swapswap

King, Lawrence

[Email address]

Abstract

Modern Single board computers are capable of running high level software and communication with data servers on the network. Additionally these single board computers need to communicate with locally connected devices. The locally connected devices can be a simple as a push button, or as complex as a micro-controller connected to multiple peripherals. This in this training course the students will learn how to install the Debian Linux operating system onto the Snapdragon 410c single board computer and then how to modify the operating system to communicate with locally attached peripherals.

Hands On Training

For DragonBoard 410c

Contents

[Introduction 3](#_Toc474312869)

[Equipment required 3](#_Toc474312870)

[Equipment Setup 3](#_Toc474312871)

[Installing Debian Operating System 3](#_Toc474312872)

[Download image 3](#_Toc474312873)

[Copy to SDCard 4](#_Toc474312874)

[Boot the 410c from the SDCard 4](#_Toc474312875)

[Initialize the 410c with the Debian Operating System. 4](#_Toc474312876)

[Connecting to 410c console 4](#_Toc474312877)

[Set the system name and change the default password 5](#_Toc474312878)

[Setup development enviroment 5](#_Toc474312879)

[Installing necessary development software 6](#_Toc474312880)

[Installing GPS (build 196 and beyond only) 8](#_Toc474312881)

[Rebuilding the Debian Operating System 8](#_Toc474312882)

[Getting the Kernel Source code 8](#_Toc474312883)

[Checkout the Required Version of the Operating System 8](#_Toc474312884)

[Build a Boot Image 9](#_Toc474312885)

[Installing the Boot Image 10](#_Toc474312886)

[Making changes to the operating system 10](#_Toc474312887)

[Adding Device Drivers to the Operating System 10](#_Toc474312888)

[Describing how the new devices are attached 11](#_Toc474312889)

[Experiments 12](#_Toc474312890)

[Blinky LED (Linker Kit) 12](#_Toc474312891)

[Pushbutton 14](#_Toc474312892)

[SPI based sliding potentiometer (Linker Kit) 14](#_Toc474312893)

[Modify the Kernel 15](#_Toc474312894)

[Hardware Setup 16](#_Toc474312895)

[Build the Application 17](#_Toc474312896)

[Accessing the slider from a Python program 17](#_Toc474312897)

[Programming the Arduino (native development) 18](#_Toc474312898)

[Power Measurement with INA219 18](#_Toc474312899)

[INA219 Breakout Board 18](#_Toc474312900)

[Monitoring CPU Speed and Temperature 21](#_Toc474312901)

[DS1307 Real Time Clock 22](#_Toc474312902)

[Resources 24](#_Toc474312903)

[TurtleBot Installation 25](#_Toc474312904)

[Hardware setup 25](#_Toc474312905)

[Install Ubuntu software onto a Win10 PC 26](#_Toc474312906)

[Enable Ubuntu under Win10 26](#_Toc474312907)

[Install an X-server 27](#_Toc474312908)

[Install ROS Software onto the Host computer 28](#_Toc474312909)

[Build the turtlebot software for the host PC 28](#_Toc474312910)

[Install ROS Software onto the 410c 28](#_Toc474312911)

[Check that you are connected to the kobukibase 30](#_Toc474312912)

[Upgrade the Astra Camera firmware 31](#_Toc474312913)

[Check that the camera works 31](#_Toc474312914)

[Connect the Host to the Robot over the Network 31](#_Toc474312915)

[Test the connection 31](#_Toc474312916)

[Make the Turtlebot Move 32](#_Toc474312917)

[Changing Follower Parameters 32](#_Toc474312918)

[Adding an Arduino powered Arm to the Turtlebot 32](#_Toc474312919)

[ROS Resources 32](#_Toc474312920)

[Installing Amazon Alexa on the 410c 32](#_Toc474312921)

# Introduction

Modern Single board computers are capable of running high level software and communication with data servers on the network. Additionally these single board computers need to communicate with locally connected devices. The locally connected devices can be a simple as a push button, or as complex as a micro-controller connected to multiple peripherals. This in this training course the students will learn how to install the Debian Linux operating system onto the Snapdragon 410c single board computer and then how to modify the operating system to communicate with locally attached peripherals.

## Equipment required

DragonBoard 410c and power supply.

Monitor – Lilliput FA1012-NP/C/T

Keyboard and Mouse – Logitech K400r

Two micro SDCards – 8GB or larger

* One for installing the operating system
* One for the swap space and file system.

