Lab 2: Build and Boot an Image

Introduction

The most basic skill required for developing embedded Linux is working in the cross-compilation environment: compiling the kernel, libraries, and applications and downloading the resulting image onto the embedded target. The purpose of this lab is to familiarize you with this process.

This lab will prepare you for the most basic task of working with embedded Linux: how to build and boot the operating system and applications. Embedded Linux target processors, such as the ARM® Cortex™-A9 MPcore, are usually developed in a cross-compilation environment. This means that the kernel and applications are compiled on a development machine (in this case, a Linux PC having a non-target processor), and then downloaded onto the target.

The PetaLinux tools support a number of configuration architectures that automate much of this process. In this lab, you will learn how to use these tools and how to download the resulting embedded Linux image onto the hardware platform.

QEMU is a generic and open-source machine emulator integrated into the PetaLinux tools. In this lab, you will use QEMU to run the Linux built for the ARM Cortex-A9 MPcore system. It can achieve near native performance by executing the guest code directly on the host CPU

Objectives

After completing this lab, you will be able to:
- Build the ARM Cortex-A9 MPcore Linux kernel and applications
- Boot the resulting system image in QEMU
- Download the resulting system image onto the development board

Preparation

If this is the first lab that you are performing, then refer to the “Before You Start” section of Lab 1 for necessary preparatory information on how to set up the environment.

If your workstation has been restarted or logout, run the following command to start DHCP server on the host:

```
[host] $ sudo service isc-dhcp-server restart
```

Please refer to the “Initializing the Workshop Environment” section of Lab 1 for detailed information.

General Flow for this Lab

Choosing a Linux Platform Step 1

A Linux platform tells what to build into the Linux image; it tells the following information:

- The hardware platform information such as address mapping, interrupts, and the processor's characteristics, for example
- The Linux kernel settings
- User space applications settings
- File system settings
- Flash partition table settings

1-1. Change the path to the project directory.

1-1-1. Run the following commands to create and change to the project directory path:

```
[host] $ mkdir ~/emblnx/labs/lab2

[host] $ cd ~/emblnx/labs/lab2
```

Each lab in this workshop is installed in the `~/emblnx/labs` directory. Adjust the path if you have installed the labs at a different path.

1-2. Use the `petalinux-create` command to create a new embedded Linux platform and choose the platform.

1-2-1. Run the following command from the `lab2` directory to create a new Petalinux project:

```
[host] $ petalinux-create -t project -s /opt/pkg/ZYBO_petalinux_v2014_2.bsp
```

```
petalinux@ubuntu:~/emblnx/labs/lab2$ petalinux-create -t project -s /opt/pkg/ZYBO_petalinux_v2014_2.bsp
INFO: Create project:
INFO: Projects:
INFO:   * ZYBO_petalinux_v2014_2
INFO: has been successfully installed to /home/petalinux/emblnx/labs/lab2/
INFO: New project successfully created in /home/petalinux/emblnx/labs/lab2/
petalinux@ubuntu:~/emblnx/labs/lab2$
```

Figure 1. Creating a new PetaLinux project

The above command assumes that the board support package (BSP) is installed in the `/opt/pkg` directory. Modify the path if the BSP is in a different location.

The command will create the PetaLinux software project directory: `ZYBO_petalinux_v2014_2` under `~/emblnx/labs/lab2`.

A PetaLinux project directory contains configuration files of the project, the Linux subsystem, and the components of the subsystem. `petalinux-build` builds the project with those configuration files. User can run `petalinux-config` to modify them. Below is the PetaLinux project directory.

```
<project-root>
    |-.petalinux/
    |-hw-description/
    |-config.project
    |-subsystems/
    |    |-linux/
    |    |    |-config
    |    |    |-hw-description/
    |    |    |-configs/
    |    |    |    |-device-tree/
    |    |    |    |    |-ps.dtsi
    |    |    |    |    |-pl.dtsi
    |    |    |    |    |-system-conf.dtsi
    |    |    |    |    |-system-top.dts
    |    |    |    |-kernel/
    |    |    |    |    |-config
    |    |    |    |-u-boot/
    |    |    |    |    |-config.mk
    |    |    |    |    |-platform-auto.h
    |    |    |    |    |-platform-top.h
    |    |    |    |-rootfs/
    |    |    |         |-config
    |-components/
    |    |-bootloader/
    |    |    |-fs-boot/ | zynq_fsbl/
    |    |-apps/
    |    |    |-myapp/
```

