Thesis Title

Institution Name

Author Name

Month Day, Year

Abstract

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut rhoncus nunc eu neque interdum placerat. Suspendisse vestibulum hendrerit vehicula. Etiam blandit eleifend pellentesque. Morbi in risus nec mi ultrices vehicula. Praesent porta dui non magna porttitor, et placerat arcu varius. Suspendisse a ullamcorper lectus, a porta mauris. Nulla et luctus mauris.

Declaration

I do hereby declare...

Acknowledgments

I want to thank...

Contents

Chapter 1

Introduction

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut rhoncus nunc eu neque interdum placerat. Suspendisse vestibulum hendrerit vehicula. Etiam blandit eleifend pellentesque. Morbi in risus nec mi ultrices vehicula. Praesent porta dui non magna porttitor, et placerat arcu varius. Suspendisse a ullamcorper lectus, a porta mauris. Nulla et luctus mauris.

Chapter 2

Chapter One Title

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut rhoncus nunc eu neque interdum placerat. Suspendisse vestibulum hendrerit vehicula. Etiam blandit eleifend pellentesque. Morbi in risus nec mi ultrices vehicula. Praesent porta dui non magna porttitor, et placerat arcu varius. Suspendisse a ullamcorper lectus, a porta mauris. Nulla et luctus mauris.

2.1 Chapter Subsection Title

Pellentesque eleifend dui vel efficitur tincidunt. Morbi mattis augue vel augue maximus, eget consectetur sem posuere. Nam scelerisque turpis sit amet laoreet vulputate. Vivamus eget dui ligula. Pellentesque eleifend diam ac turpis aliquam, ac dictum velit mollis. Interdum et malesuada fames ac ante ipsum primis in faucibus. Sed accumsan tellus sed purus sodales tempus. Cras posuere, tortor non condimentum venenatis, ipsum arcu rhoncus est, eget vestibulum lorem tortor eu nisi. Sed a pharetra odio.

2.2 Chapter Subsection Title

Cras in orci auctor, pharetra purus in, ultrices urna. Curabitur quis ultricies odio. Integer consequat lorem condimentum, imperdiet lacus in, placerat nisi. Quisque in ipsum a sapien hendrerit consequat non eu purus. Pellentesque sed consequat lorem, et facilisis ante. Phasellus mattis est eget imperdiet lobortis. Etiam venenatis mi vel pretium facilisis. Sed vel sapien non nisl sodales finibus a convallis ante. Nam iaculis vulputate orci, interdum malesuada lectus. Nulla malesuada erat vitae vestibulum feugiat. Vivamus imperdiet consectetur venenatis. Nam mollis faucibus nulla. Phasellus euismod, turpis non feugiat dapibus, sem ante rutrum neque, vel bibendum eros est sed dui. Ut interdum, nisi ac imperdiet tristique, ante sapien feugiat arcu, tempor finibus mauris lorem ac turpis.

2.3 Chapter Subsection Title

Etiam congue ullamcorper gravida. Mauris sapien lectus, iaculis vel luctus in, aliquam a urna. Quisque venenatis sollicitudin risus sed ultrices. Curabitur sit amet diam condimentum, vehicula mauris at, imperdiet leo. Curabitur quis posuere odio, vel interdum lectus. Proin a ligula vitae quam lobortis dignissim at quis risus. Aenean eros diam, dignissim sed dapibus eget, viverra tincidunt arcu. Phasellus vitae quam malesuada, eleifend lorem non, cursus urna. Morbi non nulla vel diam ullamcorper eleifend ac at lorem. Donec sed aliquet mauris. Nullam euismod aliquet odio eu iaculis. Nullam non auctor urna. Donec ipsum enim, interdum et lectus sed, consectetur condimentum ligula. In pharetra pretium vestibulum. Donec ut nisl a urna aliquam iaculis.

Suspendisse sit amet justo id urna hendrerit malesuada eu nec tellus. Phasellus vitae ipsum in nisi dapibus molestie. Curabitur sed ex iaculis, venenatis justo eget, dapibus sapien. Curabitur porttitor ex quis elit rutrum finibus. Sed varius cursus ex sit amet commodo. Sed tortor nunc, sodales non felis eu, vulputate congue ante. Quisque eu condimentum sapien. In vel neque dictum, imperdiet quam id, sollicitudin purus. Ut in augue vel purus interdum mollis sollicitudin bibendum nulla. Donec id dictum sapien, ac laoreet massa. Vestibulum mollis tincidunt enim. Suspendisse aliquet erat nec porta accumsan. Praesent lobortis lectus vehicula, lobortis lorem in, pretium neque. Cras sit amet aliquet justo. Aenean metus lacus, fermentum eu interdum nec, fermentum condimentum sem. Donec non neque eu ligula cursus sollicitudin quis sit amet sapien.

Chapter 3

Chapter Two Title

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut rhoncus nunc eu neque interdum placerat. Suspendisse vestibulum hendrerit vehicula. Etiam blandit eleifend pellentesque. Morbi in risus nec mi ultrices vehicula. Praesent porta dui non magna porttitor, et placerat arcu varius. Suspendisse a ullamcorper lectus, a porta mauris. Nulla et luctus mauris.

3.1 Chapter Subsection Title

Pellentesque eleifend dui vel efficitur tincidunt. Morbi mattis augue vel augue maximus, eget consectetur sem posuere. Nam scelerisque turpis sit amet laoreet vulputate. Vivamus eget dui ligula. Pellentesque eleifend diam ac turpis aliquam, ac dictum velit mollis. Interdum et malesuada fames ac ante ipsum primis in faucibus. Sed accumsan tellus sed purus sodales tempus. Cras posuere, tortor non condimentum venenatis, ipsum arcu rhoncus est, eget vestibulum lorem tortor eu nisi. Sed a pharetra odio.

