

Embedded Linux Components

2022.1

Abstract

This lab is a basic introduction to embedded Linux and the development board that you will use for the rest of the labs. You will perform some basic activities covered here that will be used repeatedly through the subsequent labs.

This lab should take approximately 45 minutes.

CloudShare Users Only

You are provided three attempts to access a lab, and the time allotted to complete each lab is 2X the time expected to complete the lab. Once the timer starts, you cannot pause the timer. Also, each lab attempt will reset the previous attempt—that is, your work from a previous attempt is not saved.

Objectives

After completing this lab, you will be able to:

- Power on the development board used in the labs
- Log in to the Zynq® UltraScale+™ MPSoC Linux system
- Make comparisons between the embedded Linux and desktop Linux environments

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-A53 processor used in the Zynq® UltraScale+™ MPSoC supports embedded Linux.

Here, you will build the embedded Linux environment and run it on the Arm Cortex-A53 processor using the PetaLinux tools.

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

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

Note: Here [host] \$ denotes the development (desktop) workstation where the PetaLinux tool is installed and where the Linux images will be built for the embedded target (Arm processor).

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

```
# run my Linux application
```

Understanding the Lab Environment

The labs and demos provided in this course are designed to run on a Linux platform.

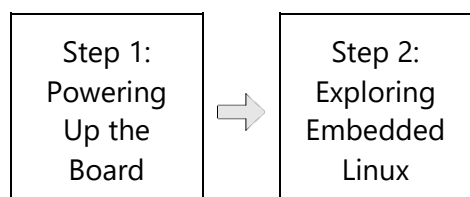
One environment variable is required: `TRAINING_PATH`, which points to where the lab files are located. This variable comes configured in the CloudShare/CustEd_VM environments.

Some tools can use this environment variable directly (that is, `$TRAINING_PATH` will be expanded), and some tools require manual expansion (`/home/xilinx/training` for the CloudShare/CustEd_VM environments). The lab instructions describe what to do for each tool.

Both the Vivado Design Suite and the Vitis platform offer a Tcl environment that is used by many labs. When the tool is launched, it starts with a clean Tcl environment with none of the procs or variables remaining from a previous launch of the tools.

If you sourced a Tcl script or manually set any Tcl variables and you closed the tool, when you reopen the tool, you will need to re-source the Tcl script and set any variables that the lab requires. This is also true of terminal windows—any variable settings will be cleared when a new terminal opens.

General Flow



Powering Up the Board and Logging In

Step 1

For this lab, pre-generated boot files are provided in the `support` directory. You will use these boot files to boot the Linux image onto the Zynq UltraScale+ MPSoC ZCU104 evaluation board.

For On-Demand/CloudShare users, if you have the ZCU104 evaluation board available locally, then copy the required boot files to the local machine and proceed with the steps below. You can also review the "Can I run the labs on my local machine?" question in the FAQ section of the On-Demand portal.

Before you start, ensure that:

- The serial cable connects the development board to the PC.
- The power cable for the development board is connected.
- The Ethernet port on the development board connects to the Ethernet port of the desktop (host) machine.
- The `BOOT.BIN`, `boot.scr` and `image.ub` files are copied from the `$TRAINING_PATH/FirstLook/support/SD_card` directory to the SD card, where `<target_board>` is ZCU104.
- The SD card is inserted back into the target board.

1-1. Bring up the ZCU104 board.

1-1-1. Ensure that the board is powered off and that the board is connected to the host with both USB and Ethernet cables.

Note: This lab will have you boot using SD Boot mode, which requires the mode pins to be set as described below.

1-1-2. Locate SW6 on the board.

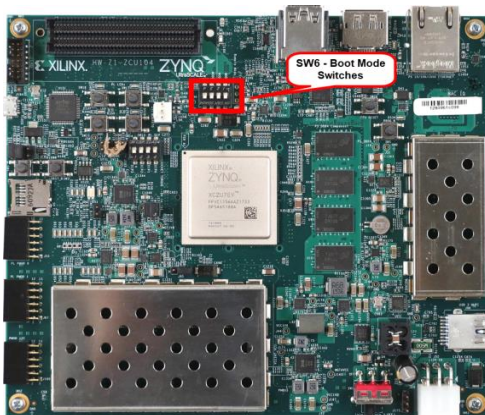


Figure 1-1: ZCU104 SW6 - Boot Mode Switches

1-1-3. Set SW6 as shown below to ensure that the board is configured to boot from **SD Boot**.