Figure 1: PetaLinux Project Directory

1-2-2. Change the directory to `~/emblnx/labs/lab2/ZYBO_petalinux_v2014_2`.

Building the Linux Image — Step 1

2-1. Now that you have selected a pre-built platform, build a Linux image based on this platform.

2-1-1. Enter the following command to build the Linux image:

```
$ petalinux-build
```

```
petalinux@ubuntu:~/emblnx/labs/lab2/ZYBO_petalinux_v2014_2$ petalinux-build
INFO: Checking component...
INFO: Generating make files and build linux
INFO: Generating make files for the subcomponents of linux
INFO: Building linux
[INFO ] pre-build linux/rootfs/fwupgrade
[INFO ] pre-build linux/rootfs/peekpoke
[INFO ] pre-build linux/rootfs/uWeb
[INFO ] build system.dtb
[INFO ] build linux/kernel
[INFO ] update linux/u-boot source
[INFO ] generate linux/u-boot configuration files
[INFO ] build linux/u-boot
[INFO ] Setting up stage config
[INFO ] Setting up rootfs config
[INFO ] Updating for armv7a-vfp-neon
[INFO ] Updating package manager
[INFO ] Expanding stagefs
[INFO ] build linux/rootfs/fwupgrade
[INFO ] build linux/rootfs/peekpoke
[INFO ] build linux/rootfs/uWeb
[INFO ] build kernel in-tree modules
[INFO ] modules linux/kernel
[INFO ] post-build linux/rootfs/fwupgrade
[INFO ] post-build linux/rootfs/peekpoke
[INFO ] post-build linux/rootfs/uWeb
[INFO ] pre-install linux/rootfs/fwupgrade
[INFO ] pre-install linux/rootfs/peekpoke
[INFO ] pre-install linux/rootfs/uWeb
[INFO ] install system.dtb
[INFO ] install linux/kernel
[INFO ] update linux/u-boot source
[INFO ] generate linux/u-boot configuration files
[INFO ] build linux/u-boot
[INFO ] install linux/u-boot
[INFO ] Expanding rootfs
[INFO ] install sys_init
[INFO ] install linux/rootfs/fwupgrade
[INFO ] install linux/rootfs/peekpoke
[INFO ] install linux/rootfs/uWeb
[INFO ] install kernel in-tree modules
[INFO ] modules_install linux/kernel
[INFO ] post-install linux/rootfs/fwupgrade
[INFO ] post-install linux/rootfs/peekpoke
[INFO ] post-install linux/rootfs/uWeb
[INFO ] package rootfs.cpio to /home/petalinux/emblnx/labs/lab2/ZYBO_petalinux_v2014_2/images/linux
[INFO ] Update and install vmlinux image
[INFO ] vmlinux linux/kernel
[INFO ] install linux/kernel
[INFO ] package zImage
[INFO ] zImage linux/kernel
[INFO ] install linux/kernel
```

Figure 3. Building the Linux image

This may take a few minutes. During this time, the following will occur:

- Cross-compiling and linking of the Linux kernel (`linux-3.x/*`)
- Cross-compiling and linking of the default user libs and applications (`lib/*` and `user/*`)
- Building of a local copy of the ARM Cortex-A9 processor Linux root file system (`romfs/*`)
- Assembling of the kernel and root file system into a single downloadable binary image file (`images/*`)
- Copying of the image files from `images/` in to `/tftpboot`

The build log is saved in the `~/emblnx/labs/lab2/ZYBO_petalinux_v2014_2/build.log` file.