3.2 Chapter Subsection Title

Cras in orci auctor, pharetra purus in, ultrices urna. Curabitur quis ultricies odio. Integer consequat lorem condimentum, imperdiet lacus in, placerat nisi. Quisque in ipsum a sapien hendrerit consequat non eu purus. Pellentesque sed consequat lorem, et facilisis ante. Phasellus mattis est eget imperdiet lobortis. Etiam venenatis mi vel pretium facilisis. Sed vel sapien non nisl sodales finibus a convallis ante. Nam iaculis vulputate orci, interdum malesuada lectus. Nulla malesuada erat vitae vestibulum feugiat. Vivamus imperdiet consectetur venenatis. Nam mollis faucibus nulla. Phasellus euismod, turpis non feugiat dapibus, sem ante rutrum neque, vel bibendum eros est sed dui. Ut interdum, nisi ac imperdiet tristique, ante sapien feugiat arcu, tempor finibus mauris lorem ac turpis.

3.3 Chapter Subsection Title

Etiam congue ullamcorper gravida. Mauris sapien lectus, iaculis vel luctus in, aliquam a urna. Quisque venenatis sollicitudin risus sed ultrices. Curabitur sit amet diam condimentum, vehicula mauris at, imperdiet leo. Curabitur quis posuere odio, vel interdum lectus. Proin a ligula vitae quam lobortis dignissim at quis risus. Aenean eros diam, dignissim sed dapibus eget, viverra tincidunt arcu. Phasellus vitae quam malesuada, eleifend lorem non, cursus urna. Morbi non nulla vel diam ullamcorper eleifend ac at lorem. Donec sed aliquet mauris. Nullam euismod aliquet odio eu iaculis. Nullam non auctor urna. Donec ipsum enim, interdum et lectus sed, consectetur condimentum ligula. In pharetra pretium vestibulum. Donec ut nisl a urna aliquam iaculis.

Suspendisse sit amet justo id urna hendrerit malesuada eu nec tellus. Phasellus vitae ipsum in nisi dapibus molestie. Curabitur sed ex iaculis, venenatis justo eget, dapibus sapien. Curabitur porttitor ex quis elit rutrum finibus. Sed varius cursus ex sit amet commodo. Sed tortor nunc, sodales non felis eu, vulputate congue ante. Quisque eu condimentum sapien. In vel neque dictum, imperdiet quam id, sollicitudin purus. Ut in augue vel purus interdum mollis sollicitudin bibendum nulla. Donec id dictum sapien, ac laoreet massa. Vestibulum mollis tincidunt enim. Suspendisse aliquet erat nec porta accumsan. Praesent lobortis lectus vehicula, lobortis lorem in, pretium neque. Cras sit amet aliquet justo. Aenean metus lacus, fermentum eu interdum nec, fermentum condimentum sem. Donec non neque eu ligula cursus sollicitudin quis sit amet sapien.

Chapter 4

BeagleBone Black with Heavy Vehicle Truck Cape

The Heavy Truck Cape with BeagleBone is a versatile Linux device designed for use with heavy vehicles. It has the ability to interface with different vehicle communication protocols both in and outside of userspace. This guide is designed to inform the reader over the many features of the device, as well as how to build your own tools.

Figure 4.1: Heavy Truck Cape w/ Beaglebone Black

4.1 Device Hardware

The Heavy Vehicle Truck Cape device is made up of two primary components: the BeagleBone Black and the Truck Cape. The cape provides additional peripheral connections

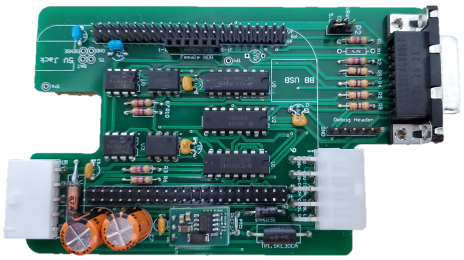

Figure 4.2: Truck Cape Board w/o BeagleBone.

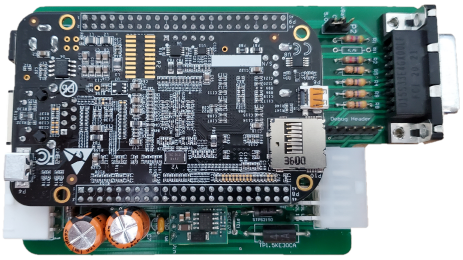

Figure 4.3: Truck Cape Board w/ BeagleBone Installed.

for the BeagleBone. The BeagleBone is a Linux microcomputer with two on-board microcontrollers called "Programmable Realtime Units" (PRUs).

4.1.1 BeagleBone Black

The BeagleBone Black is a Linux Microcomputer. It is called a "microcomputer" because it is a feature complete computer small enough to fit in the palm of your hand. With an on-board USB port and a micro-HDMI connector, one could connect the BeagleBone to a monitor and a keyboard for a "full" computer experience.

4.1.2 Truck Cape

The Heavy Vehicle Truck Cape provides more interface connections for the BeagleBone as well as power delivery for the components on the cape.

As shown in figure 2 and figure 3, the BeagleBone is installed directly onto the cape board. It is then placed inside of an enclosure (fig. 1). Very little logic is performed on the cape. Its primary function is to provide peripheral interfaces for the BeagleBone to use. The logic performed by the cape is relegated to lower-level connection logic for CAN and other vehicle communication standards.

4.1.3 Device Connections

4.1.4 Future Work

Future work may include the installation of a newer form of BeagleBone, such as the BeagleBone Black Wireless. The current iteration of the Truck Cape has a few design flaws which must be fixed manually at the end of production. Future work may also include a new Truck Cape schematic without these design artifacts and with surface-mounted components.

4.1.5 Device Assembly

This section may be out of date (January 2021). It is written for the v4 hardware. The most detailed and up-to-date guide to assemble the device may be found on the Truck Cape Projects GitHub Repository.

PCB Assembly

Hardware Errata

This section is unique to the v4 version of the hardware.