Note: Settings are shown to illustrate different boot mode settings. Make sure you select SD Boot.

Boot Mode	Mode Pins [3:0]	Mode SW6 [4:1]
JTAG	0000/0x0	ON,ON,ON,ON
QSPI32	0010/0x2 ⁽¹⁾	ON,ON,OFF,ON
SD1	1110/0xE	OFF,OFF,OFF,ON

Figure 1-2: Board Configuration

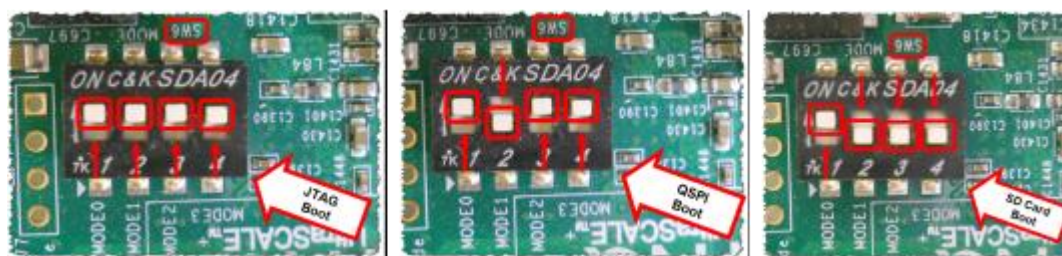


Figure 1-3: ZCU104 Boot Settings

1-1-4. Slide the power switch to the "ON" position to power on the board.

1-2. Launch the serial port terminal.

By default, USB devices will be enabled after you power on the board (wait for approximately 30–45 seconds).

1-2-1. Verify that **Devices > USB > Xilinx JTAG+3Serial** in VirtualBox is enabled.

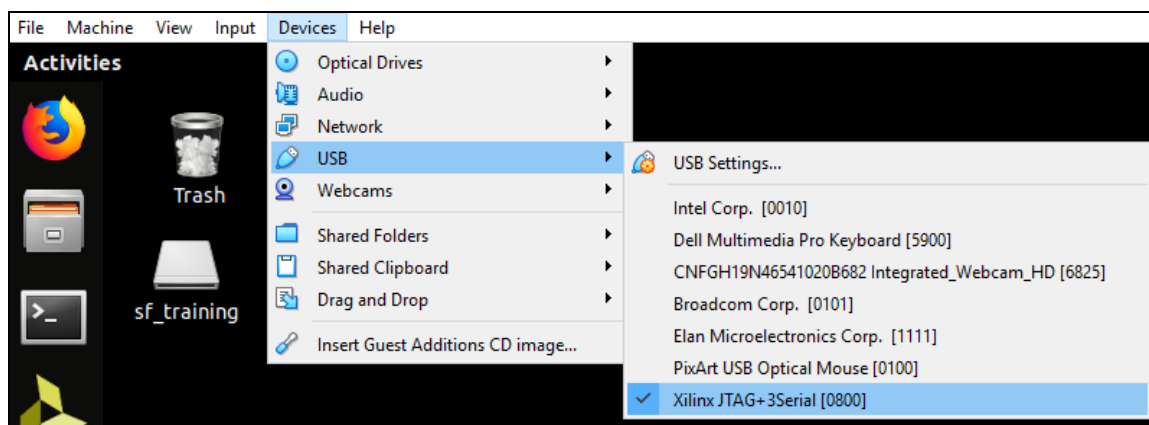


Figure 1-4: Enabling the USB-UART Connection in VirtualBox

- 1-2-2.** Click the **GtkTerm** icon in the taskbar to launch a serial terminal to view the Linux console of the ZCU104 board.

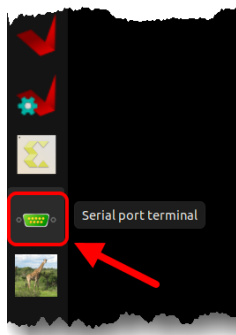


Figure 1-5: Launching a Terminal

Note: Wait for 15 seconds and open the serial port terminal application. If you see the dialog box "Cannot open /dev/ttyUSBX: Device or resource busy", open the serial port terminal application again after 10 seconds, or a few more seconds if necessary.

- 1-2-3.** Configure the serial terminal to connect the ttyUSBX port.