2-1-2. Once compilation completes, look at the contents in the `images/linux` subdirectory by executing the following commands from the project directory:

```
[host] $ cd images/linux
```

```
[host] $ ls -la
```

```
petalinux@ubuntu:~/emblnx/labs/lab2/ZYBO_petalinux_v2014_2/images/linux$ ls -la
total 59704
drwxrwxr-x 2 petalinux petalinux     4096 Oct  7 20:33 .
drwxrwxr-x 3 petalinux petalinux     4096 Oct  7 20:19 ..
-rwxrwxr-x 1 petalinux petalinux 10887796 Oct  7 20:32 image.elf
-rw-rw-r-- 1 petalinux petalinux  7108252 Oct  7 20:33 image.ub
-rw-rw-r-- 1 petalinux petalinux  8045568 Oct  7 20:32 rootfs.cpio
-rw-rw-r-- 1 petalinux petalinux  3536083 Oct  7 20:32 rootfs.cpio.gz
-rw-rw-r-- 1 petalinux petalinux    16281 Oct  7 20:30 system.dtb
-rw-rw-r-- 1 petalinux petalinux  1762086 Oct  7 20:33 System.map.linux
-rw-rw-r-- 1 petalinux petalinux   253188 Oct  7 20:30 u-boot.bin
-rwxrwxr-x 1 petalinux petalinux  1436510 Oct  7 20:30 u-boot.elf
-rw-rw-r-- 1 petalinux petalinux   253188 Oct  7 20:33 u-boot-s.bin
-rwxrwxr-x 1 petalinux petalinux  1436510 Oct  7 20:33 u-boot-s.elf
-rw-rw-r-- 1 petalinux petalinux   759716 Oct  7 20:30 u-boot.srec
-rw-rw-r-- 1 petalinux petalinux   759696 Oct  7 20:33 u-boot-s.srec
-rw-rw-r-- 1 petalinux petalinux  3536147 Oct  7 20:32 urootfs.cpio.gz
-rwxrwxr-x 1 petalinux petalinux 14063189 Oct  7 20:33 vmlinux
-rwxrwxr-x 1 petalinux petalinux  7107216 Oct  7 20:33 zImage
-rwxrwxr-x 1 petalinux petalinux   297253 Oct  7 20:29 zynq_fsbl.elf
```

Figure 4. Various generated files

2-1-3. Examine the contents of the `/tftpboot` directory by executing:

```
[host] $ ls /tftpboot
```

All these files in the `~/emblnx/labs/lab2/ZYBO_petalinux_v2014_2/images/linux` directory have a copy in `/tftpboot` because as part of the build process, the image files have also been copied there. The development machine has been configured as a TFTP (trivial FTP) server, allowing the board to pull new kernel images directly over the network from the fixed known location (instead of knowing the actual paths of the project directories). You will use this capability in the next exercise.

Image Name Linux Kernel and System Images	Descriptions
`image.elf`	Linux image in ELF format
`image.srec`	Linux image in SREC format
`image.ub`	Linux image in U-Boot format
`rootfs.cpio`	Root file system image
`u-boot.bin`	U-Boot image in binary format
`u-boot.srec`	U-Boot image in SREC format
`u-boot.elf`	U-Boot image in ELF format

Image Name Linux Kernel and System Images	Descriptions
u-boot-s.*	Relocatable U-Boot image

Booting the System Step 2

3-1. As mentioned earlier, you can run Linux for the ARM Cortex-A9 MPcore system on QEMU.

Load the ARM Cortex-A9 MPcore Linux on QEMU.

