ULTRA96 Custom Linux Image Tutorial

Corbin Thurlow

April 23, 2020

Tutorial Overview

Xilinx Tools and Host OS

- This tutorial will use Xilinx PetaLinux and Vivado 2018.2 tools with a host OS of Ubuntu 18.04 LTS
- It is important that the versions of these tools match (Ubuntu 16.04 LTS should be used with Xilinx PetaLinux 2018.2, however it is possible to use Ubuntu 18.04 LTS by modifying the tools as will be shown in this tutorial)
- Link to download Xilinx PetaLinux 2018.2 Installer: Xilinx PetaLinux v2018.2
- Link to download Xilinx Vivado 2018.2: Xilinx Vivado v2018.2
- Additional PetaLinux Documentation: PetaLinux Documentation

Creating Hardware Platform

- Download board files for Avnet Ultra96 Rev-1 here: Avnet U96 Board Files
- Place board file folders in $< Xilinx\_Vivado\_install\_dir > /data/boards/board\_files/$
- Create a Vivado project, and select Ultra96v1 as shown in figure 1

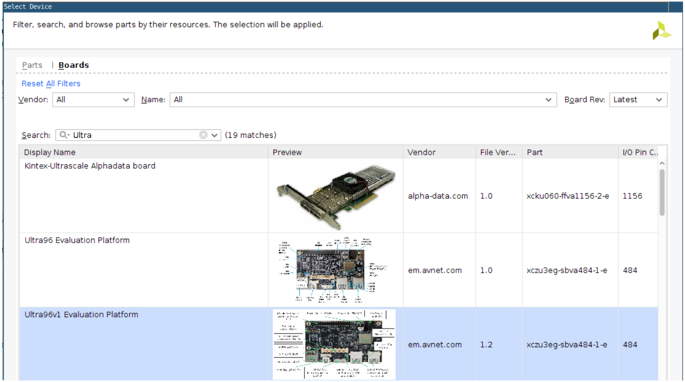


Figure 1: Select Ultra96v1 as the project board

Instantiating and Customizing Zynq MPSoC Core

- Create a new block diagram in Vivado, and select add IP
- Search for "Zynq" and select Zynq UltraScale+ MPSoC as shown in figure 2
- Add this IP to the block design

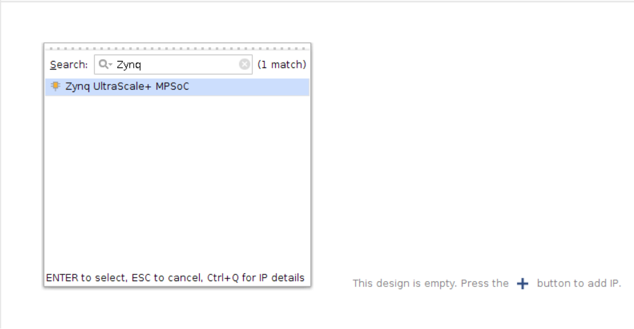


Figure 2: Adding the Zynq UltraScale+ MPSoC to the block diagram

Customizing Zynq MPSoC Core

- Click on "Run Block Automation" as shown in figure 3
- Next, click ok to apply presets to the IP core as show in figure 3

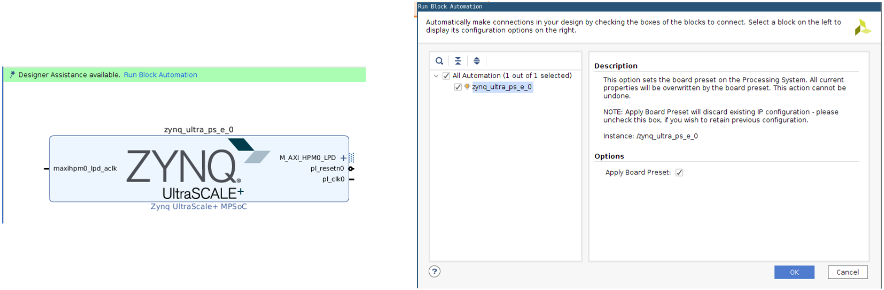


Figure 3: Apply MPSoC presets

Customizing Zynq MPSoC Core

- We need to disable some master interfaces between the PS-PL
- Double click on the MPSoC IP core and the menu shown in figure 4 will appear
- Click PS-PL Configuration > PS-PL Interfaces > Master Interfaces
- Uncheck the check boxes as shown in figure 4 and click "OK"