Note: In the Zynq® UltraScale+™ MPSoC, which is the SoC of the ZCU104 board, there are four UARTs (serial interfaces) available, all of which are accessible from a single JTAG/USB cable that is connected from the ZCU104 board and are mapped to ttyUSB0, ttyUSB1, ttyUSB2, and ttyUSB3.

UART mapping changes dynamically based on system to system. If you are not able to view the log on the ttyUSB0 port then simultaneously open new GTK terminals for ttyUSB1, ttyUSB2, and ttyUSB3 until you are able to view the log.

- 1-2-4.** Set the baud rate to **115200** while leaving the other settings at their defaults.

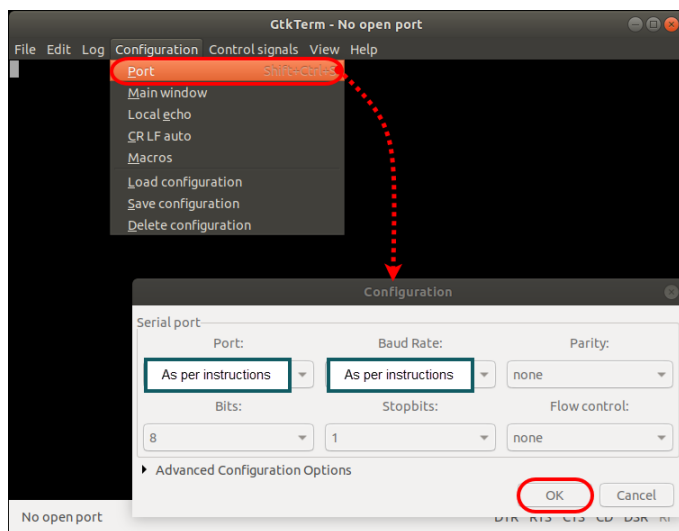


Figure 1-6: Configuring GtkTerm

- 1-2-5.** Watch the GtkTerm (serial port) console as the board goes through the boot process.

Exploring the Embedded Linux Environment

Step 2

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

- 2-1-1.** Log in to the system after it boots by entering **petalinux** as the both the login name and password, if prompted.

You may be asked to create a new password before the booting process completes. You can reuse **petalinux** as the password.

- 2-1-2.** Scroll up the terminal and review the boot log.

Existing Linux users should recognize the output.

You will see **image.ub** loading, as well as drivers such as USB, SD, etc.

- 2-1-3.** Make sure you are logged into the Linux system and that you see the **@BasicHwDesign** command prompt.

- 2-1-4.** Spend the next 10 minutes exploring basic Linux commands such as:

pwd, **vi**, **whoami**, **date**

To insert data, press: ***** and **i**, then enter the message.

To save the data, press: **w!**.

To exit the vi editor, press **:** and enter **q!**.

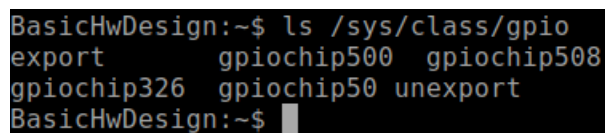
- 2-1-5.** Run the following command to list the applications currently installed:

```
# 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
```



```
BasicHwDesign:~$ ls /sys/class/gpio
export      gpiochip500  gpiochip508
gpiochip326 gpiochip50   unexport
BasicHwDesign:~$
```

Figure 1-7: Verifying 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/gpiochip504/label
```

The GPIO label file contains the GPIO label. The label contains the GPIO's physical address information. The GPIO label format is @<PHYSICAL_ADDRESS>.

```
BasicHwDesign:~$ cat /sys/class/gpio/gpiochip504/label
a0010000.gpio
BasicHwDesign:~$
```

Figure 1-8: GPIO Physical Address Information

The above is given as an example—you may get a different address.

For the ZCU104 board, the on-board GPIOs are four LEDs, four pushbuttons, and four DIP switches.

The `gpiochip<ID>` to GPIOs mapping is:

- `gpiochip500` for 4 buttons
- `gpiochip504` for 4 LEDs
- `gpiochip508` for 4 switches

- 2-2-2.** Run the following command to turn on all four LEDs (labeled as *LED0* to *LED3* on the board):

```
# sudo gpio-demo -g 504 -o 255
```

All the LEDs on the board will be ON (labeled as *LED0* to *LED3* on the board).

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'255 is 0xFF.

You can also try to enter the following command, which turns OFF all the LEDs:

```
# sudo gpio-demo -g 504 -o 0
```

- 2-2-3.** Run the following command to print the status of the four buttons:

```
# sudo gpio-demo -g 500 -i
```

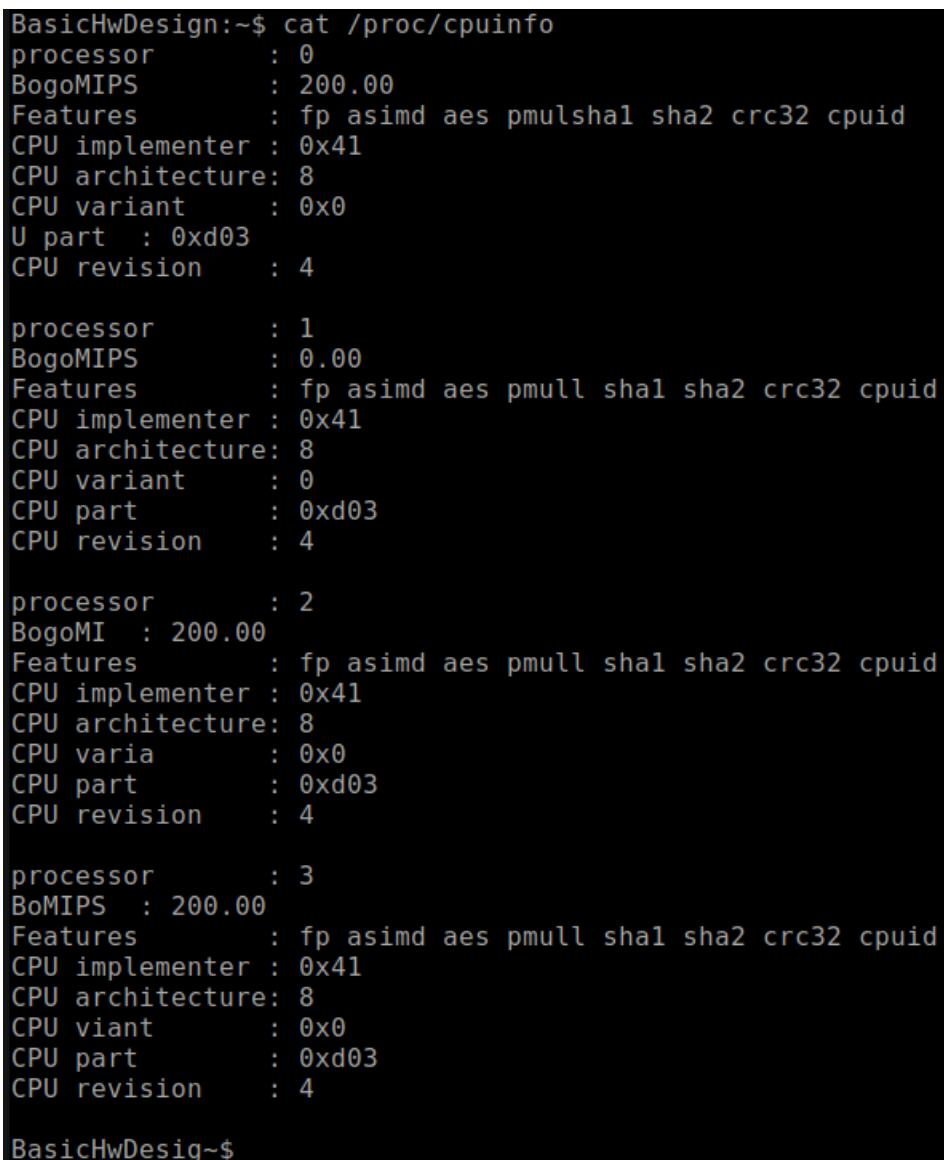
Note that you can try changing the value of the pushbuttons by pressing one of the buttons and re-entering the above command.

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
```



```
BasicHwDesign:~$ cat /proc/cpuinfo
processor       : 0
BogoMIPS      : 200.00
Features      : fp asimd aes pmulsha1 sha2 crc32 cpuid
CPU implementer : 0x41
CPU architecture: 8
CPU variant   : 0x0
U part       : 0xd03
CPU revision  : 4

processor       : 1
BogoMIPS      : 0.00
Features      : fp asimd aes pmull sha1 sha2 crc32 cpuid
CPU implementer : 0x41
CPU architecture: 8
CPU variant   : 0
CPU part      : 0xd03
CPU revision  : 4

processor       : 2
BogoMI       : 200.00
Features      : fp asimd aes pmull sha1 sha2 crc32 cpuid
CPU implementer : 0x41
CPU architecture: 8
CPU varia    : 0x0
CPU part      : 0xd03
CPU revision  : 4

processor       : 3
BoMIPS       : 200.00
Features      : fp asimd aes pmull sha1 sha2 crc32 cpuid
CPU implementer : 0x41
CPU architecture: 8
CPU viant    : 0x0
CPU part      : 0xd03
CPU revision  : 4

BasicHwDesig~$
```