3-1-1. Enter the following command in the host Terminal window to load the kernel only:

```
[host]$ petalinux-boot --qemu --kernel
```

```
petalinux@ubuntu:~/emblnx/labs/lab2/ZYBO_petalinux_v2014_2/images/linux$ petalinux-boot --qemu --kernel
INFO: The image provided is a zImage
INFO: TCP PORT is free
INFO: Starting arm QEMU
INFO:  qemu-system-arm -L /opt/pkg/petalinux-v2014.2-final/etc/qemu -M arm-generic-fdt -smp 2 -machine linux=on   --serial mon:s
tdio --nographic -kernel /tmp/tmp.zU8qsI6NqU -gdb tcp::9000 -dtb /home/petalinux/emblnx/labs/lab2/ZYBO_petalinux_v2014_2/images/
linux/system.dtb -tftp /tftpboot
Uncompressing Linux... done, booting the kernel.
Booting Linux on physical CPU 0x0
Linux version 3.14.2-xilinx (petalinux@ubuntu) (gcc version 4.8.1 (Sourcery CodeBench Lite 2013.11-53) ) #2 SMP PREEMPT Tue Oct
7 20:32:19 UTC 2014
CPU: ARMv7 Processor [410fc090] revision 0 (ARMv7), cr=10c5387d
CPU: PIPT / VIPT nonaliasing data cache, VIPT aliasing instruction cache
Machine model: ZYBO_petalinux_v2014_2
bootconsole [earlycon0] enabled
Memory policy: Data cache writealloc
PERCPU: Embedded 8 pages/cpu @dfbdc000 s10752 r8192 d13824 u32768
Built 1 zonelists in Zone order, mobility grouping on.  Total pages: 130048
Kernel command line: console=ttyPS0,115200 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: 503676K/524288K available (4835K kernel code, 310K rwdata, 1772K rodata, 3642K init, 5337K bss, 20612K reserved, 0K high
mem)
Virtual kernel memory layout:
    vector  : 0xffff0000 - 0xffff1000   (   4 kB)
    fixmap  : 0xfff00000 - 0xfffe0000   ( 896 kB)
    vmalloc : 0xe0800000 - 0xff000000   ( 488 MB)
    lowmem  : 0xc0000000 - 0xe0000000   ( 512 MB)
    pkmap   : 0xbfe00000 - 0xc0000000   (   2 MB)
    modules : 0xbf000000 - 0xbfe00000   (  14 MB)
      .text : 0xc0008000 - 0xc067bf7c   (6608 kB)
      .init : 0xc067c000 - 0xc0a0aa00   (3643 kB)
      .data : 0xc0a0c000 - 0xc0a598a8   ( 311 kB)
       .bss : 0xc0a598b4 - 0xc0f8fd88   (5338 kB)
Preemptible hierarchical RCU implementation.
        RCU lockdep checking is enabled.
        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
ps7-slcr mapped to e0802000
zynq_clock_init: clkc starts at e0802100
Zynq clock init
sched_clock: 16 bits at 81kHz, resolution 12288ns, wraps every 805296141ns
ps7-ttc #0 at e0804000, irq=43
Console: colour dummy device 80x30
```

Figure 5. Console output 1

```
Lock dependency validator: Copyright (c) 2006 Red Hat, Inc., Ingo Molnar
... MAX_LOCKDEP_SUBCLASSES:  8
... MAX_LOCK_DEPTH:          48
... MAX_LOCKDEP_KEYS:        8191
... CLASSHASH_SIZE:          4096
... MAX_LOCKDEP_ENTRIES:     16384
... MAX_LOCKDEP_CHAINS:      32768
... CHAINHASH_SIZE:          16384
 memory used by lock dependency info: 3695 kB
 per task-struct memory footprint: 1152 bytes
Calibrating delay loop... 2039.80 BogoMIPS (lpj=10199040)
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
missing device node for CPU 0
missing device node for CPU 1
CPU0: thread -1, cpu 0, socket 0, mpidr 80000000
Setting up static identity map for 0x496b30 - 0x496b88
L2x0 series cache controller enabled
l2x0: 8 ways, CACHE_ID 0x00000000, AUX_CTRL 0x00000000, Cache size: 512 kB
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 0
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
syscon f8000000.ps7-slcr: regmap [mem 0xf8000000-0xf8000fff] registered
hw-breakpoint: debug architecture 0x0 unsupported.
bio: create slab <bio-0> at 0
vgaarb: loaded
SCSI subsystem initialized
usbcore: registered new interface driver usbfs
usbcore: registered new interface driver hub
usbcore: registered new device driver usb
media: Linux media interface: v0.10
Linux video capture interface: v2.00
pps_core: LinuxPPS API ver. 1 registered
pps_core: Software ver. 5.3.6 - Copyright 2005-2007 Rodolfo Giometti <giometti@linux.it>
PTP clock support registered
EDAC MC: Ver: 3.0.0
DMA-API: preallocated 4096 debug entries
DMA-API: debugging enabled by kernel config
Switched to clocksource ttc_clocksource
NET: Registered protocol family 2
TCP established hash table entries: 4096 (order: 2, 16384 bytes)
TCP bind hash table entries: 4096 (order: 5, 147456 bytes)
TCP: Hash tables configured (established 4096 bind 4096)
TCP: reno registered
UDP hash table entries: 256 (order: 2, 20480 bytes)
```

