Lab 1: A First Look

Introduction

Embedded Linux is the use of a Linux operating system in embedded systems. Unlike desktop and server versions of Linux, embedded versions of Linux are designed for devices with relatively limited resources. The ARM® Cortex™-A9 processor used in Xilinx Zynq® All Programmable SoCs support embedded Linux. In most of the labs in this workshop, you will run embedded Linux, build using the PetaLinux tools, on the ARM Cortex-A9 MPcore.

This first lab is a basic introduction to embedded Linux and the development board that you are using for the workshop. The basic activities covered here will be used repeatedly through the later lab sessions, so be sure to ask your instructor if you have any questions or concerns.

Objectives

After completing this lab, you will be able to:

- Power on the development board used in the workshop
- Log in to the Zynq Linux system
- Make comparisons between the embedded Linux and desktop Linux environments

Typographic Conventions

Commands to be executed on the development (desktop) workstation look like the following:

```
[host]$ command and parameters
```

Commands to be executed on the ARM processor Linux target look like the following:

```
# run my Linux application
```

Before You Start

Before you start, ensure that the:

- Power switch is in the 'off' position
- A micro-USB cable is connected between the board's PROG UART connector to the PC
- The board is jumpered to be powered by the USB
- The Ethernet port on the development board connects to the Ethernet port of the desktop (host) machine
- The `BOOT.BIN` and `image.ub` files are copied from the ~/`emblnx/sources/lab1/SDCard` directory
- The micro-SD card is inserted back into the target board.
- Set the jumpers to boot from the micro-SD card as shown below.

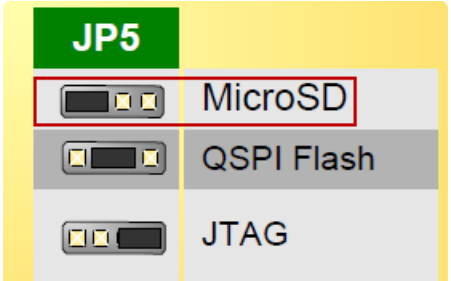


Figure 1. SD Card Boot - jumper settings

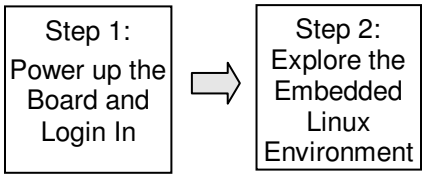
Initializing the Workshop Environment

By default, your Ubuntu image has already set up the workshop environment for you.

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
```

General Flow for this Lab



Power up the Board and Log in Step 1

1-1. Power up the board and run the DHCP server on the host.

1-1-1. Power ON the board.

1-1-2. Run the DHCP server:

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

1-2. Set the serial port terminal.

1-2-1. Ensure that /dev/ttyUSB1 is set to read/write access:

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

1-2-2. In the dashboard, in the Search field, enter the serial port.

1-2-3. Select the **Serial port terminal** application from the desktop.

1-2-4. Reset (BTN7) the board to see the booting info.

Watch the GtkTerm (Serial Port) console as the board goes through the boot process. Messages similar to the following can be found in the board console:

```
U-Boot 2014.01 (Oct 08 2014 - 14:07:06)

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

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

Hit any key to stop autoboot:  2


Freeing unused kernel memory: 3644K (c067c000 - c0a0b000)
INIT: version 2.88 booting
Starting Bootlog daemon: bootlogd.
Creating /dev/flash/* device nodes
random: dd urandom read with 8 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 (con
tinuing)
 Removing any system startup links for run-postinsts ...
  /etc/rcS.d/S99run-postinsts
INIT: Entering runlevel: 5
Configuring network interfaces... done.
Stopping Bootlog daemon: bootlogd.

Built with PetaLinux v2014.2 (Yocto 1.6) software /dev/ttyPS0
software login:
```

Figure 2. Linux booting process in the board console

Exploring the Embedded Linux Environment Step 1

2-1. Explore the booting message and basic Linux commands.

2-1-1. Scroll up in the terminal window and review the bootup log.
Existing Linux users should recognize the output.
You will see `image.ub` loading, drivers loading such as USB, SD, etc., and the starting of the uWeb server.

2-1-2. Log in by entering `root` as both the login and password.

2-1-3. Spend the next 15 minutes exploring basic Linux commands such as:
`ls -l, vi, whoami, date`

2-1-4. Run the following command to list the applications currently installed:
`# ls /bin`

2-2. Use the gpio-demo application to test the GPIOs. The gpio-demo application is used to write value to the GPIO peripheral or read value from the GPIO peripheral.

2-2-1. Use the following command to see the available GPIOs in the system.
`# ls /sys/class/gpio`

```
root@software:~# ls /sys/class/gpio/
export       gpiochip0    gpiochip244  gpiochip248  gpiochip252  unexport
```

Figure 3. Checking for the available GPIOs in the system

The GPIOs are presented as `gpiochip<ID>` in the directory. Have a look at the file `/sys/class/gpio/gpiochip<ID>/label`.

For example:
`# cat /sys/class/gpio/gpiochip244/label`

The GPIO label file contains the GPIO label. The label contains the GPIO's physical address information. The GPIO label format is `/amba@0/gpio@<PYHSICAL_ADDRESS>`.

