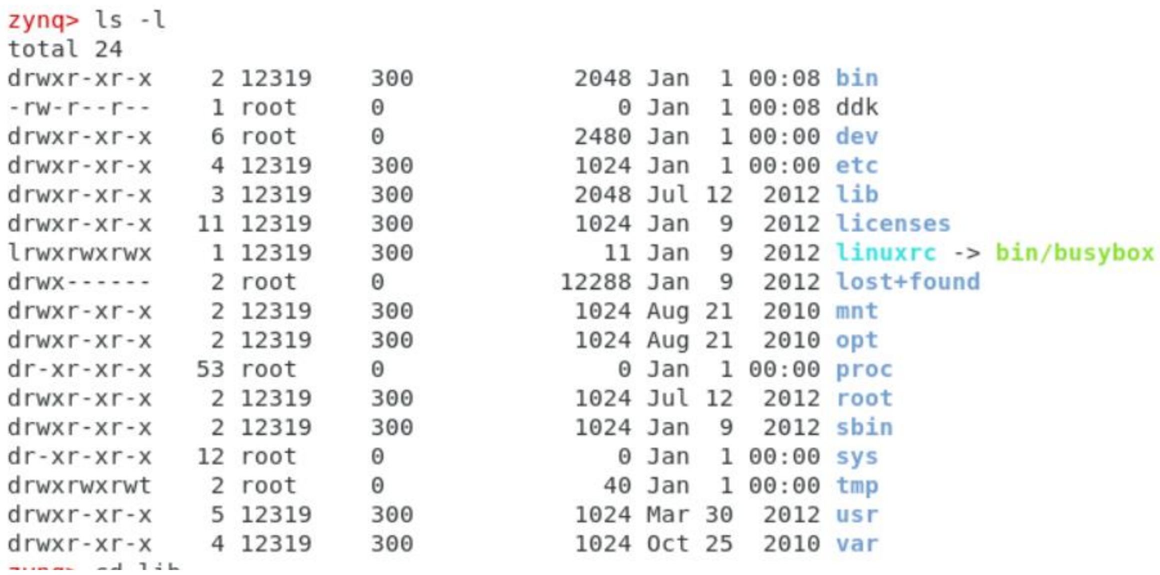
a. Compared to lab 3, the lab 4 microprocessor system shown in Figure 1 has 512 MB of SDRAM. However, our system still includes a small amount of local memory. What is the function of the local memory? Does this 'local memory" exist on a standard motherboard? If so, where?

Ans. Considering that the local memory is the on chip memory. The on chip memory (OCM) module on the Zynq Z7-10 contains 256 kB of RAM and 128 kB of ROM (BootROM). The BootROM memory is used exclusively by the boot process and is not visible to the user. Modern motherboard also contain non-volatile memory to initialize the system and load operating system from some external peripheral device. Some motherboard design use a BIOS, which is stored in EEPROM chip. These local memories are generally chip soldered or socketed on the motherboard.

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?

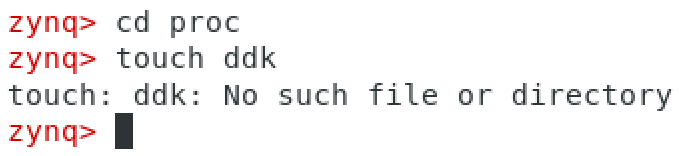
Ans. The directories such as bin, dev, etc, lib, licenses, lost+found, mnt, opt, root, sbin, tmp, usr and var are writable directories. This can be seen from the terminal console image.

Only proc and sys do not have write permission in them and hence are called as non-writable directories.