Linker Kit – LinkSprite kit from Arrow Electronics.

Laptop - provided by student. In this course we will show examples on a Win7 laptop, but other versions of windows and operating system can be made to work.

USB cable

## Equipment Setup

connect display

connect keyboard

connect power

## Installing Debian Operating System

This section outline how to download and install the Debian operating system onto the 410c. Since these steps can take some time, there will also be “shortcut” instructions throughout the document to save time during the training course.

### Download image

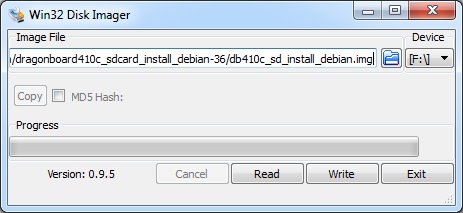
The Debian Linux image for the 410c board is available on [www.96Boards.org](http://www.96Boards.org) . Click on this link to download the image to your local laptop. <https://www.96boards.org/products/ce/dragonboard410c/downloads/>

After the file has downloaded you should have the file dragonboard410c\_sdcard\_install\_debian-36.zip unzip this file and you should then see the file dragonboard410c\_sdcard\_install\_debian-36.img. The .img file will be used in the next step.

### Copy to SDCard

Once the .img file has downloaded, it needs to be copied to a SDcard for transfer to the 410c. The image is not a file that can be copied to a simple FAT file system, it is a binary image of the complete contents of the boot file system for the 410c. It will be necessary to use a SDCard Image program, in this case we will use Win32DiskImager.exe which is available from sourceforge.net at this link: <https://sourceforge.net/projects/win32diskimager/> Download and install Win32DiskImager on your laptop.

Use Win32DiskImager to copy the image to the micro-SDCard by clicking the Write button.



### Boot the 410c from the SDCard

* Change dip switch 2 to ON
* Insert the SDCard
* Connect Power
* Note: the SDCard bootloader will not work if there is a mezzanine board attached to the low-speed connector.

### Initialize the 410c with the Debian Operating System.

* Copy Image to eMMC
* Remove SDCard (install the swap/file-system SDcard at this time).
* Reboot

### Connecting to 410c console

There are three possible ways to access a development command window on the 410c Dragonboard:

* Direct with monitor and keyboard
* USB cable to uart adapter – console
* SSH via WiFi or wired Ethernet (part of Cygwin)
  + Start a console on the terminal using either of the methods above
  + Connect to Wifi network (or wired Ethernet network using a USB to Ethernet adapter)
  + On the 410c run the command ‘ip addr’
  + The output of ip addr will contain an Ethernet address, something like 192.168.0.107
  + In the laptop run the command ‘ssh linaro@192.168.0.170’, and login with the password ‘linaro’.
  + Note: ssh is part of Cygwin, and is available in other packages such as Windows Subsystem for Linux (WSL).

## Set the system name and change the default password

Change the name of the system from the default “linaro-alip” to something better suited to your needs. This is important if you have multiple 410c DragonBoards and want to be able to easily distinguish them.

sudo sed -i '1s/linaro-alip/turtlebot/' /etc/hostname

sudo sed -i 's/linaro-alip/turtlebot/' /etc/hosts

passwd

sudo reboot now

## Setup development enviroment

* Insert a blank SDCard into the SD Card slot then partition and mount it.
* Execute the following commands (enter the answers in red):

sudo gdisk /dev/mmcblk1

Command (? for help): o

This option deletes all partitions and creates a new protective MBR.

Proceed? (Y/N): y

Command (? for help): n

Partition number (1-128, default 1):

First sector (34-123764702, default = 2048) or {+-}size{KMGTP}:

Last sector (2048-123764702, default = 123764702) or {+-}size{KMGTP}: 8G

Current type is 'Linux filesystem'

Hex code or GUID (L to show codes, Enter = 8300): 8200

Changed type of partition to 'Linux swap'

Command (? for help): n

Partition number (1-128, default 2):

First sector (34-123764702, default = 16779264) or {+-}size{KMGTP}:

Last sector (2048-123764702, default = 123764702) or {+-}size{KMGTP}:

Current type is 'Linux filesystem'

Hex code or GUID (L to show codes, Enter = 8300):

Command (? for help): w

Final checks complete. About to write GPT data. THIS WILL OVERWRITE EXISTING PARTITIONS!!