BeagleBone Truck Cape Assembly

Laser Etching and Cutting the Enclosure

Deutch 9 Pin to Molex Connector for Enclosure

Enclosure Assembly

4.2 Device Firmware

All firmware is located on the BeagleBone. As mentioned earlier, the truck cape has very litinternal logic. The logic performed by the cape is performed on prefabricated microchips suchthe MCP 2562 CAN Controller. Although one could install any number of different operating systand device firmwares, this guide will focus on the image and software used by the Systems CyProgram at Colorado State Universi

4.2.1 BootLoader

A bootloader is a low-level software which is used to install firmware onto an embeddevice. The BeagleBone Black uses u-boot for its bootloader. However, the easiest wayprogram the BeagleBone firmware is to create a flashable SD Car

4.2.2 Operating System

While varying operating systems may be installed on the BeagleBone, the chosen operatsystem for this guide is a flavor of Debian Linux. The most detailed and up-to-date guideconfigure the operating system may be found on the Truck Cape Projects GitHub Repository.

4.2.3 Future Work

There are many ways to flash software onto the BeagleBone, including some which are possithrough remote access. Future work may include a toolchain for the remote reprogrammingHeavy Vehicle Truck Cape Devices.

4.3 Embedded Linux Operating System: Installation, Basic Use, and Configuration

This portion of the guide covers the initial installation and configuration of the Linux operating system. This guide may be out of date (January 2021), so refer to the Truck Cape Projects GitHub Repository for the most up-to-date installation instructions.

If you already have a fully configured BeagleBone, you may skip ahead to PC Necessary Software.

4.3.1 Base Installation

The Linux image to run on the BeagleBone's eMMC can be downloaded, decompressed, imaged to an SD card. When the BeagleBone Black boots from the SD card, it will burn the eMMC to have the necessary contents to run the exercises in this repository.

For the impatient, here is a link to the most current image based on the modifications: TruckCapeImage-2020-11-14_g.xz

To start from scratch, here is a link to the chosen Linux base image: bone-eMMC-flasher-debian-10.3-iot-armhf-2020-04-06-4gb.img.xz

Installing the Linux Image via SD Card Flasher Image

This process is the same if you are starting from the pre-built image or from scratch.

1. Download your chosen image file (pre-built or scratch)
2. Using a compression utility such as 7-Zip or WinRAR decompress the image
3. Using a program like Win32DiskImager or Balena Etcher burn the image file onto a 4 GB SD Card.
4. Disconnect power from the BeagleBone, then insert the SD card into the SD card slot on the back of the BeagleBone.
5. While holding down the button near the USB Port (Red Circle in Fig. 4.4), Power On the BeagleBone. Continue holding the button until all four of the User LEDs (Blue Rectangle in Fig. 4.4) turn on. Once all four are lit, release the button.

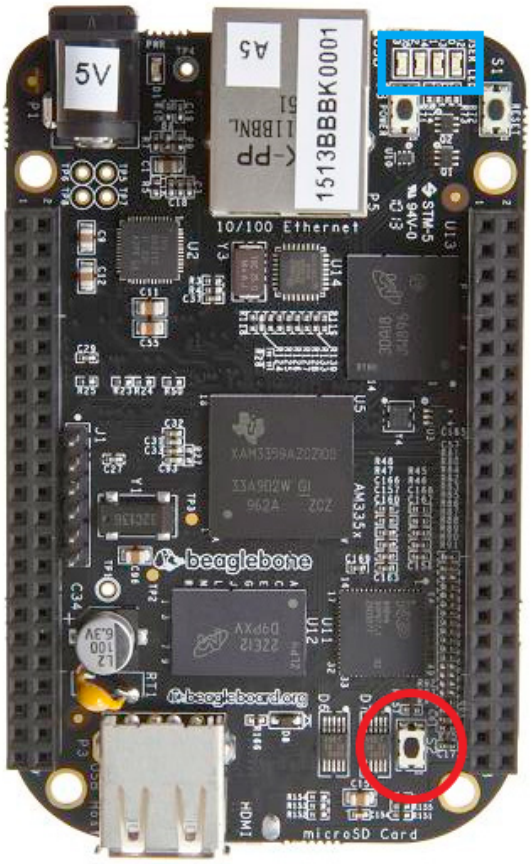

Figure 4.4: BeagleBone Format Button (Red Circle) and User LEDs (Blue Rectangle)

6. The LEDs will then flash intermittenly for a few seconds before assuming a back-and-forth pattern. This process takes some time, up to ten or even fifteen minutes. Eventually, the pattern will stop, all LEDs will light up before turning off.

7. Once all of the LEDs are off, disconnect power, remove the SD card from the slot, and power on the BeagleBone. The first boot may take longer than others.

8. You may now connect to the BeagleBone

Building and Installing Linux Kernel from Scratch

4.3.2 PC Necessary Software

These Linux images do not have the video output enabled. They are designed for a host device (PC) to log in to the terminal of the BeagleBone. If this is your first time with a Heavy Truck Cape with BeagleBoneBlack, then you'll need some software on your computer to interface with it.

Windows Software

Windows users will need the following:

1. SSH Software: PuTTY
2. SCP Software: WinSCP or FileZilla

Linux Software

Linux users will need the following:

1. SSH Software: SSH
2. SCP Software: SCP

Mac Software

Mac users will need the following terminal commands:

1. SSH Software: SSH
2. SCP Software: SCP

4.3.3 Remote Connecting to the Beaglebone

Logging into Terminal through USB with Putty

1. Plug in the a Mini USB cable between the BeagleBone and your computer
2. Give the system some time to boot. This may take several minutes. In Windows, the BeagleBone should appear in File Explorer when ready.
3. Open PuTTY

4. Specify the Host Name (or IP address) as 192.168.7.2 and port 22 (Fig. 4.5).

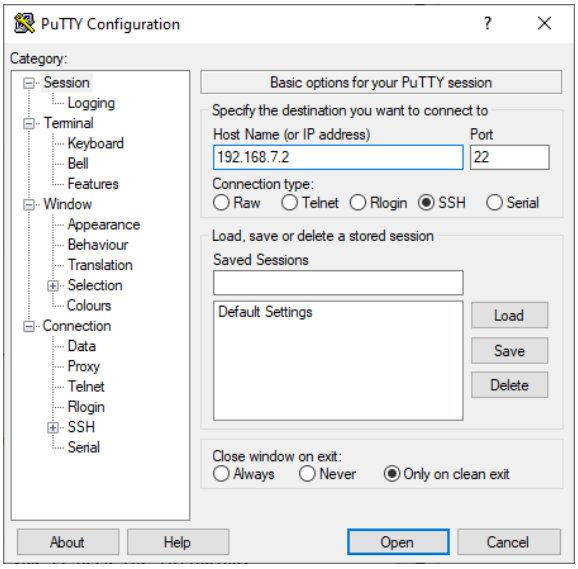

Figure 4.5: PuTTY Window

5. (Optional) For easier access in the future, the session access information may be saved in PuTTY. Write <username>@<ipaddress>in the "Saved Sessions" field. For this guide, the session is "debian@192.168.7.2" (Fig. 4.6). Then click "save"

6. Press Open

7. Enter the login credentials:

   Default Credentials U: debian P: temppwd

   (If connected to the internet, this password should be changed).