Figure 6. Console output 2

```
UDP-Lite hash table entries: 256 (order: 2, 20480 bytes)
NET: Registered protocol family 1
RPC: Registered named UNIX socket transport module.
RPC: Registered udp transport module.
RPC: Registered tcp transport module.
RPC: Registered tcp NFSv4.1 backchannel transport module.
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 983
io scheduler noop registered
io scheduler deadline registered
io scheduler cfq registered (default)
dma-pl330 f8003000.ps7-dma: Loaded driver for PL330 DMAC-267056
dma-pl330 f8003000.ps7-dma:     DBUFF-128x8bytes Num_Chans-8 Num_Peri-4 Num_Events-16
e0001000.serial: ttyPS0 at MMIO 0xe0001000 (irq = 82, base_baud = 992063) is a xuartps
console [ttyPS0] enabled
console [ttyPS0] enabled
bootconsole [earlycon0] disabled
bootconsole [earlycon0] disabled
brd: module loaded
loop: module loaded
m25p80 spi32766.0: s25fl256s1 (32768 Kbytes)
4 ofpart partitions found on MTD device spi32766.0
Creating 4 MTD partitions on "spi32766.0":
0x000000000000-0x000000500000 : "boot"
0x000000500000-0x000000520000 : "bootenv"
0x000000520000-0x000000fa0000 : "kernel"
0x000000fa0000-0x000002000000 : "spare"
e1000e: Intel(R) PRO/1000 Network Driver - 2.3.2-k
e1000e: Copyright(c) 1999 - 2013 Intel Corporation.
libphy: XEMACPS mii bus: probed
xemacps e000b000.ps7-ethernet: invalid address, use assigned
xemacps e000b000.ps7-ethernet: MAC updated be:d4:87:99:ac:05
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
cpufreq_cpu0: failed to find cpu0 node
cpufreq-cpu0: probe of cpufreq-cpu0.0 failed with error -2
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
usbcore: registered new interface driver usbhid
usbhid: USB HID core driver
TCP: cubic registered
NET: Registered protocol family 17
zynq_pm_ioremap: no compatible node found for 'xlnx,zynq-ddrc-1.0'
zynq_pm_late_init: Unable to map DDRC IO memory.
zynq_pm_remap_ocm: no compatible node found for 'xlnx,zynq-ocmc-1.0'
zynq_pm_late_init: Unable to map OCM.
Registering SWP/SWPB emulation handler
regulator-dummy: disabling
/opt/pkg/petalinux-v2014.2-final/components/linux-kernel/xlnx-3.14/drivers/rtc/hctosys.c: unable to open rtc device (rtc0)
Freeing unused kernel memory: 3640K (c067c000 - c0a0a000)
INIT: version 2.88 booting
Starting Bootlog daemon: bootlogd.
Creating /dev/flash/* device nodes
random: dd urandom read with 3 bits of entropy available
```

```
i2c /dev entries driver
cdns-i2c e0004000.ps7-i2c: 400 kHz mmio e0004000 irq 57
cpufreq_cpu0: failed to find cpu0 node
cpufreq-cpu0: probe of cpufreq-cpu0.0 failed with error -2
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
usbcore: registered new interface driver usbhid
usbhid: USB HID core driver
TCP: cubic registered
NET: Registered protocol family 17
zynq_pm_ioremap: no compatible node found for 'xlnx,zynq-ddrc-1.0'
zynq_pm_late_init: Unable to map DDRC IO memory.
zynq_pm_remap_ocm: no compatible node found for 'xlnx,zynq-ocmc-1.0'
zynq_pm_late_init: Unable to map OCM.
Registering SWP/SWPB emulation handler
regulator-dummy: disabling
/opt/pkg/petalinux-v2014.2-final/components/linux-kernel/xlnx-3.14/drivers/rtc/hctosys.c: unable to open rtc device (rtc0)
Freeing unused kernel memory: 3640K (c067c000 - c0a0a000)
INIT: version 2.88 booting
Starting Bootlog daemon: bootlogd.
Creating /dev/flash/* device nodes
random: dd urandom read with 3 bits of entropy available
starting Busybox inet Daemon: inetd... done.
Starting uWeb server:
NET: Registered protocol family 10
update-rc.d: /etc/init.d/run-postinsts exists during rc.d purge (continuing)
 Removing any system startup links for run-postinsts ...
  /etc/rcS.d/S99run-postinsts
INIT: Entering runlevel: 5
Configuring network interfaces... xemacps e000b000.ps7-ethernet: eth0: no PHY found
xemacps e000b000.ps7-ethernet: XEMACPS mii bus mii_probe fail.
ifconfig: SIOCSIFFLAGS: No such device or address
Stopping Bootlog daemon:
Built with PetaLinux v2014.2 (Yocto 1.6) ZYBO_petalinux_v2014_2 /dev/ttyPS0
ZYBO_petalinux_v2014_2 login:
```