Do you want to proceed? (Y/N): y

* Reboot and run the following commands

sudo reboot now

sudo mkfs –t ext4 /dev/mmcblk1p2

sudo mkswap /dev/mmcblk1p1

* Alternate way to create a 1G swapfile

sudo dd if=/dev/zero of=/swapfile bs=1M count=1024

sudo chmod 600 /swapfile

sudo mkswap /swapfile

sudo swapon /swapfile

sudo sed -i '$a /swapfile none swap sw 0 0\n' /etc/fstab

* To update /etc/fstab run the following command:

sudo sed -i '$a /dev/mmcblk1p1 none swap sw 0 0\n' /etc/fstab

* Reboot and run the following command

# ensure swap is mounted

sudo reboot now

free

## Installing necessary development software

The base system requires some additional packages for in order to rebuild the operating system kernel, and to perform all of the experiments later in this document. In this section we will install upgrade the system to the latest software and install the necessary tools (about 40 mins). To make debugging and learning easier we will also install the manual pages so that you can use the man and apropos commands.

sudo apt-get update

sudo apt-get upgrade -y

sudo apt-get dist-upgrade –u -y

sudo apt-get install man-db manpages manpages-dev -y

sudo apt-get install autoconf automake libtool -y

sudo apt-get install libncurses-dev libfdt-dev -y

sudo apt-get install device-tree-compiler stress -y

sudo apt-get install libpython-dev libpython3-dev -y

sudo apt-get install arduino-mk arduino git build-essential -y

sudo apt-get install swig3.0 python-dev python-pip -y

sudo apt-get install nodejs-dev cmake pkg-config libpcre3-dev –y

sudo apt-get install libmraa-dev libsoc-dev -y

sudo apt-get clean

sudo apt-get autoremove

sudo adduser linaro i2c

git clone <https://github.com/96boards/96boards-tools>

sudo cp 96boards-tools/70-96boards-common.rules \

/etc/udev/rules.d/

sudo cp /etc/profile.d/96boards-sensors.sh \ /etc/X11/Xsession.d/96boards-sensors

In your .bashrc add the following lines at the end of the file

export PYTHONPATH=\

/usr/local/lib/python3.5/dist-packages:\

/usr/local/lib/python2.7/site-packages

export ARCH=arm64

export PATH=$PATH:/usr/sbin

export MONITOR\_PORT=/dev/tty96B0

export JAVA\_TOOL\_OPTIONS="-Dgnu.io.rxtx.SerialPorts=/dev/tty96B0"

Then run the following command to update the environment in the current shell.

source ~/.bashrc

### Installing GPS (build 196 and beyond only)

Run the following commands to enable GPS under Linux on the board.

sudo apt-get install gpsd-clients gnss-gpsd -y

sudo systemctl start qdsp-start.service

After GPS packages are installed you can see GPS position with and gpsd clients such as gpsmon or xgps

## Rebuilding the Debian Operating System

The default Debian operatoing system operates most of the basic features on the board, occasionally when adding additional peripherals it may become necessary to make changes to the operating system to accommodate the new devices. This section give a recipie on how to rebuild the operating system. In later experiment chapters it may be necessary to rebuild the operating system and these instructions pave the way to making changes.

### Getting the Kernel Source code

Note: The ‘git clone’ operation below takes about 35 minutes (depending on your network speeds) and downloads a little over 1GB of data.

See <http://builds.96boards.org/releases/dragonboard410c/linaro/debian/latest/> for detailed instructions, or follow the simplified instructions here.

The DragonBoard 410c is an ARMv8 platform, and the kernel is compiled for the Aarch64 target. It is possible to build natively, on the target board, in this case on the 410c. To build the Linux kernel, you can use the following instructions [do-git-clone]:

ln –s /media/linaro/<long name>/workspace workspace

cd workspace

sudo mkdir Debian

sudo chown linaro.linaro Debian

cd Debian

git clone -n \

http://git.linaro.org/landing-teams/working/qualcomm/kernel.git

Note: ensure that you have added a swap space on the SDCard (see instructions above), otherwise the git clone will fail. (just over 1GB of swap is used while resolving deltas)