   If the password has been changed from the default, contact an administrator for access.

   There may be a warning about the security of the device. This is in reference to the SSH certificate offered by the device. This may be accepted.

8. PuTTY should now be logged into the Debian account on the BeagleBone. A Window should open similar to Fig. 4.7.

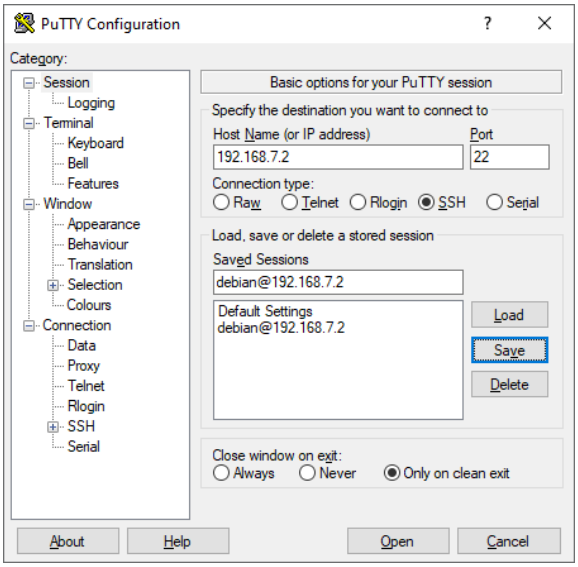

Figure 4.6: PuTTY Saved Session

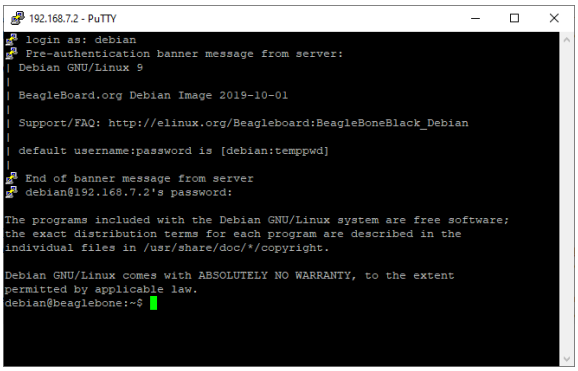

Figure 4.7: Open Session

Logging into Terminal through USB with SSH (Linux and Mac)

4.3.4 File Transfer between Computer and BeagleBone

The easiest way to copy and move files between a Windows computer and a BeagleBone is through an SCP software like WinSCP or FileZilla.

File Transfer with WinSCP

WinSCP is an SCP program available for Windows.

1. Set the BeagleBone as if starting an SSH connection. Either ensure that the BeagleBone is fully connected via USB or is powered and connected to the network.

2. Start WinSCP on the Windows Computer

3. Enter BeagleBone access credentials. This process is very similar to starting a PuTTY connection (Fig. 4.8)

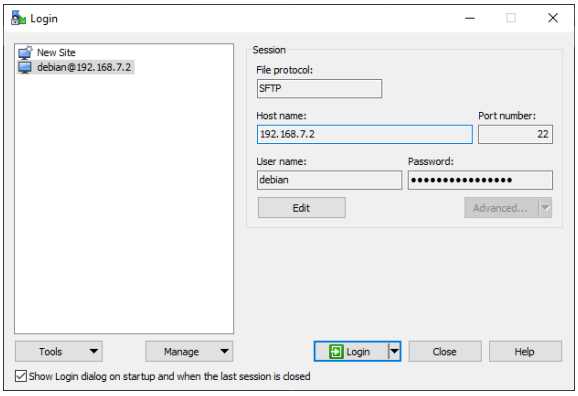

Figure 4.8: WinSCP Credentials Menu

4. Optional: If desired, save the credentials for quick future access.

5. There may be some popups about the security of the connection. This is due to how the BeagleBone handles certificates. They may be accepted.

6. If the connection is successful, a dual-explorer window will open. The left pane is the Windows Computer file system, the right pane is the BeagleBone file system. You may navigate the two file systems and move/copy files.

7. When finished, close the window.

File Transfer with FileZilla

1. Set the BeagleBone as if starting an SSH connection. Either ensure that the BeagleBone is fully connected via USB or is powered and connected to the network.

2. Start FileZilla on the Windows Computer

3. Enter BeagleBone access credentials at the top of the window. This process is very similar to starting a PuTTY connection (Fig. 4.9).

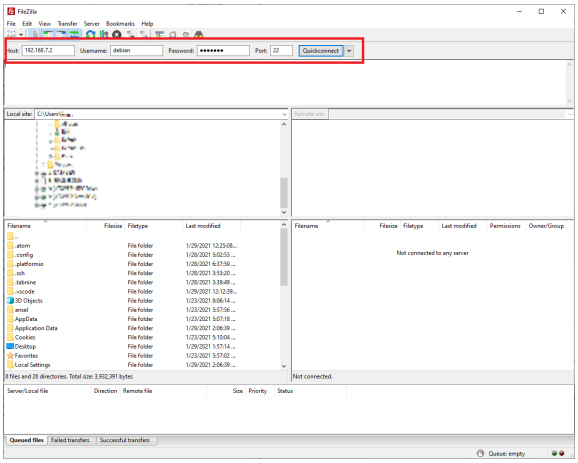

Figure 4.9: FileZilla Credentials Menu

4. Optional: If desired, save the credentials for quick future access.

5. There may be some popups about the security of the connection. This is due to how the BeagleBone handles certificates. They may be accepted.