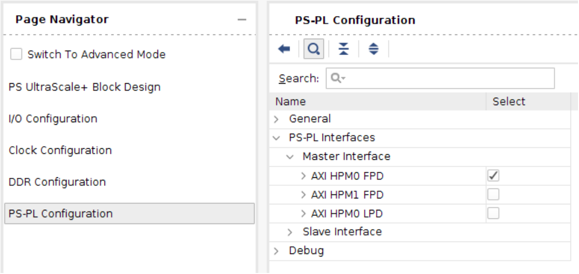


Figure 4: Remove Master Interfaces

Adding Custom Programmable Logic

- For this tutorial we will add a simple AXI BRAM controller that we will use later
- As shown in figure 5 search for "BRAM" and select to add AXI BRAM Controller

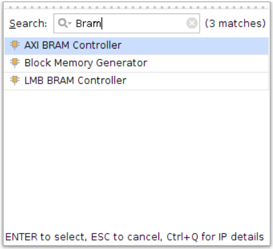


Figure 5: Adding AXI BRAM Controller

Adding Custom Programmable Logic

- Double click on the BRAM controller and select the settings as shown in figure 6

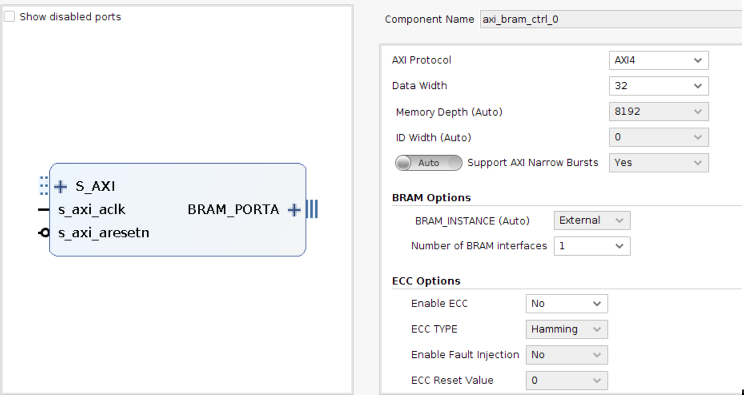


Figure 6: Configure AXI BRAM Controller

Adding Custom Programmable Logic

- Double click on the BRAM Block Memory Generator and disable "Enable Safety Circuit" as shown in figure 7

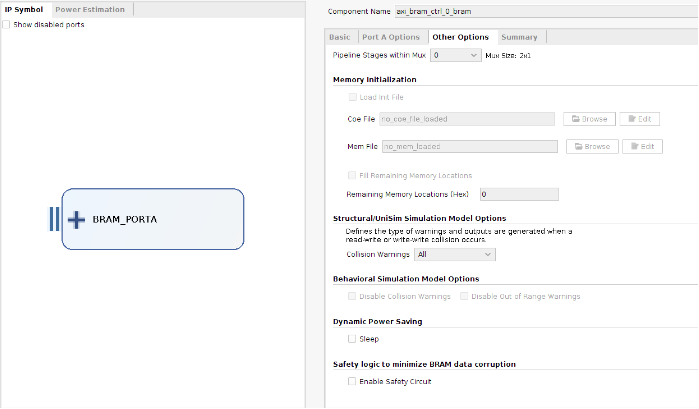


Figure 7: Configure Memory Generator

Adding Custom Programmable Logic

- Add a Processor Reset System as shown in figure 8

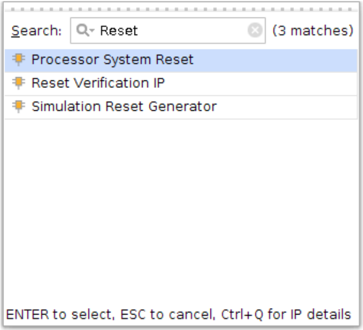


Figure 8: Processor Reset System

Adding Custom Programmable Logic

- Next, click "Run Connection Automation"
- A settings box like figure 9 will appear, select all boxes to connect the design
- This will connect all components together in the block diagram