### Checkout the Required Version of the Operating System

Building the Linux kernel and the modules from source (about 45 minutes). The git checkout command takes two parameters, –b option specifies the build you want to checkout, typically a release number like kernel-16.02, or a daily build number like build-80. The second parameter is either a tag (releases have tags) like debian-qcom-dragonboard410c-16.02 or a ‘Kernel version’ number, you can find the ‘Kernel version’ number for a specific daily build in the Build description notes in the daily build directory (usually [www.builds.linaro.com/snapshots/dragonboard410c/linaro/debian/latest](http://www.builds.linaro.com/snapshots/dragonboard410c/linaro/debian/latest)). [../do-checkout] [../do-make-defconfig] [../do-make-kernel]

cd kernel

#git checkout -b kernel-16.02 debian-qcom-dragonboard410c-16.02

#git checkout -b build-190 [4c23d139dbab35c183d94ea69b339c1b4c9fd14c](https://git.linaro.org/landing-teams/working/qualcomm/kernel.git/commit/4c23d139dbab35c183d94ea69b339c1b4c9fd14c)

git checkout -b kernel-17.04 debian-qcom-dragonboard410c-17.04

make defconfig distro.config

make -j4 Image dtbs modules KERNELRELEASE=`make kernelversion`-linaro-lt-qcom

Note: ensure that you have added a swap space on the SDCard (see instructions above), otherwise the make will fail. (~100MB of swap gets used while compiling the kernel)

While the kernel is compiling, please read-ahead in this document and select which hardware experiment you would like to try. Prepare the hardware and understand what changes you will need to make to the kernel and the device tree.

### Build a Boot Image

Once you have completed building the kernel, you need to create a valid boot image containing your kernel build. You need to install the skales tools, you also need a ramdisk image, you can use the one from the release [../do-get-skales]:

git clone git://codeaurora.org/quic/kernel/skales

wget \ http://builds.96boards.org/snapshots/dragonboard410c/linaro/debian/initrd.img-`make kernelversion`-linaro-lt-qcom

The boot image consists of the table of device tree (dt.img), the kernel image (Image) and an init ramdisk image. The dtbTool (device tree blob) is a standalone application that will process the DTBs generated during the kernel build, to create the table of device tree image. This tool is included in the skales git tree cloned [../do-make-device-tree]:

./skales/dtbTool -o dt.img -s 2048 arch/arm64/boot/dts/qcom/

The tool mkbootimg (also in the skales git tree previously cloned) is a standalone application that will process all files and create the boot image that can then be flashed into the on-board eMMC. The boot image also contains the kernel bootargs, which can be changed as needed in the next command [../do-make-boot-image]:

./skales/mkbootimg --kernel arch/arm64/boot/Image \

--ramdisk initrd.img-4.4.0-linaro-lt-qcom \

--output boot-db410c.img \

--dt dt.img \

--pagesize 2048 \

--base 0x80000000 \

--cmdline "root=/dev/disk/by-partlabel/rootfs rw rootwait console=ttyMSM0,115200n8"

### Installing the Boot Image

Finally you can install the Boot Image and Modules that you have built in the sections above [../do-install]:

sudo make INSTALL\_MOD\_STRIP=1 modules\_install KERNELRELEASE=`make kernelversion`-linaro-lt-qcom

sudo dd if=boot-db410c.img of=/dev/mmcblk0p8

Then reboot the system to run your new version of the kernel and device-tree.

sudo reboot now

If you elected to rebuild the operating system kernel on your laptop, you will want to copy the image to the 410c. You can do this with scp. First determine the IP address of your 410c. On the 410c run the command:

ip addr

in section ‘3: wlan0’ of the output you will find an IP address like ‘10.76.70.117’. On your laptop run the command:

scp boot-410c.img [linaro@10.76.70.117:boot-410c.img](mailto:linaro@10.76.70.117:boot-410c.img)

and provide the default linaro password which is ‘linaro’, this will copy the boot image file over to the home directory where it can be installed using the dd command as described above.

# Making changes to the operating system

Once you have demonstrated that you can rebuild the standard OS kernel, you can make changes. To add additional hardware to the 410c board we need to perform two basic steps, 1) add the necessary device drivers to the operating system kernel (kernel config), and 2) describe to the operating system where the new devices are attached (device tree).

## Adding Device Drivers to the Operating System

In this example we will add support for an INA219 power monitor chip, the basic steps are the same for other chips. Execute the following extra steps in the procedure above:

cd ~/workspace/Debian/kernel

make menuconfig

Select “Device Drivers” then <Enter>

Select “I2C Support” followed by <Enter>

Select “I2C Support” Followed by ‘Y’ then <esc><esc> to move back up one level.