6. If the connection is successful, a dual-explorer window will open. The left pane is the Windows Computer file system, the right pane is the BeagleBone file system. You may navigate the two file systems and move/copy files.

7. When finished, close the window.

SCP File Transfer between Linux/Mac and BeagleBone

Logging in through Ethernet

If the BeagleBone is instead powered externally and connected via Ethernet, follow the same instructions as logging in through USB, except use the Ethernet IP address of the BeagleBone for the Host Address.

4.3.5 Linux Basics

This guide is not designed to replace other Linux how-to guides and tutorials. The internet wbe the best resource for Linux information. It is important to have a basic understanding of Linux Operating System. In Linux, everything is a file. Directories (read "folders" for Windusers) are files which point to other files. Linux allows near-total control of the OperatSystem. Unlike Windows where certain files or folders are protected from users,

Command	Description
man <command>	Manual for Command
ls	list everything in current directory
cd <directory>	change directory
sudo <command>	Execute command as "super user"
mkdir <newDirectory>	Make new Directory
chmod <file><accessCode>	Change access permissions of file
mv <sourcefile><destination>	move (or rename) file
cp <sourcefile><destination>	copy file
rm <file>	Remove file
rm -r <DIR>	Recursively delete all directories and files starting from DIR
nano <file>	text editor

Linux trusts tthe user knows best and allows them to execute whatever they like assuming they have the proprivileges.

Privileges are separated into user, group, and everyone.

A Linux "root" user is a user with unlimited privileges. Root accounts are generally locked dto prevent unauthorized access. Because of this, Linux configurations usually include a "suusers" group which provides limited root access for users. This is useful so that files importto the operating system are not accidentally modified or deleted though users may still modifydelete these files if they login as root.

Basic Linux Commands

Particularly useful basic commands are listed below. This list is by no means exhaustive, but should be enough to explore the file system and continue with this guide. Sometimes commands must be used in conjunction with another command. For instance, when trying to remove a directory, the user may need to type:

```
sudo rm -r DirectoryToDelete
```

Also of note: Linux files and directories ARE case sensitive. The file "foo" is a different file than "Foo".

4.3.6 Configuring Linux

This portion of the guide will assume that you are starting from scratch.

If you have a pre-configured BeagleBone, you may skip the remainder of this chapter.

This portion of the guide also assumes that the BeagleBone is connected to the internet through the ethernet port. It is possible to perform a full offline installation of the software

by moving packages to the BeagleBone over SCP. However, that is beyond the scope of this guide.

Testing the System

Using an SSH client, like PuTTY, and a USB to computer connection, connect to the Beagle Bone Black SSH using IP 192.168.7.2 on port 22.

The new login uses the following credentials:

U: debian

P: temppwd

The availability if this connection may take longer than you might like, but be patient, the board will finish booting and enumerate as a drive on your host computer. Connect a live Internet connection by the Ethernet cable into your Beagle Bone Black. Check to see if you have a valid IP address on ‘eth0‘:

```
$ ifconfig
```

Version Check

If you are having troubles, be sure you are using the same version that's documented here. When the kernel changes, the results may be different.

Enter the following command to check the image version:

```
debian@beaglebone:~$ cat /etc/dogtag
BeagleBoard.org Debian Buster IoT Image 2020-04-06
```

Enter the following command to check the kernel version:

```
debian@beaglebone:~$ uname -a
Linux beaglebone 4.19.94-ti-r42 #1buster
SMP PREEMPT Tue Mar 31 19:38:29 UTC 2020 armv7l GNU/Linux
```

Disable Unused Hardware

Edit the uEnv.txt file and uncomment the some disable commands. This opens the pins up faccessing other functions.

```
sudo nano /boot/uEnv.txt
```

Uncommment the disable_uboot_overlay as follows:

```
#dtb_overlay=/lib/firmware/<file8>.dtbo
###
###Disable auto loading of virtual capes (emmc/video/wireless/adc)
#disable_uboot_overlay_emmc=1
```

```
disable_uboot_overlay_video=1
disable_uboot_overlay_audio=1
disable_uboot_overlay_wireless=1
disable_uboot_overlay_adc=1
###
```

Be sure to keep the emmc line commented, since that's the root file system.

Also, enable the universal overlay. This ensures the kernel has access to the hardware pmultiplexers.

```
###Additional custom capes
uboot_overlay_addr4=/lib/firmware/cape-universal.dtbo
#uboot_overlay_addr5=/lib/firmware/<file5>.dtbo
#uboot_overlay_addr6=/lib/firmware/<file6>.dtbo
#uboot_overlay_addr7=/lib/firmware/<file7>.dtbo
###
```

Finally, enable uio_pruss kernel module for the Programmable Realtime Unit (PRU).

```
###PRUSS OPTIONS
###pru_rproc (4.14.x-ti kernel)
#uboot_overlay_pru=/lib/firmware/AM335X-PRU-RPROC-4-14-TI-00A0.dtbo
###pru_rproc (4.19.x-ti kernel)
#uboot_overlay_pru=/lib/firmware/AM335X-PRU-RPROC-4-19-TI-00A0.dtbo
###pru_uio (4.14.x-ti, 4.19.x-ti & mainline/bone kernel)
uboot_overlay_pru=/lib/firmware/AM335X-PRU-UIO-00A0.dtbo
###
```

Save and reboot: 'sudo shutdown -r now'

Verify the uio_pruss kernel module is running:

```
lsmod | grep pru
```

Setup Pin Multiplexing

Write the following commands to get the CAN hardware to access the pins upon boot. Create a file in the /etc directory:

```
sudo nano /etc/pin_config.sh
```