Figure 7. Console output 3

3-1-2. Log into the system and explore it as you did in the "A First Look" lab.
Note: Use `root` as the login name and password.

3-1-3. Exit QEMU by pressing <**Ctrl + a**> then <**x**>.

3-2. Copy the `BOOT.BIN` file from the pre-built directory to the MICRO-SD card.

3-2-1. Copy only the `BOOT.BIN` file from the `~/emblnx/labs/lab2/ZYBO_petalinux_v2014_2/pre-built/linux/images` directory to the MICRO-SD card.

3-2-2. Make sure that the board is turned OFF.

3-2-3. Insert the MICRO-SD card into the target board.

3-2-4. Make sure that the board is set to boot from the MICRO-SD card.

3-3. Run the DHCP server on the host.

3-3-1. Run the DHCP server:

```
[host]$ sudo service isc-dhcp-server restart
```

3-4. Power up the board and set the serial port terminal.

3-4-1. Power ON the board.

3-4-2. Run the following command to make sure that /dev/ttyUSB1 is set to read/write access:

```
[host]$ sudo chmod 666 /dev/ttyUSB1
```

3-4-3. In the dashboard, in the Search field, enter the serial port.

3-4-4. Select the **Serial port terminal** application.

3-5. Boot the new Linux image on the board.

3-5-1. Reset the board (BTN7) to see the booting info on the GtkTerm console as the board goes through the boot process.