Select “Hardware Monitoring support” then ‘Y’ followed by <Enter>

Select “Texas Instruments INA219 and compatibles” then ‘Y’

Back out of the menus with <esc><esc> and at the end select ‘Yes’ to save the changes.

## Describing how the new devices are attached

The device tree gives a description of the devices attached to the system bus, this includes the devices internal to the SOC. In general we need to describe the devices attached to the external busses, the existing files already describe the devices on the 410c board, and give convenient names to the various busses that are available on the Low-speed and high speed connectors of the 410c board.

These links gives some background information on what the device tree is:

<http://elinux.org/Device_Tree>

<http://www.devicetree.org/Main_Page>

<http://www.devicetree.org/Device_Tree_Usage>

Our task is to describe the devices that are being added to the 410c board when connecting a mezzanine board and a device. This additional connectivity information will need to be added in the file arch/arm64/boot/dts/qcom/apq8016-sbc.dtsi,and then compiled into a binary blob that can be merged with the updated kernel to form the new boot image.

In this example we are adding an ina219 device and it will be connected to the i2c0 bus on the low speed connector (LS-I2C0).

Edit the file arch/arm64/boot/dts/qcom/apq8016-sbc.dtsi and add the lines marked with ‘+’ as shown (do not include the ‘+’):

i2c@78b9000 {

           /\* On Low speed Expansion \*/

Label = “LS-I2C0”

status = "okay";

+               ina219@40 {

+                       compatible = "ti,ina219";

+                       reg = <0x40>;

+                       shunt-resistor = <100000>;

+               };

           };

Example device tree entries for various devices can be found under /usr/src/Debian/kernel/Documentation/devicetree/bindings. To find the device tree info for a specific peripheral try the following command:

grep –ri ina219 Documentation/devicetree/

The path from this command will also help you locate the kernel driver to select in the menuconfig step in the previous section.

After you have made changes to the kernel configuration and the device tree, you can use the scripts to recompile and install your new kernel, the rebuild takes about 5 minutes:

../do-rebuild-kernel

../do-install

sudo reboot now

# Using the MIPI-CSI Camera

## Introduction

Only basic use cases are supported at the moment. More features will be added, such as support for scaling, cropping.

The release is configured with no camera, and users with camera are expected to configure the DTS file accordingly. Please check commit “dts: Disable camera sensors in dtsi” in the kernel, it should mostly be a matter of reverting this patch. Note that this patch assumes that 2 OV5645 sensors are connected on each CSI port. The setup is validated using an upcoming STM based mezzanine (STM32F446 sensor board).

cd ~/workspace/kernel

git log

search for “dts: Disable camera sensors in dtsi” and copy the commit number (commit bffd64e24b45824e495532b0521cb5c5568c6ee0)

git show bffd64e24b45824e495532b0521cb5c5568c6ee0| patch -p 1 -R

After applying the patch, you will need to rebuild and uinstall the patched kernel (instructions above), then reboot with the new kernel. This would also be a good time to turn on spidev (instructions below).

The 17.04 release includes drivers for:

* OV5645 camera sensor;
* QC MSM camera sub-system (CSIPHY, CSID, ISPIF, VFE);
* QC Camera control interface.

The OV5645 camera sensor driver is in a final stage of implementation and is currently undergoing review in upstream lists.

The CAMSS (camera sub-system) driver is supports direct dump to memory and format conversion (UYVY to NV12). The version which supports direct dump to memory only was published on linux-media, the format conversion will follow. The CAMSS HW on 8016 consists of two CSIPHY modules, two CSID modules, ISPIF module and VFE module. The driver is implemented to configure each of these HW modules. It is a V4L2 driver which also utilizes the media controller framework to model the internal topology of the system. The driver registers V4L2 subdevice nodes for each of the CSIPHY and CSID modules, then registers two V4L2 subdevice nodes to represent the ISPIF module. Then it registers three V4L2 subdevice nodes to represent each of the RDI channels in the VFE and another V4L2 subdevice node for the processing part of the VFE.

The CCI (camera control interface) driver is a version which originates from QC Android driver and is now separated from the CAMSS driver and compiled on our linux. For this another V4L2 driver and media device are created – this is only a temporary work to enable control on the camera sensor. Proper implementation will follow.