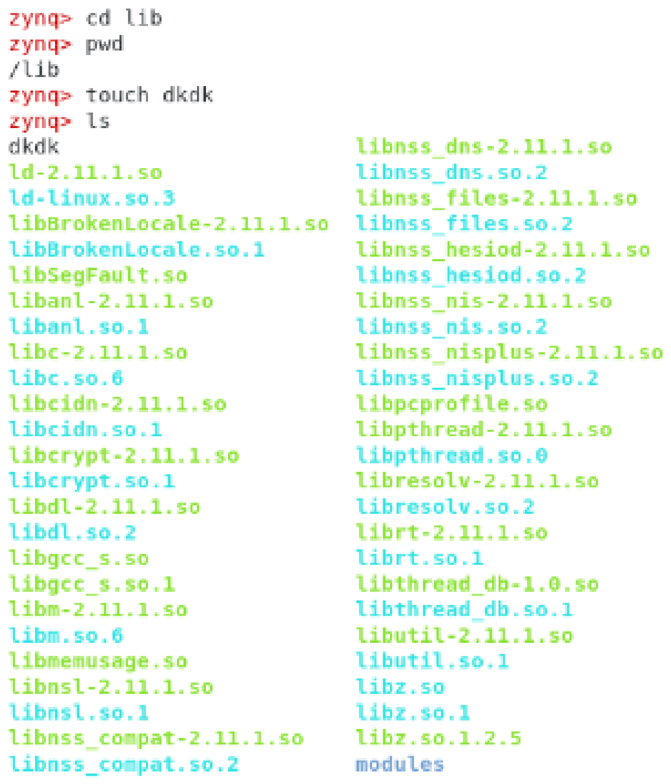
```
zynq> ls -l
total 24
drwxr-xr-x    2 12319    300           2048 Jan  1 00:08 bin
-rw-r--r--    1 root     0                0 Jan  1 00:08 ddk
drwxr-xr-x    6 root     0             2480 Jan  1 00:00 dev
drwxr-xr-x    4 12319    300           1024 Jan  1 00:00 etc
drwxr-xr-x    3 12319    300           2048 Jul 12  2012 lib
drwxr-xr-x   11 12319    300           1024 Jan  9  2012 licenses
lrwxrwxrwx    1 12319    300             11 Jan  9  2012 linuxrc -> bin/busybox
drwx------    2 root     0            12288 Jan  9  2012 lost+found
drwxr-xr-x    2 12319    300           1024 Aug 21  2010 mnt
drwxr-xr-x    2 12319    300           1024 Aug 21  2010 opt
dr-xr-xr-x   53 root     0                0 Jan  1 00:00 proc
drwxr-xr-x    2 12319    300           1024 Jul 12  2012 root
drwxr-xr-x    2 12319    300           1024 Jan  9  2012 sbin
dr-xr-xr-x   12 root     0                0 Jan  1 00:00 sys
drwxrwxrwt    2 root     0               40 Jan  1 00:00 tmp
drwxr-xr-x    5 12319    300           1024 Mar 30  2012 usr
drwxr-xr-x    4 12319    300           1024 Oct 25  2010 var
```

If we try write in proc or sys directory using touch command, then the file is not created.



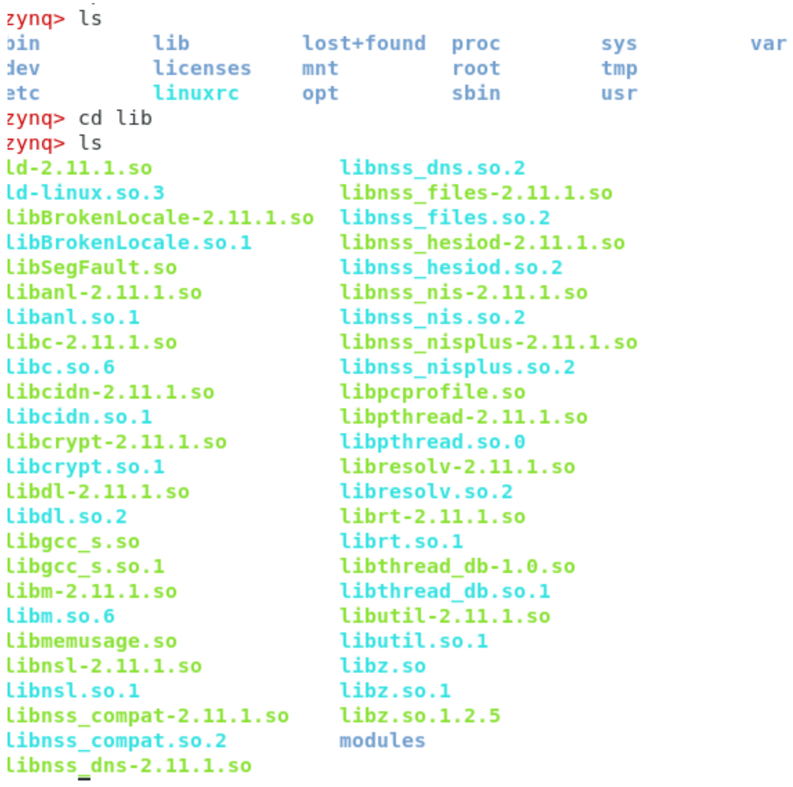
```
zynq> cd proc
zynq> touch ddk
touch: ddk: No such file or directory
zynq> 
```