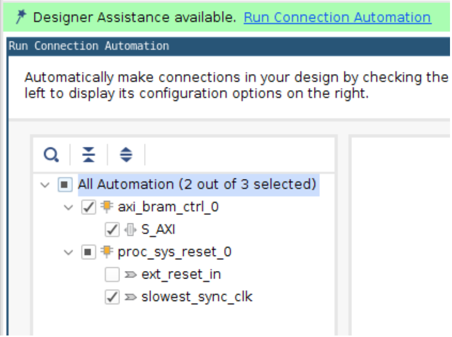


Figure 9: Connection Automation

Adding Custom Programmable Logic

- The final diagram should look like figure 10

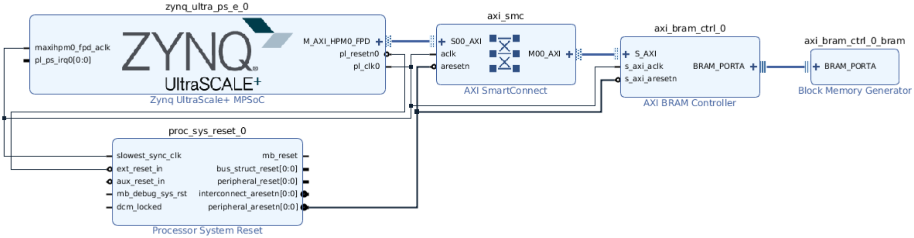


Figure 10: Final Block Diagram

Adding Custom Programmable Logic

- Next, look under the Address Editor tab and note the address space (beginning at 0x00A0000000) of the BRAM as shown in figure 11
- Finally, generate a bitstream and export the hardware including the bitstream as seen in figure 12

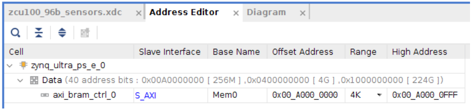


Figure 11: Address Editor Tab

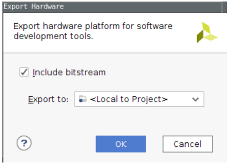


Figure 12: Export the Hardware design

Installing Xilinx PetaLinux Tools

- First, you may need to change the permissions on the PetaLinux installer by running the following command: sudo chmod 777 $petalinux - installer - file$
- Create an installation directory, the directory used here is "/tools/peta2018/"
- Run the following command
  - ./petalinux-v2018.2-final-installer.run /tools/peta2018 where "/tools/peta2018" is the destination directory (Permissions may need to change on the destination directory)
- It is likely that dependencies will be missing. The installer should let you know which packages need to be installed.

Creating Xilinx PetaLinux Project

- To create the PetaLinux project run the following command
  - petalinux-create -t project -n project-name - -template zynqMP
- This will create a folder "project-name", cd into this folder

Configuring PetaLinux Hardware Platform

- Next, we will set the hardware platform of the project to the hardware we created in Vivado by running the following command
  - petalinux-config - -get-hw-description=$< path - to - vivado - sdk >$
- The path to the sdk should contain an HDF file that was created when we exported the hardware
- The menu seen in figure 13 should appear after running this command (Ensure your terminal window is at least half of a window as the menu will fail to launch if the window is too small)

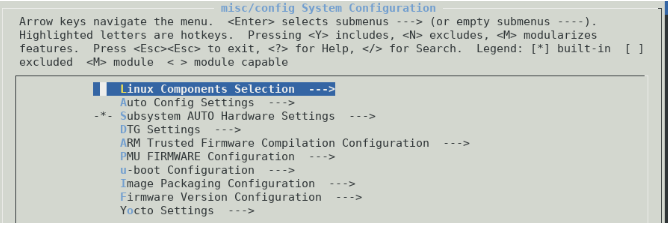


Figure 13: PetaLinux Hardware Menu

Changing Default PS UART

- Using the arrow keys, select Subsystem AUTO Hardware Settings > Serial Settings > Primary stdin/stdout
    - Select psu_uart_1 as shown in figure 14

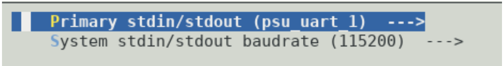


Figure 14: Change the PS UART

Setting Default Device Tree

- To set the default Device Tree, navigate to DTG Settings
  - Select MACHINE_NAME and change the name to "zcu100-revc" as shown in figure 15

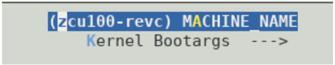


Figure 15: Change the DTG MACHINE_NAME

Modifying u-boot Configuration

- To modify the boot configuration file, navigate to u-boot Configuration
  - Select u-boot config target, and change the name to “xilinx_zynqmp_zcu100_revC_defconfig” as shown in figure 16