Figure 1-9: Viewing the CPU Information

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

2-3-2. Enter the following command:

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

```
BasicHuDesign:~$ cat /proc/interrupts
CPU0          CPU1          CPU2          CPU3
11: 62790      37392      68908      50517      GICv2 30 Level arch_timer
14: 0          0          0          0          GICv2 67 Level zynq_lpi
15: 0          0          0          0          GIC 175 Level arm-pmu
16: 0          0          0          0          GICv2 176 Level arm-pmu
17: 0          0          0          0          GICv2 177 Level armmu
18: 0          0          0          0          GICv2 178 Level arm-pmu
20: 0          0          0          0          GICv2 155 Level axi-pmon, axi-pmon
21: 0          0          0          0          GICv2 156 Level zynqmp-dma
22: 0          0          0          0          GICv2 157 Level zynqmp-dma
23: 0          0          0          0          GICv2 158 Level zynqmp-dma
24: 0          0          0          0          Cv2 159 Level zynqmp-dma
25: 0          0          0          0          GICv2 160 Level zynqmp-dma
26: 0          0          0          0          GICv2 161 Level zynqmp-dma
27: 0          0          0          0          GICv2 162 Level zynqmp-dma
28: 0          0          0          0          GICv2 163 Level zynqmp-dm
30: 0          0          0          0          GICv2 10Level zynqmp-dma
31: 0          0          0          0          GICv2 110 Level zynqmp-dma
32: 0          0          0          0          GICv2 111 Level zynqmp-dma
33: 0          0          0          0          GICv2 112 Level zynqmp-dma
34: 0          0          0          0          GICv2 113 Level zynqmp-dma
35: 0          0          0          0          GICv2 114 Level zynqmp-dma
36: 0          0          0          0          GICv2 115 Level zynqmp-dma
37: 0          0          0          0          GICv2 116 Level zynqmp-dma
39: 564        0          0          0          Gv2 95 Level eth0, eth0
41: 0          0          0          0          GICv2 50 Level cdns-i2c
42: 0          0          0          0          GICv2 42 Level ff960000.memory-controller
43: 0          0          0          0          GICv2 57 Level axi-pmon, axi-pmon
44: 0          0          0          0          GICv2 4Level fff00000.spi
45: 0          0          0          0          GICv2 58 Level ffa60000.rtc
46: 0          0          0          0          GICv2 59 Level ffa60000.rtc
47: 0          0          0          0          GICv2 165 Level ahci-ceva[fd0c0000.ahci]
48: 63          0          0          0          GICv2 81 Level mmc0
49: 418         0          0          0          Gv2 53 Level xuartps
52: 0          0          0          0          GICv2 84 Edge fff50000.watchdog
: 0          0          0          0          GICv2 88 Lel ams-irq
54: 0          0          0          0          GICv2 154 Level fd4c0000.dma-controller
57: 0          0          0          0          GICv2 97 Level xhci-hcd:usb1
58: 3          0          0          0          GICv2 101 Level dwc3-otg
IPI0: 53        127        103        88        Rescheduling interrupts
IPI1: 2815      4036      3240      3787      Function call interrupts
IPI2: 0          0          0          0          CPU stop interrupts
IPI3: 0          0          0          0          CPU stop (for crash dump) interrupts
IPI4: 0          0          0          0          Tlr broadcast interrupts
IPI5: 0          0          0          0          IRQ work interrupts
IPI6: 0          0          0          0          CPU wake-up interrupts
Err 0
```

Figure 1-10: 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 the output with the embedded Linux information by using the same command.