Write the following into the directory:

```
#!/bin/sh -e
# DCAN1
config-pin p9.24 can
config-pin p9.26 can
```

```
# DCAN0
config-pin p9.19 can
config-pin p9.20 can
#ttyO2:
config-pin p9.21 uart
config-pin p9.22 uart
#ttyO4:
config-pin p9.11 uart
config-pin p9.13 uart
#ttyO5:
config-pin p8.37 uart
config-pin p8.38 uart
# PWMs
config-pin p8.46 pwm
config-pin p8.45 pwm
config-pin p8.34 pwm
config-pin p8.36 pwm
# GPIO
config-pin p9.12 gpio
config-pin p9.14 gpio

exit 0
```

Make the script executable:

```
sudo chmod +x /etc/pin_config.sh
```

However, these commands need to be run upon boot, so let's make a script to do this and it to a boot sequence.

```
sudo nano /lib/systemd/system/pin_config.service
```

Add this to the file:

```
[Unit]
Description=Setup for BBB

[Service]
Type=simple
ExecStart=/bin/bash /etc/pin_config.sh

[Install]
WantedBy=multi-user.target
```

Start the service

```
sudo systemctl start pin_config.service
```

Verify the service

```
sudo systemctl status pin_config.service
```

Enable the service at boot

```
sudo systemctl enable pin_config.service
```

To confirm the pin_config.service was enabled, look for a symbolic link in '/etc/systsystem'

```
debian@beaglebone:/etc/systemd/system$ ls -la
lrwxrwxrwx  1 root root   38 Sep 17 04:51 pin_config.service -> /lib/systemd/
pin_config.service
```

Reboot and verify. 'sudo shutdown -r now'

```
debian@beaglebone:~$ config-pin -q p9.24

Current mode for P9_24 is:      can
```

Verify the status of the pin_config.service was successful by looking for an output lthis:

```
debian@beaglebone:~$ sudo systemctl status pin_config.service
pin_config.service - Setup for BBB p
Loaded: loaded (/etc/systemd/system/pin_config.service; enabled; vendor prese
Active: inactive (dead) since Tue 2020-09-08 23:52:06 UTC; 2min 18s ago
Process: 856 ExecStart=/bin/bash /home/debian/pin_config.sh (code=exited, stat
Main PID: 856 (code=exited, status=0/SUCCESS)

Sep 21 14:06:18 beaglebone bash[2040]: Current mode for P9_13 is:
uart
Sep 21 14:06:18 beaglebone bash[2040]: Current mode for P8_37 is:
uart
Sep 21 14:06:18 beaglebone bash[2040]: Current mode for P8_38 is:
uart
Sep 21 14:06:18 beaglebone bash[2040]: Current mode for P8_46 is:
pwm
Sep 21 14:06:18 beaglebone bash[2040]: Current mode for P8_45 is:
pwm
Sep 21 14:06:18 beaglebone bash[2040]: Current mode for P8_34 is:
pwm
Sep 21 14:06:18 beaglebone bash[2040]: Current mode for P8_36 is:
pwm
Sep 21 14:06:18 beaglebone bash[2040]: Current mode for P9_12 is:
gpio
```

```
Sep 21 14:06:18 beaglebone bash[2040]: Current mode for P9_14 is:
gpio
Sep 21 14:06:18 beaglebone systemd[1]: pin_config.service: Succeeded.
```

If this doesn't work, be sure to disable the overlays in the uEnv.txt file.

Network Interface Configuration

Add Virtual CAN Drivers

Additional Software and Drivers

Truck Cape Projects

J1708 Drivers

Writing Current Linux Image to SD Card

4.4 CAN on BeagleBone

4.4.1 CAN Channels

The BeagleBone is generally configured to have multiple CAN Channels. Usually, these channels are can0, can1, vcan0, and vcan1. VCan is a "virtual CAN bus" that to the BeagleBone appears to be a normal CAN channel. However, this CAN channel does not send messages on any physical bus. Only the BeagleBone hosting the virtual CAN network can send or receive messages. Because of this "loopback" functionality, the virual CAN channels are often used for device and application testing.

Baudrate

The term baudrate is generally interchangeable with the term bitrate. Either way, it refers to the speed of the bus in bits-per-second. The BeagleBone uses 250k baud (250,000 bits per second) by default. To set the baudrate, use the following commands, replacing <channel>with the channel name and <baudrate>with the desired baudrate (usually 125kbps, 250kbps, 500kbps, or 1000kbps). Below is the general syntax for changing the baudrate of a CAN channel on the BeagleBone. If you are having connectivity issues, you may have the channel set at the incorrect baudrate.

```
$ sudo ip link set <channel> down
$ sudo ip link set <channel> up type can bitrate <baudrate>
```

An example is shown below

```
$ sudo ip link set can0 down
$ sudo ip link set can0 up type can bitrate 500000
```

4.4.2 Linux CAN-Utils

The easiest way to read CAN messages from an active bus is through the use of the Linux CAN-Utils package. This software suite includes programs for reading, sending, and generating CAN messages. This guide assumes a basic understanding of the CAN bus and J1939. The CAN-utils suite includes many different functions but this guide will explain only the most basic functions.

Reading CAN Messages with CAN-Utils

There are two primary functions for reading CAN messages in CAN utils: candump and cansniffer. The cansniffer program provides more visual information, but only does so for standard CAN frames. Cansniffer does not work for extended CAN frames with 29 bit arbitration IDs commonly used in heavy vehicles. Thus, this guide will focus on the 'candump' utility.

CANDump

CANDump listens for CAN messages on the channel sockets. When a message is received, it is printed to the console. The only necessary candump argument is the CAN channel. Syntax examples are shown below. Replacing <channel>with the desired CAN channel:

```
$ candump <channel>
$ candump can0
$ candump can1
$ candump vcan0
$ candump vcan1
```

A useful channel argument is "any". It will listen on all CAN channels instead of just one. Its syntax is shown below:

```
$ candump any
```

CANDump has a variety of different modes and printing options. Refer to the MAN Page for specific information.

To stop candump, use ctrl+c.