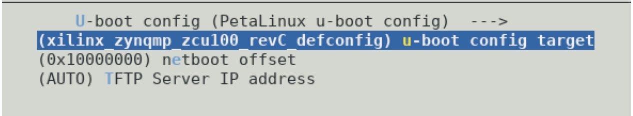


Figure 16: Change u-boot Configuration file

Setting Image Packaging Configuration

- To change the boot method to SD Card, navigate to Image Packaging Configuration
  - Select Root filesystem type, and select "SD CARD" as seen in figure 17

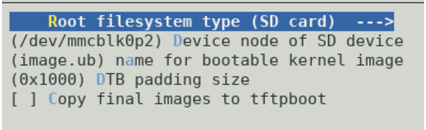


Figure 17: Change the boot method

Setting YOCTO Machine Name

- Finally we will change the YOCTO machine name by navigating to YOCTO Settings
  - Select YOCTO_MACHINE_NAME and change it to "ultra96-zynqmp" shown in figure 18

```
(ultra96-zynqmp) YOCTO_MACHINE_NAME
    TMPDIR Location  --->
    Parallel thread execution  --->
    Add pre-mirror url   --->
    Local sstate feeds settings  --->
[ ] Enable Debug Tweaks
[*] Enable Network sstate feeds
      Network sstate feeds URL  --->
[ ] Enable BB NO NETWORK
    User Layers  --->
```

Figure 18: Change the YOCTO_MACHINE_NAME

Customizing PetaLinux Kernel

- To configure the kernel, run the following command
  - petalinux-config -c kernel
- The menu seen in figure 19 will appear
- There are many different options that can be explored for kernel features
  - Different kernel features
  - Various power management settings can be modified
  - Both firmware and device drivers can be customized as well
- For this tutorial we will leave everything default

```
    General setup  --->
[*] Enable loadable module support  --->
[*] Enable the block layer  --->
    Platform selection  --->
    Bus support  --->
    Kernel Features  --->
    Boot options  --->
    Userspace binary formats  --->
    Power management options  --->
    CPU Power Management  --->
[*] Networking support  --->
    Device Drivers  --->
    Firmware Drivers  --->
[ ] ACPI (Advanced Configuration and Power Interface) Support
    File systems  --->
-*- Virtualization  --->
    Kernel hacking  --->
    Security options  --->
-*- Cryptographic API  --->
    Library routines  --->
```

Figure 19: Configure Kernel Menu

Customizing PetaLinux Root File System

- To configure the root filesystem, run the following command
  - petalinux-config -c rootfs
- The menu seen in figure 20 will appear
- There are many different options that can be explored for the root filesystem features
  - This is where various desired packages can be added to the image such as libstdc++, python, gdb and many others
  - You can easily search for desired packages by pressing / and then typing the name of the package
  - The root account password can also be modified by selecting "PetaLinux RootFS Settings"
  - Finally, if you have created any user modules or applications they can be included here under the apps and user packages menus (modules and apps will be discussed later)
- For this tutorial we will leave everything default

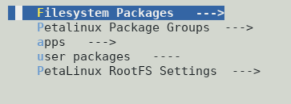


Figure 20: Configure Root Filesystem Menu

Adding Modules and Applications

- To add modules and applications run the following commands
  - petalinux-create -t apps -n app-name –enable
  - petalinux-create -t modules -n module-name –enable
- This tutorial will not cover these in details
- However, the PetaLinux documentation linked in this tutorial provides more details in chapter 7: Customizing the Rootfs, under the Creating and Adding Custom Modules and Applications sections.
  - These steps would be necessary for creating any custom Linux drivers for any custom hardware added in the Vivado design
- For this tutorial we will not build any modules or applications

Building the PetaLinux Platform

- To build the project after making any changes, run the following command
  - petalinux-build
- This step will take some time so be patient
- If you are attempting to run this on Ubuntu 18.04 LTS there will be various errors that could occur
  - An error related to setting the locale could occur as seen in figure 21
    - This link provides a solution for this problem
  - Another possible error is that the build process may say your system does not support “en_US.UTF8 locale”
    - This link provides a solution for this problem

```
ERROR: petalinux-user-image-1.0-r0 do_rootfs: [log_check] petalinux-user-image: found 4 error messages i
n the logfile:
[log_check] Failed to set locale, defaulting to C
[log_check] Failed to set locale, defaulting to C
[log_check] Failed to set locale, defaulting to C
[log_check] Failed to set locale, defaulting to C
```