3-5-2. Press any key to stop auto-boot when you see messages similar to the following in the GtkTerm window:

```
*****************************************************************
*            ZYBO Unique MAC Addr: 00 04 A3 D1 D5 B2            *
*---------------------------------------------------------------*
* The Unique MAC address printed above can be used by modifying *
* the u-boot environment stored in QSPI flash. This is done with *
* the saveenv command in u-boot. Refer to the Xilinx document    *
* titled "Petalinux Tools User Guide: Firmware Upgrade Guide"    *
* (UG983) for more information.                                  *
*****************************************************************


U-Boot 2014.01 (Jul 01 2014 - 19:56:17)

Memory: ECC disabled
DRAM:  512 MiB
MMC:   zynq_sdhci: 0
SF: Detected S25FL128S_64K with page size 256 Bytes, erase size 64 KiB, total 16 MiB
*** Warning - bad CRC, using default environment

In:    serial
Out:   serial
Err:   serial
Net:   Gem.e000b000
U-BOOT for ZYBO_petalinux_v2014_2

Gem.e000b000 Waiting for PHY auto negotiation to complete........ done
BOOTP broadcast 1
DHCP client bound to address 192.168.1.11
Hit any key to stop autoboot:  0
```

Figure 8. Stopping the autoboot

3-5-3. If you did not see the "DHCP client bound to address" message during uboot bootup, you will need to run `dhcp` to obtain the IP address.

```
U-Boot-PetaLinux> dhcp

Hit any key to stop autoboot:  0
U-Boot-PetaLinux> dhcp
Gem.e000b000:0 is connected to Gem.e000b000.  Reconnecting to Gem.e000b000
Gem.e000b000 Waiting for PHY auto negotiation to complete....... done
BOOTP broadcast 1
DHCP client bound to address 192.168.1.6
```

Figure 9. Running DHCP to obtain the IP address

3-5-4. Set the TFTP server IP to the host IP by running the following command in the u-boot console:

```
U-Boot-PetaLinux> set serverip 192.168.1.1
```

3-5-5. Download and boot the new image using `TFTP` by executing this command in the u-boot console:

```
U-Boot-PetaLinux> run netboot
```

This command will download the `image.ub` file from `/tftpboot` on the host to the main memory of the ARM Cortex-A9 MPcore system and boot the system with the image.

3-5-6. Watch the GtkTerm window.

Messages similar to the following show the image download progress.

```
U-Boot-PetaLinux> run netboot
Gem.e000b000:1 is connected to Gem.e000b000.  Reconnecting to Gem.e000b000
Gem.e000b000 Waiting for PHY auto negotiation to complete........ done
Using Gem.e000b000 device
TFTP from server 192.168.1.1; our IP address is 192.168.1.12
Filename 'image.ub'.
Load address: 0x1000000
Loading: ##################################################################
         ##################################################################
         ##################################################################
/dev/ttyUSB1 115200-8-N-1                                              DTR
```

Figure 10. Downloading the built image

The `netboot` command will automatically boot the system as soon as the image is finished downloading.

3-5-7. Watch the booting messages on the GtkTerm window.

Other booting messages are the same as from the Lab1 because you used the default configuration.

3-6. Use `ping` command to test the network connection.

3-6-1. After the system boots, log into the system by entering `root` as both the login name and password.

3-6-2. Execute the `ping` command to ping the host machine.

```
# ping 192.168.1.1
```

You should see the response from the host machine.

3-6-3. Execute the `ping` command from the host machine terminal window to see the response from the target board.

```
[host]$ ping 192.168.1.11
```

Use different ip address if the board is bound to different address (see Figure 8 to find out the address). You should see the response from the host machine.

3-7. Soft reboot from Linux.

3-7-1. Run the reboot command in the serial terminal window tools to reboot the system:

```
# reboot
```

The system should reboot.

3-7-2. Close the GtkTerm window.

3-7-3. Power off the board.

Conclusion

In this lab, you have learned how to:
- Cross-compile Linux
- Boot Linux for an ARM Cortex-A9 MPcore system in QEMU
- Download a new image to the board via Ethernet

You will use these capabilities in subsequent labs.

Completed Solution

If you want to run the solution then copy **`BOOT.bin`** from the *`labsolution\lab2\SDCard`* directory onto a MICRO-SD card. Place the MICRO-SD card in the Zybo. Set Zybo in the MICRO-SD Card boot mode. Connect the Zybo to the host machine using Ethernet cable.