## Basic usage

Make sure that you have the following package installed:

sudo apt-get install v4l-utils

To ensure that your sensor is properly connected, you can inspect the output of the following command:

sudo media-ctl -d /dev/media1 -p

If everything is ok, you should see something like this:

entity 13: ov5645 1-0076 (1 pad, 1 link)

type V4L2 subdev subtype Unknown flags 0

device node name /dev/v4l-subdev9

pad0: Source

[fmt:UYVY8\_2X8/1920x1080 field:none

crop:(0,0)/0x0]

-> "msm\_csiphy0":0 [ENABLED,IMMUTABLE]

At this point you need to configure the pipeline: link CSIPHY to CSID, CSID to ISPIF, ISPIF to VFE. Then configure formats on all entities in the pipeline. For direct dump to memory (RDI channels) this looks like this:

sudo media-ctl -d /dev/media1 -l '"msm\_csiphy0":\

1->"msm\_csid0":0[1],"msm\_csid0":\

1->"msm\_ispif0":0[1],"msm\_ispif0":\

1->"msm\_vfe0\_rdi0":0[1]'

sudo media-ctl -d /dev/media1 -V \

'"ov5645 1-0076":0[fmt:UYVY8\_2X8/1920x1080 field:none],"msm\_csiphy0":0[fmt:UYVY8\_2X8/1920x1080 field:none],"msm\_csid0":0[fmt:UYVY8\_2X8/1920x1080 field:none],"msm\_ispif0":0[fmt:UYVY8\_2X8/1920x1080 field:none],"msm\_vfe0\_rdi0":0[fmt:UYVY8\_2X8/1920x1080 field:none]'

At this point, the pipeline should be configured and ready to be used by any application that can use v4l2. For example, you can use Gstreamer to take a JPEG picture:

gst-launch-1.0 v4l2src device=/dev/video0 num-buffers=1 ! 'video/x-raw,format=UYVY,width=1920,height=1080,framerate=30/1' ! jpegenc ! filesink location=image01.jpg

Or you can use Gstreamer to show a live preview from the camera:

gst-launch-1.0 v4l2src ! glimagesink

Pipeline configuration for the format conversion looks like this:

sudo media-ctl -d /dev/media1 -l '"msm\_csiphy0":1->"msm\_csid0":0[1],"msm\_csid0":1->"msm\_ispif0":0[1],"msm\_ispif0":1->"msm\_vfe0\_pix":0[1]'

sudo media-ctl -d /dev/media1 -V '"ov5645 1-0076":0[fmt:UYVY8\_2X8/1920x1080 field:none],"msm\_csiphy0":0[fmt:UYVY8\_2X8/1920x1080 field:none],"msm\_csid0":0[fmt:UYVY8\_2X8/1920x1080 field:none],"msm\_ispif0":0[fmt:UYVY8\_2X8/1920x1080 field:none],"msm\_vfe0\_pix":0[fmt:UYVY8\_2X8/1920x1080 field:none]'

And similar Gstreamer pipeline for a JPEG picture:

gst-launch-1.0 v4l2src device=/dev/video3 num-buffers=1 ! 'video/x-raw,format=NV12,width=1920,height=1080,framerate=30/1' ! jpegenc ! filesink location=image02.jpg

If you have a second camera sensor and intend to use it concurrently then link and configure another pipeline which includes the second camera and the unused entities. Use a v4l2 application the same way only pointing the correct video device node used in this pipeline.

## Video record pipeline

Starting with this release a basic video recording GStreamer pipeline is supported involving the camera and video encoder. Currently this has the following limitations:

The frame data must be vertically aligned to 32 lines. For the OV5645 camera this means that only 1280×960 resolution is supported.

A GStreamer video encoder plugin is needed which is not a part of the release. To enable this the user must patch and rebuild gstreamer1.0-plugins-good package – this is recommended for advanced users only.

To enable the video encoder plugin follow these steps. Install needed packages and dependencies:

sudo apt-get install build-essential fakeroot devscripts quilt

sudo apt-get build-dep gstreamer1.0-plugins-good

Get gstreamer1.0-plugins-good source:

apt-get source gstreamer1.0-plugins-good

cd gst-plugins-good1.0-1.10.4/

Get the video encoder plugin patches. These are the 12 last commits from here. Download them as patch files and to apply them do the following for each patch:

quilt import path\_to\_patch.patch

quilt push

Edit debian/rules and add --enable-v4l2-probe and --without-libv4l2 options to DEB\_CONFIGURE\_EXTRA\_FLAGS.