Figure 21: Failed to set locale on host OS

Packaging the PetaLinux Platform

- To package the platform into a bootable image that can be placed on an SD card, run the following command
  - petalinuxpackage –boot –fsbl images/linux/zynqmp_fsbl.efl –u-boot images/linux/u-boot.elf –pmufw images/linux/pmufw.elf –fpga images/linux/system.bit
- After this command, a BOOT.BIN and image.ub will be located in the $< project - name >$/images/linux folder
- Format an SD card with two separate partitions
  - Partition one should be a FAT32 of size 500 MB or larger, name this partition boot
  - Partition two should be an EXT4 of size 4 GB or larger, name this partition root

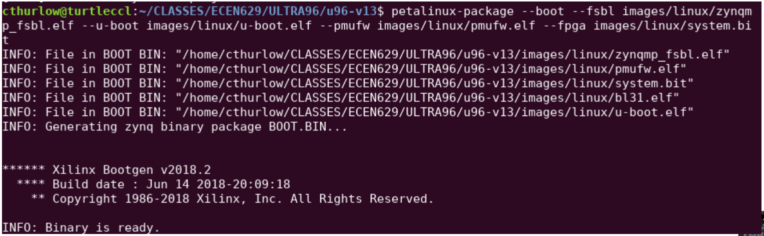


Figure 22: Successful package of boot files

Placing files on the SD Card

- With the SD card mounted, cd into images/linux, then run
  - sudo cp BOOT.BIN $< path - to - mount >$/boot
  - sudo cp image.ub $< path - to - mount >$/boot
  - sudo cp rootfs.cpio $< path - to - mount >$/root
- After copying all files, cd to $< path - to - mount >$/root and run
  - sudo pax -rvf rootfs.cpio .
- After this command, you should be able to see folders in the root directory as seen in figure 23

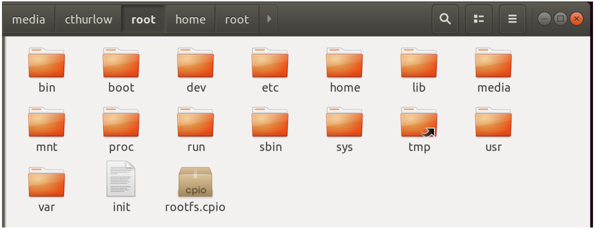
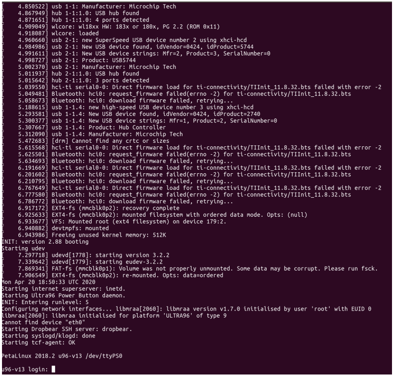