Run the following command to start DHCP server on the host:

```
[host] $ sudo service isc-dhcp-server restart
```

Copy the **`image.ub`** file from the *`labsolution\lab2\tftpboot`* directory into */tftpboot* directory.

Power ON the board. Set the terminal session. Interrupt the boot process when autoboot message is shown. Set the serverip address using the following command in the target board terminal window:

```
#set serverip 192.168.1.1
```

Run the netboot command:

```
#run netboot
```

Login into the system and test the lab.

Appendix A. General setup of lab network

This section explains the network configuration between PC, QEMU, and the development board.

Two class C subnets are introduced, “192.168.1.*” and “10.0.2.*”. Subnet “192.168.0.*” is used to establish the network connection between the PC and the board via physical connection (eth0 of the PC and the board) as shown in **Figure** .

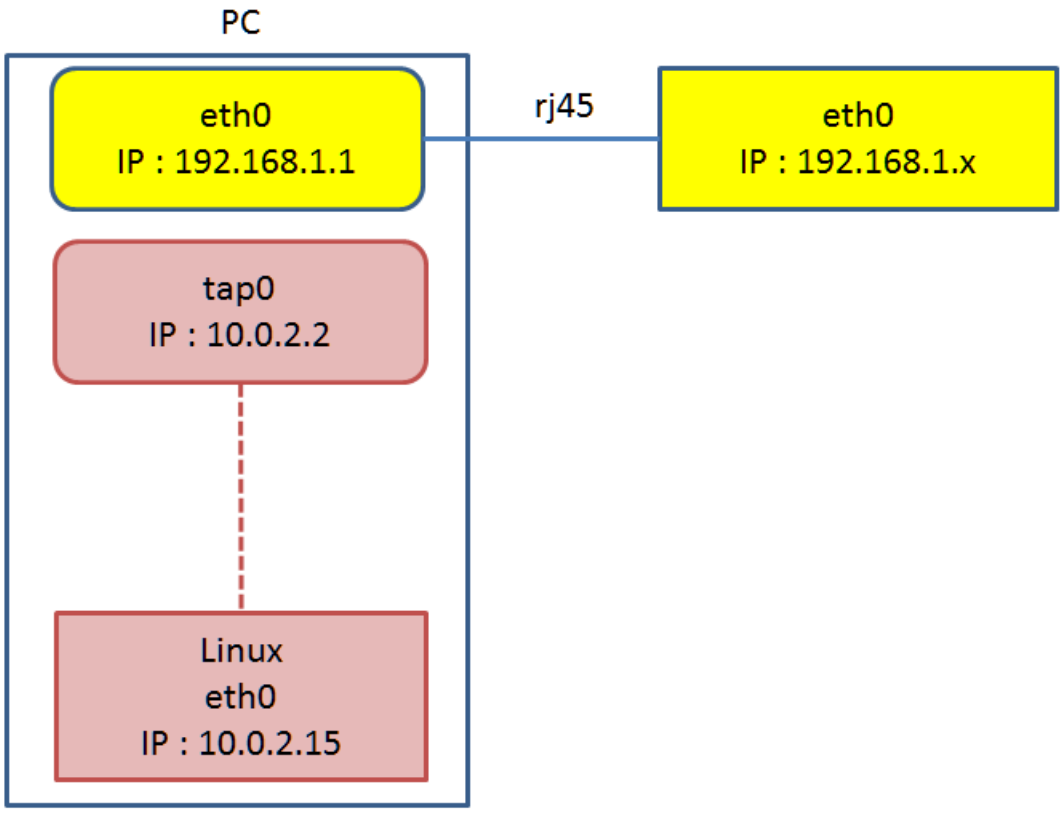

Figure 10. Network Configuration

Subnet “10.0.2.*” provides a communication link through the virtual network. QEMU uses TAP interfaces (in this case, the “tap0” interface) to provide full network capability for the ARM Cortex-A9 MPcore Linux guest OS.

These two class C subnets cannot communicate with each other. For example, the development board can only talk to PC but not the ARM Cortex-A9 MPcore Linux guest OS.