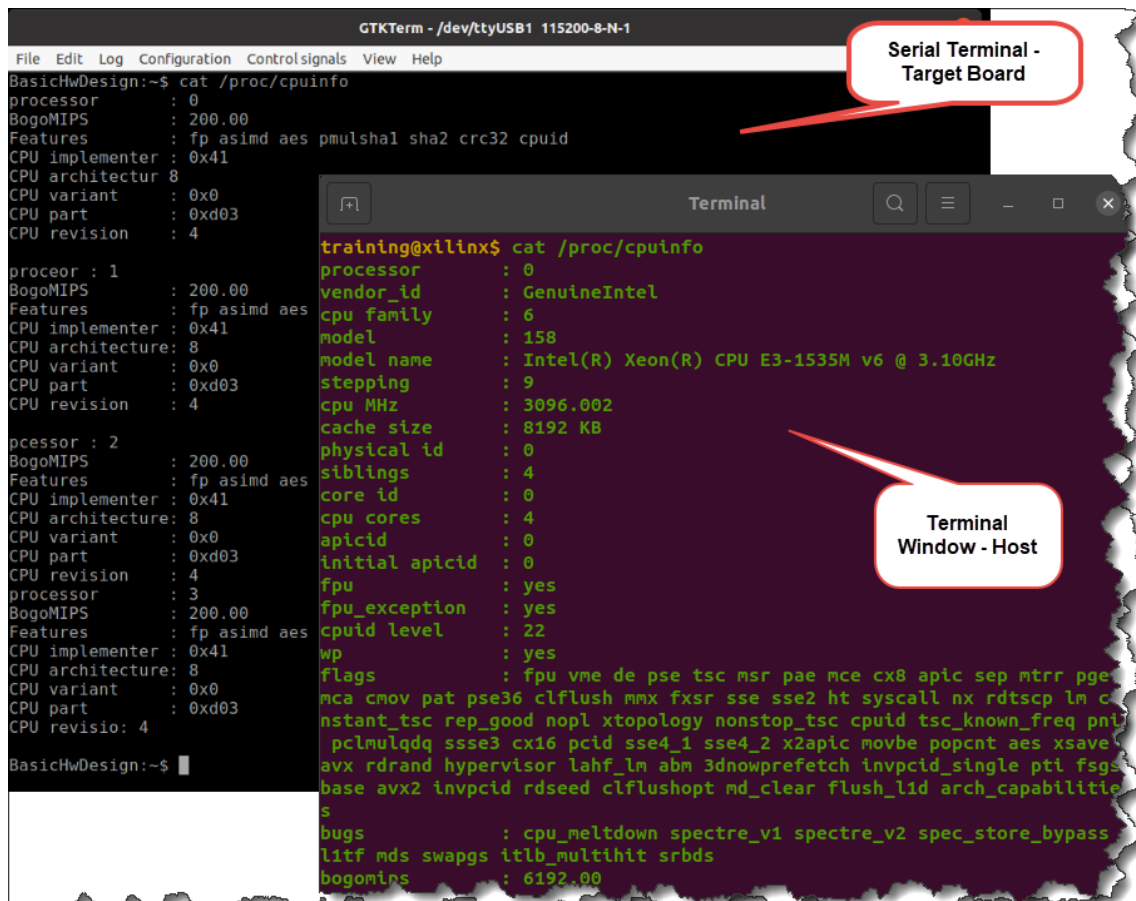


Figure 1-11: Serial Port and Terminal Window - `cpuinfo`

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

2-3-5. Close the serial terminal once you have finished and power off the board.

Some systems (particularly VMs) may be memory constrained. Removing the workspace frees a portion of the disk space, allowing other labs to be performed.

You can delete the directory containing the lab you just ran by using the graphical interface or the command-line interface. You can choose either mechanism. Both processes will recursively delete all the files in the `$TRAINING_PATH/FirstLook` directory.

2-4. [Optional] [Only for local VMs—not for CloudShare] Clean up the file system.

Using the GUI:

2-4-1. Using the graphical browser (Windows: press the **<Windows>** key + **<E>**; Linux: press **<Ctrl + N>**), navigate to `$TRAINING_PATH/FirstLook`.

2-4-2. Select **FirstLook**.

2-4-3. Press **<Delete>**.

-- OR --

Using the command line:

2-4-4. Open a terminal window (Windows: press the **<Windows>** key + **<R>**, then enter **cmd**; Linux: press **<Ctrl + Alt + T>**).

2-4-5. Enter the following command to delete the contents of the workspace:

[Windows users]: `rd /s /q $TRAINING_PATH/FirstLook`

[Linux users]: `rm -rf $TRAINING_PATH/FirstLook`

Summary

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.