Using CANDump to Create CAN Logs

CANDump may be used to create logs of CAN Traffic. To do so, save the command output to a text or log file. The syntax and an example are shown below

```
$ candump <channel> <optional arguments> -> <filename>
$ candump any -> logfile.txt
```

This example will save any CAN message received on any channel to the file "logfile.txt" until the user stops the candump.

Sending CAN Messages with CAN-Utils

There are two primary commands for sending CAN messages. CANSend is used for specific individual messages, while CANGen is used to generate random CAN traffic.

CANSend

CANSend requires a CAN channel and a message. The message is the arbitration ID followed a pound sign followed by the data field. Arbitration ID and Data bytes are represented by a hexadecimal string. Basic syntax and examples are shown below:

```
$ cansend <channel> <can id>#<data>
$ cansend vcan0 123#DEADBEEF
$ cansend vcan1  1F334455#1122334455667788
```

For more cansend documentation, review the cansend MAN Page.

CANGen

CANGen may be used to generate random CAN traffic. Using command line arguments, aspects of the message may be selected rather than randomized. For instance, you could generate random CAN messages with a specific pre-determined Arbitration ID. Basic syntax and examples are shown below:

```
$ cangen <channel> <optional arguments>
$ cangen vcan0
$ cangen vcan0 -g 4 -I 42A -L 1 -D i -v -v
```

The third example listed is for a CAN message with fixed CAN ID and data length.

For further cangen documentation, review the cangen MAN Page.

4.4.3 SocketCAN

Descending into lower levels, SocketCAN is a set of open-source CAN drivers for the Linux Kernel. SocketCAN takes received CAN messages and makes them usable for the operating system by creating a network socket. "Sockets" are part of an API beyond the scope of this subsection. Essentially, incoming CAN messages are processed by the driver and fed into the socket. Applications may listen to this socket to receive the CAN Traffic. CAN-Utils is built using the SocketCAN sockets.

SocketCAN for Python

It may be useful to create a Python Script which can send and receive CAN. Python Libraries for SocketCAN are housed in the python-can package. Examples sourced from the python-can pypi.org page

Installing python-can

```
$ python3 pip install python-can
```

Sending and Receiving CAN with python-can

Below is example code for sending CAN. The important steps are connecting to the Socket-CAN channel, **can.interface.Bus()**, defining the message, **can.Message()**, and sending the message, **bus.send(msg)**.

```
# import the library
import can

# create a bus instance
# many other interfaces are supported as well (see below)
bus = can.Bus(interface='socketcan',
              channel='vcan0',
              receive_own_messages=True)

# send a message
message = can.Message(arbitration_id=123, is_extended_id=True,
                      data=[0x11, 0x22, 0x33])
bus.send(message, timeout=0.2)

# iterate over received messages
for msg in bus:
    print("{X}: {}".format(msg.arbitration_id, msg.data))

# or use an asynchronous notifier
notifier = can.Notifier(bus, [can.Logger("recorded.log"), can.Printer()])
```

Review online python-can documentation for more functions and examples.

Below is example code for listening to the CAN socket and printing the incoming messages.

SocketCAN for C

4.5 J1708 Serial on BeagleBone

4.6 PLC on BeagleBone

4.7 Development Tool Chain

New tools and utilities may be developed for the BeagleBone in many different programming languages. Although possible, code writing and programming need not be performed on the BeagleBone itself. External IDEs may be configured to cross-compile for the BeagleBone. For example, code could be written in the CLion IDE on Windows 10, but compiled and run on the BeagleBone through a remote connection.

4.7.1 Cross Compiling in C

C Code can be cross compiled for the BeagleBone using a variety of IDEs. This guide will describe how to set up remote cross compile on a few different IDEs.

C Cross Compiling in C Lion

CLion is an IDE for C from JetBrains. It has a built-in utility for cross-compiling via remote access devices. This subsection of the guide concerns CLion and BeagleBone configruation for cross compilation

1. Install C Lion. It requires a license, but thankfully JetBrains provides Free Educational Licenses for C Lion and other IDEs.