Figure 23: Root Filesystem on SD Card

Connect to UART 1

- Place the SD Card into the board
- Run the following command to connect to UART 1 connected to the PS
  - sudo screen /dev/ttyUSB1 115200
- Turn the power on the board and you should see a boot process similar to figure 24



```
[    4.850522] usb 1-1: Manufacturer: Microchip Tech
[    4.867949] hub 1-1:1.0: USB hub found
[    4.871651] hub 1-1:1.0: 4 ports detected
[    4.909049] wlcore: wl18xx HW: 183x or 180x, PG 2.2 (ROM 0x11)
[    4.918087] wlcore: loaded
[    4.960660] usb 2-1: new SuperSpeed USB device number 2 using xhci-hcd
[    4.984986] usb 2-1: New USB device found, idVendor=0424, idProduct=5744
[    4.991611] usb 2-1: New USB device strings: Mfr=2, Product=3, SerialNumber=0
[    4.998727] usb 2-1: Product: USB5744
[    5.002370] usb 2-1: Manufacturer: Microchip Tech
[    5.011937] hub 2-1:1.0: USB hub found
[    5.015642] hub 2-1:1.0: 3 ports detected
[    5.039550] hci-ti serial0-0: Direct firmware load for ti-connectivity/TIInit_11.8.32.bts failed with error -2
[    5.049481] Bluetooth: hci0: request_firmware failed(errno -2) for ti-connectivity/TIInit_11.8.32.bts
[    5.058673] Bluetooth: hci0: download firmware failed, retrying...
[    5.188615] usb 1-1.4: new high-speed USB device number 3 using xhci-hcd
[    5.293581] usb 1-1.4: New USB device found, idVendor=0424, idProduct=2740
[    5.300377] usb 1-1.4: New USB device strings: Mfr=1, Product=2, SerialNumber=0
[    5.307667] usb 1-1.4: Product: Hub Controller
[    5.312090] usb 1-1.4: Manufacturer: Microchip Tech
[    5.472683] [drm] Cannot find any crtc or sizes
[    5.615568] hci-ti serial0-0: Direct firmware load for ti-connectivity/TIInit_11.8.32.bts failed with error -2
[    5.625501] Bluetooth: hci0: request_firmware failed(errno -2) for ti-connectivity/TIInit_11.8.32.bts
[    5.634693] Bluetooth: hci0: download firmware failed, retrying...
[    6.191669] hci-ti serial0-0: Direct firmware load for ti-connectivity/TIInit_11.8.32.bts failed with error -2
[    6.201602] Bluetooth: hci0: request_firmware failed(errno -2) for ti-connectivity/TIInit_11.8.32.bts
[    6.210795] Bluetooth: hci0: download firmware failed, retrying...
[    6.767649] hci-ti serial0-0: Direct firmware load for ti-connectivity/TIInit_11.8.32.bts failed with error -2
[    6.777580] Bluetooth: hci0: request_firmware failed(errno -2) for ti-connectivity/TIInit_11.8.32.bts
[    6.786772] Bluetooth: hci0: download firmware failed, retrying...
[    6.917172] EXT4-fs (mmcblk0p2): recovery complete
[    6.925633] EXT4-fs (mmcblk0p2): mounted filesystem with ordered data mode. Opts: (null)
[    6.933677] VFS: Mounted root (ext4 filesystem) on device 179:2.
[    6.940882] devtmpfs: mounted
[    6.943986] Freeing unused kernel memory: 512K
INIT: version 2.88 booting
Starting udev
[    7.297718] udevd[1778]: starting version 3.2.2
[    7.339642] udevd[1779]: starting eudev-3.2.2
[    7.869341] FAT-fs (mmcblk0p1): Volume was not properly unmounted. Some data may be corrupt. Please run fsck.
[    7.906549] EXT4-fs (mmcblk0p2): re-mounted. Opts: data=ordered
Mon Apr 20 18:50:33 UTC 2020
Starting internet superserver: inetd.
Starting Ultra96 Power Button daemon.
INIT: Entering runlevel: 5
Configuring network interfaces... libmraa[2060]: libmraa version v1.7.0 initialised by user 'root' with EUID 0
libmraa[2060]: libmraa initialised for platform 'ULTRA96' of type 9
Cannot find device "eth0"
Starting Dropbear SSH server: dropbear.
Starting syslogd/klogd: done
Starting tcf-agent: OK

PetaLinux 2018.2 u96-v13 /dev/ttyPS0

u96-v13 login:
```

Figure 24: Successful boot of custom image

Basic Linux Commands

- To login, use root as the username and password
- Basic Linux commands are available as can be seen in figure 25

```
PetaLinux 2018.2 u96-v13 /dev/ttyPS0

u96-v13 login: root
Password:
root@u96-v13:~# 
```

```
root@u96-v13:~# ls
root@u96-v13:~# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: sit0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN group default qlen 1000
    link/sit 0.0.0.0 brd 0.0.0.0
3: wlan0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default qlen 1000
    link/ether f0:45:da:f9:26:79 brd ff:ff:ff:ff:ff:ff
root@u96-v13:~# ifconfig
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1%4882584/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

root@u96-v13:~# pwd
/home/root
```

Figure 25: Usage of basic Linux Commands

Read and Write using BRAM Controllers

- Since we included a BRAM controller in our hardware, we can use the Linux “devmem” command to read and write to the BRAM on the PL portion of the device as seen in figure 26
- To write a value, run the following command using the address space of the BRAM
  - devmem 0x00A0000000 32 $< value - to - write >$
- To read the value at the address, run the following command
  - devmem 0x00A0000000 32

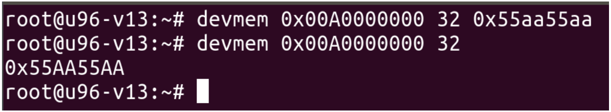


Figure 26: Use devmem to read and write to BRAM