However, if we use the same touch command in writable directories, then the file is created.



```
zynq> cd lib
zynq> pwd
/lib
zynq> touch dkdk
zynq> ls
dkdk                           libnss_dns-2.11.1.so
ld-2.11.1.so                   libnss_dns.so.2
ld-linux.so.3                  libnss_files-2.11.1.so
libBrokenLocale-2.11.1.so      libnss_files.so.2
libBrokenLocale.so.1           libnss_hesiod-2.11.1.so
libSegFault.so                 libnss_hesiod.so.2
libanl-2.11.1.so               libnss_nis-2.11.1.so
libanl.so.1                    libnss_nis.so.2
libc-2.11.1.so                 libnss_nisplus-2.11.1.so
libc.so.6                      libnss_nisplus.so.2
libcidn-2.11.1.so              libpcprofile.so
libcidn.so.1                   libpthread-2.11.1.so
libcrypt-2.11.1.so             libpthread.so.0
libcrypt.so.1                  libresolv-2.11.1.so
libdl-2.11.1.so                libresolv.so.2
libdl.so.2                     librt-2.11.1.so
libgcc_s.so                    librt.so.1
libgcc_s.so.1                  libthread_db-1.0.so
libm-2.11.1.so                 libthread_db.so.1
libm.so.6                      libutil-2.11.1.so
libmemusage.so                 libutil.so.1
libnsl-2.11.1.so               libz.so
libnsl.so.1                    libz.so.1
libnss_compat-2.11.1.so        libz.so.1.2.5
libnss_compat.so.2             modules
```

When we restart the ZYBO Z7-10 board, the created file is deleted. This is because the Linux kernel is working on DDR3, which is a volatile memory. The files created when the linux is online are deleted whenever the system is switched off, in this case when the ZYBO Z7-10 board is switched off.



```
zynq> ls
bin         lib         lost+found  proc        sys         var
dev         licenses    mnt         root        tmp
etc         linuxrc     opt         sbin        usr
zynq> cd lib
zynq> ls
ld-2.11.1.so              libnss_dns.so.2
ld-linux.so.3             libnss_files-2.11.1.so
libBrokenLocale-2.11.1.so libnss_files.so.2
libBrokenLocale.so.1      libnss_hesiod-2.11.1.so
libSegFault.so            libnss_hesiod.so.2
libanl-2.11.1.so          libnss_nis-2.11.1.so
libanl.so.1               libnss_nis.so.2
libc-2.11.1.so            libnss_nisplus-2.11.1.so
libc.so.6                 libnss_nisplus.so.2
libcidn-2.11.1.so         libpcprofile.so
libcidn.so.1              libpthread-2.11.1.so
libcrypt-2.11.1.so        libpthread.so.0
libcrypt.so.1             libresolv-2.11.1.so
libdl-2.11.1.so           libresolv.so.2
libdl.so.2                librt-2.11.1.so
libgcc_s.so               librt.so.1
libgcc_s.so.1             libthread_db-1.0.so
libm-2.11.1.so            libthread_db.so.1
libm.so.6                 libutil-2.11.1.so
libmemusage.so            libutil.so.1
libnsl-2.11.1.so          libz.so
libnsl.so.1               libz.so.1
libnss_compat-2.11.1.so   libz.so.1.2.5
libnss_compat.so.2        modules
libnss_dns-2.11.1.so
```

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?

Ans. To add another peripheral in the programming system after compiling the kernel, we need to modify the devicetree.dtb file. A device tree is a data structure describing the hardware components of a particular computer so that the operating system's kernel can use and manage those components, including the CPU or CPUs, the memory, the buses and the peripherals. We need to follow the steps from Part 3 step 8. If we need to add an another peripheral in the programmable logic similar to the multiply IP, then we need to create a new BOOT.bin file.