2. After installation, navigate to File Settings Build, Execution, Deployment Toolchains.

3. Add a new toolchain using the plus button at the top left and select "Remote Host" (Fig. 4.10).

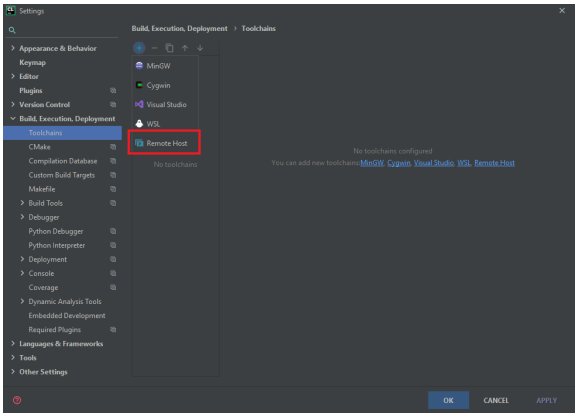

Figure 4.10: CLion Remote Toolchain Option

4. Add new device credentials using the three dots to the right of the credentials field. (Fig. 4.11).

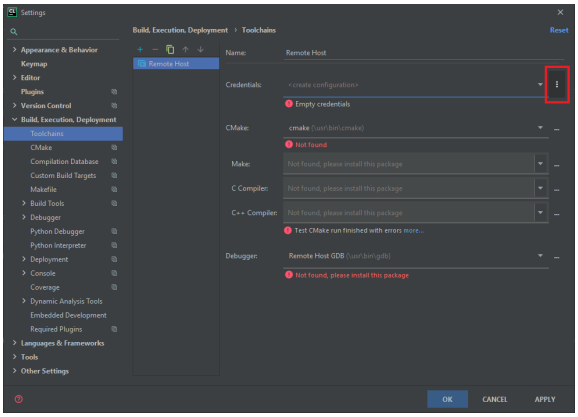

Figure 4.11: CLion Remote Toolchain Credentials Button

5. Add device credentials just like PuTTY (Fig. 4.12).

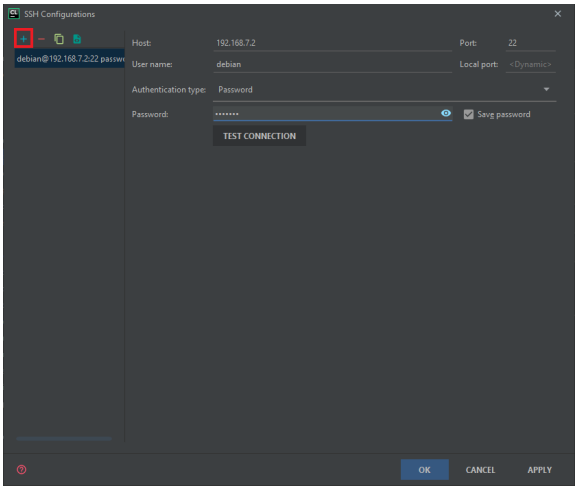

Figure 4.12: CLion Remote Toolchain Credentials Screen

6. Test connection with the "Test Connection" button. If successful, click "Apply" then "OK."

7. Now back on the Toolchains page, the credential field should say "Connected." However, the other fields may show errors. CMake as well as the C compiler must be installed on the beaglebone

   (a) Connect to the BeagleBone using PuTTY. To install these packages, the BeagleBone should be connected to the internet. Although there are offline methods of installation, this guide will assume the BeagleBone is connected to the internet.

   (b) For installing CMake and GNU DeBugger (gdb), run the following commands:

```
$ sudo apt-get update -y
$ sudo apt-get update
$ sudo apt-get install cmake -y
$ sudo apt-get install gdb -y
```

8. Assuming the previous installation steps were successful, the Toolchain menu should now be connected to the BeagleBone and recognize that cmake and Remote Host GDB are installed. Click "Apply" and then "OK".

C Cross Compiling in Atom

C Cross Compiling in Sublime Text

C Cross Compiling in Visual Studio Code

4.7.2 Remote Python Environment

The BeagleBone is configured with both Python 2 and Python 3. Text editors like Nano, VIM, and Emacs may be installed and used for writing Python scripts. However, the user may prefer to write scripts on an external device like a laptop. This portion of the guide covers Remote Python Development

Remote Python Environment using Jupyter Notebook

Remote Python Environment for PyCharm

PyCharm Professional is required for configuring Remote Python Interpreters. This feature is not currently supported for PyCharm Community Edition. Like CLion, JetBrains offers free educational licenses for PyCharm Professional.

1. Navigate to File, Settings, Project:, then "Python Interpreter."

2. Select the button with three dots at the top right side of the screen (Fig. 4.13). Choose "Add..."

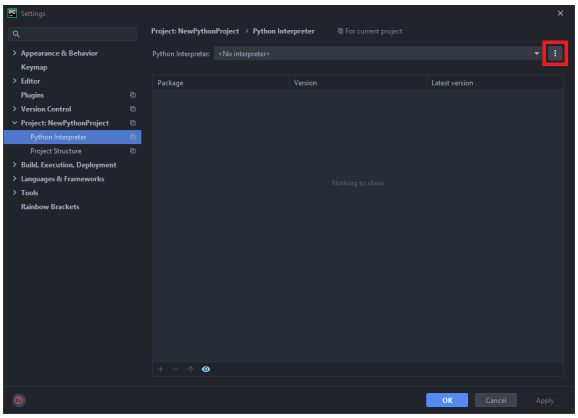

Figure 4.13: Add Interpreter to PyCharm

3. Select "SSH Interpreter"

4. Enter credentials just like other SSH utilities in the guide (Fig. 4.14)

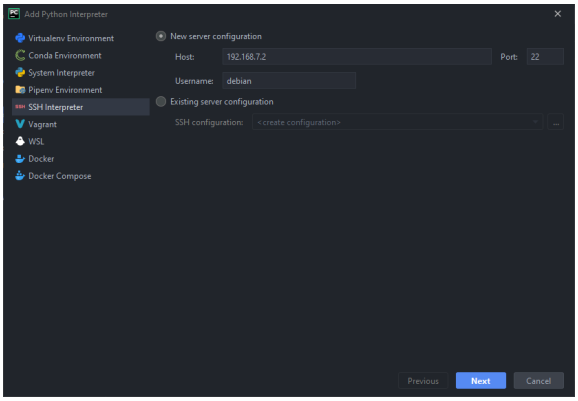

Figure 4.14: PyCharm SSH Credentials

5. Continue the setup pages, entering credentials when necessary.
6. Choose where you want PyCharm to place files and which BeagleBone Python Interpreter to use.
7. Once the setup pages have finished, select "Apply" and then "OK"
8. The current PyCharm Project should now be configured to use the BeagleBone Python Interpreter.

Remote Python Environment for Atom

Remote Python Environment for Sublime Text

Remote Python Environment for Visual Studio Code

4.8 Programmable Realtime Units (PRU)

Chapter 5

Introduction

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut rhoncus nunc eu neque interdum placerat. Suspendisse vestibulum hendrerit vehicula. Etiam blandit eleifend pellentesque. Morbi in risus nec mi ultrices vehicula. Praesent porta dui non magna porttitor, et placerat arcu varius. Suspendisse a ullamcorper lectus, a porta mauris. Nulla et luctus mauris.

Etiam congue ullamcorper gravida. Mauris sapien lectus, iaculis vel luctus in, aliquam a urna. Quisque venenatis sollicitudin risus sed ultrices. Curabitur sit amet diam condimentum, vehicula mauris at, imperdiet leo. Curabitur quis posuere odio, vel interdum lectus. Proin a ligula vitae quam lobortis dignissim at quis risus. Aenean eros diam, dignissim sed dapibus eget, viverra tincidunt arcu. Phasellus vitae quam malesuada, eleifend lorem non, cursus urna. Morbi non nulla vel diam ullamcorper eleifend ac at lorem. Donec sed aliquet mauris. Nullam euismod aliquet odio eu iaculis. Nullam non auctor urna. Donec ipsum enim, interdum et lectus sed, consectetur condimentum ligula. In pharetra pretium vestibulum. Donec ut nisl a urna aliquam iaculis.

Appendix A

Appendix Title