```
root@software:~# cat /sys/class/gpio/gpiochip244/label
/amba@0/gpio@41200000
```

Figure 4. GPIO Physical Address Information

On the Zybo board, the on-board GPIOs are: four LEDs (four channels), four buttons (four channels), and four switches (four channels).

The `gpiochip<ID>` to GPIOs mapping is:
`gpiochip244 for 4 switches;`
`gpiochip248 for 4 LEDs;`
`gpiochip252 for 4 buttons;`

2-2-2. Run the following command to turn ON all four LEDs:

```
# gpio-demo -g 248 -o 15
```

Note: The output is in HEX format using the lower four bits of the HEX value written. For example, in this case the equivalent of d'15 is 0xF.

2-2-3. Run the following command to print the status of the four DIP switches:

```
# gpio-demo -g 234 -i
```

Note that you can try changing the DIP switch values.

2-3. Find the CPU information and interrupts.

Another interesting place to explore is the `/proc` directory. This is a virtual directory that provides a window into the kernel. For example, the file `/proc/cpuinfo` contains details about the CPU, `/proc/interrupts` provides interrupt statistics, and so on.

2-3-1. Enter the following command:

```
# cat /proc/cpuinfo
```

```
root@software:~# cat /proc/cpuinfo
processor       : 0
model name      : ARMv7 Processor rev 0 (v7l)
Features        : swp half thumb fastmult vfp edsp neon vfpv3 tls vfpd32
CPU implementer : 0x41
CPU architecture: 7
CPU variant     : 0x3
CPU part        : 0xc09
CPU revision    : 0

processor       : 1
model name      : ARMv7 Processor rev 0 (v7l)
Features        : swp half thumb fastmult vfp edsp neon vfpv3 tls vfpd32
CPU implementer : 0x41
CPU architecture: 7
CPU variant     : 0x3
CPU part        : 0xc09
CPU revision    : 0

Hardware        : Xilinx Zynq Platform
Revision        : 0000
Serial          : 0000000000000000
```

Figure 5. Viewing the CPU information

The contents of `/proc/cpuinfo` shows the processor information, such as its version and hardware features. Your `/proc/cpuinfo` may be different than above, depending on the configuration of the processor.

2-3-2. Enter the following command:

```
# cat /proc/interrupts
```

```
root@software:~# cat /proc/interrupts
           CPU0       CPU1
 27:          0          0       GIC  27  gt
 29:       1051        482       GIC  29  twd
 35:          0          0       GIC  35  f800c000.ps7-ocmc
 40:          0          0       GIC  40  f8007000.ps7-dev-cfg
 43:       6054          0       GIC  43  ttc_clockevent
 45:          0          0       GIC  45  f8003000.ps7-dma
 46:          0          0       GIC  46  f8003000.ps7-dma
 47:          0          0       GIC  47  f8003000.ps7-dma
 48:          0          0       GIC  48  f8003000.ps7-dma
 49:          0          0       GIC  49  f8003000.ps7-dma
 51:          3          0       GIC  51  e000d000.ps7-qspi
 54:          8          0       GIC  54  eth0
 56:         35          0       GIC  56  mmc0
 72:          0          0       GIC  72  f8003000.ps7-dma
 73:          0          0       GIC  73  f8003000.ps7-dma
 74:          0          0       GIC  74  f8003000.ps7-dma
 75:          0          0       GIC  75  f8003000.ps7-dma
 82:        304          0       GIC  82  xuartps
IPI1:         0        252  Timer broadcast interrupts
IPI2:      1400       1624  Rescheduling interrupts
IPI3:         0          0  Function call interrupts
IPI4:        34         67  Single function call interrupts
IPI5:         0          0  CPU stop interrupts
IPI6:        89         85  IRQ work interrupts
IPI7:         0          0  completion interrupts
Err:          0
```

Figure 6. Viewing the Interrupts

`/proc/interrupts` shows the interrupts information of the system. Your results may be different depending on when the command was executed and what were the other commands executed to access the hardware devices. `/proc/interrupts` tells you what interrupts are present in your system, their type, and how many interrupts have happened.

2-3-3. Open another terminal window on the desktop machine.

2-3-4. Enter `cat /proc/cpuinfo` and compare with the embedded Linux information by using the same command.

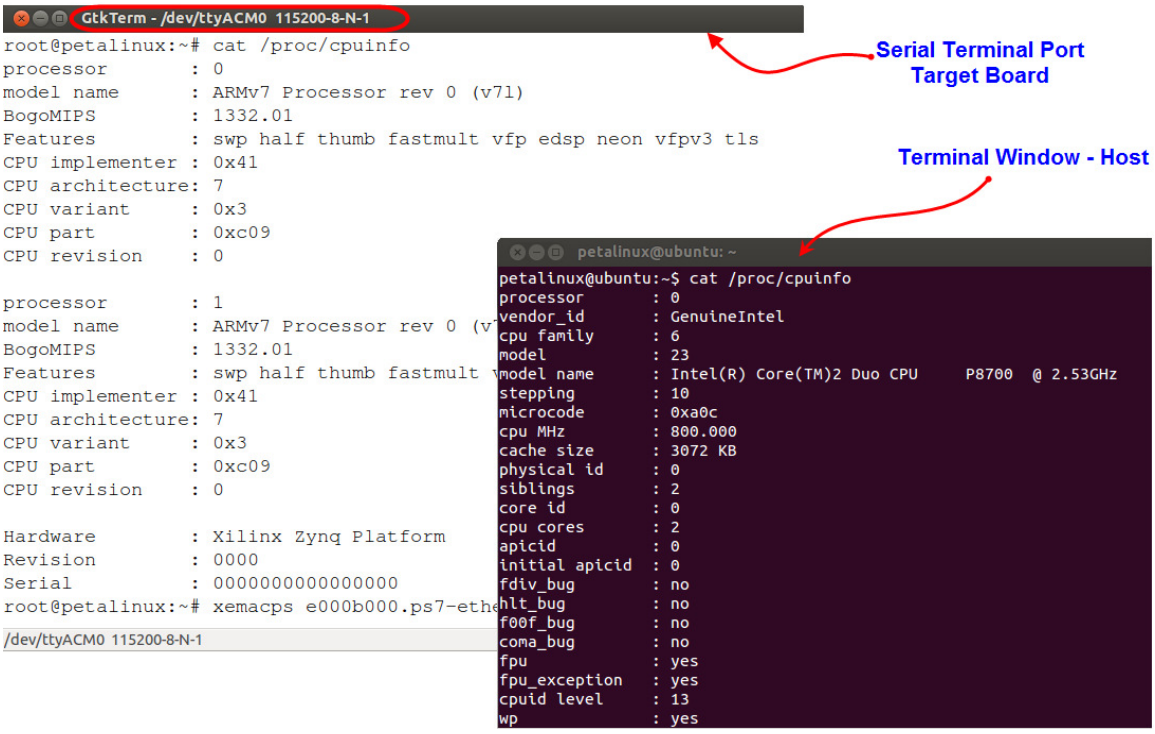


Figure 7. Serial Port and Terminal Window - cpuinfo

Another thing to note is the standard Linux directory structure: `/bin`, `/dev`, `/tmp`, `/var`, and so on.

2-4. Use `ping` command to test the network connection.

2-4-1. After the system boots, log into the system by entering `root` as both the login name and password.

2-4-2. Execute the ping command to ping the host machine.

```
# ping 192.168.1.1
```

You should see the response from the host machine.

2-4-3. Execute the ping command from the host machine terminal window to see the response from the target board.

```
[host]$ ping 192.168.1.2
```

The static ip address has been assigned when the system was built. You should see the response from the target machine.

2-4-4. Close the GtkTerm window

2-4-5. Power OFF the board.

Conclusion

The purpose of this lab was to introduce you to the embedded Linux target and demonstrate its heritage in the desktop Linux genealogy. This is one of the immediate benefits of embedded Linux. As an application and user environment, it has tremendous commonality with standard desktop Linux platforms.

Although brief, this introduction should have provided you with some basic experience with setting up and powering on the board, and logging into and navigating around the embedded Linux target. These basic capabilities will be expanded upon in subsequent lab sessions.

Completed Solution

If you want to run the solution then copy **`BOOT.bin`** and **`image.ub`** from the *`sources\lab1\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
```

Power ON the board. Set the terminal session.

Press PS-SRST (BTN7) button. Let the board boot. Login into the system and test the lab.