Build gstreamer1.0-plugins-good package:

debuild -b -uc -us

Install the newly built gstreamer1.0-plugins-good package:

dpkg -i ../gstreamer1.0-plugins-good\_1.10.4-1\_arm64.deb

Now you are ready to do video recording! Configure the pipeline:

sudo media-ctl -d /dev/media1 -l '"msm\_csiphy0":1->"msm\_csid0":0[1],"msm\_csid0":1->"msm\_ispif0":0[1],"msm\_ispif0":1->"msm\_vfe0\_pix":0[1]'

sudo media-ctl -d /dev/media1 -V '"ov5645 1-0076":0[fmt:UYVY8\_2X8/1280x960 field:none],"msm\_csiphy0":0[fmt:UYVY8\_2X8/1280x960 field:none],"msm\_csid0":0[fmt:UYVY8\_2X8/1280x960 field:none],"msm\_ispif0":0[fmt:UYVY8\_2X8/1280x960 field:none],"msm\_vfe0\_pix":0[fmt:UYVY8\_2X8/1280x960 field:none]'

And do video recording:

gst-launch-1.0 -e v4l2src device=/dev/video3 ! video/x-raw,format=NV12,width=1280,height=960,framerate=30/1 ! v4l2video5h264enc extra-controls="controls,h264\_profile=4,video\_bitrate=2000000;" ! h264parse ! mp4mux ! filesink location=video.mp4

When using the above command make sure that video device node numbers are correct or check and use the correct ones:

examine the output of media-ctl -d /dev/media1 -p and check that /dev/video3 is connected to msm\_vfe0\_pix

examine the output of gst-inspect-1.0 video4linux2 and check the name of the V4L2 H.264 Encoder plugin – e.g. v4l2video5h264enc

## Using CSID Test Generator

If you do not have any sensor, it is possible to use the internal CSID Test Generator (TG). To enable it download and compile the yavta tool from here:

git clone git://git.ideasonboard.org/yavta.git

cd yavta

make

Then, enable the test generator:

sudo ./yavta --no-query -w '0x009f0903 1' /dev/v4l-subdev2

Then, configure the pipeline:

sudo media-ctl -d /dev/media1 -l '"msm\_csid0":1->"msm\_ispif0":0[1],"msm\_ispif0":1->"msm\_vfe0\_rdi0":0[1]'

sudo media-ctl -d /dev/media1 -V '"msm\_csid0":1[fmt:UYVY2X8/1920x1080],"msm\_ispif0":0[fmt:UYVY2X8/1920x1080],"msm\_vfe0\_rdi0":0[fmt:UYVY2X8/1920x1080]'

Finally, you can use any v4l2 application, such as Gstreamer.

## Optimized video pipeline

In order to exercise a fully optimized video pipeline some specific settings need to be applied throughout the pipeline elements. With the fully optimized video use case there is no extra buffer copy, and the various drivers involved are sharing all video buffers using dmabuf.

To exercise the opimized video pipeline, you can use for example the following gstreamer command:

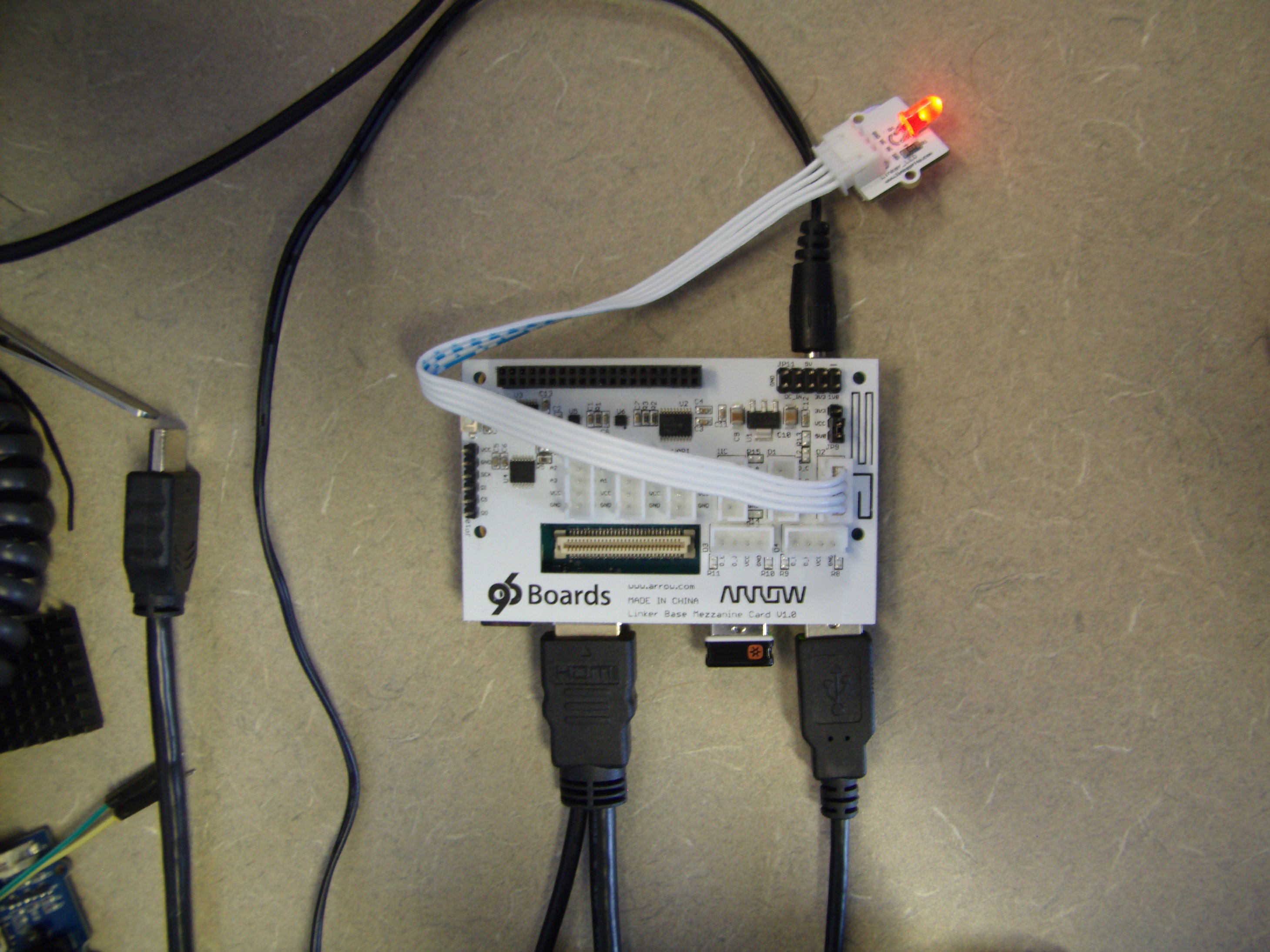
GST\_GL\_PLATFORM=egl gst-launch-1.0 filesrc location=<path to file> ! qtdemux name=m m.video\_0 ! h264parse ! v4l2video0dec capture-io-mode=dmabuf ! glimagesink

# Experiments

In the following sections we will give a few example experiments where we will connect a specific peripheral, add the driver to the kernel, add the connectivity information to the device tree, and finally create a small program to display information gathered from the peripheral. The student is expected to take these example recipes and extend the concepts to peripherals specific to their needs. Additionally the student can write applications in their favorite programming language to do more interesting things with the data collected from (or output to) the peripheral.

## Blinky LED (Linker Kit)

The simplest experiment which is usually done on every embedded system is the ‘blinky’ program, a LED is attached to a GPIO, and then the LED is turned on and off. The drivers for the GPIOs are already in the default kernel, hence no changes are required. This experiment will simply connect a LED to a GPIO and a shell script will be used to turn the LED on and off. The script enables the GPIO, then turns the LED on and off.

Connect the LED to GPIO\_A as shown in the picture below:

The following shell commands will, create the GPIO in the file system (export), set the direction to output, and then turn the LED on and off:

cd /sys/class/gpio

echo 13 | sudo tee export

cd gpio13

echo out | sudo tee direction

echo 1 | sudo tee value

echo 0 | sudo tee value

Further challenge, write a program in a programing language other than shell script to blink the LED. An example ‘C’ program which uses libsoc is included in the Linker Kit tutorial at <http://linksprite.com/wiki/index.php5?title=Linker_Mezzanine_card_starter_kit_for_96board> . libsoc also supports Python programs.

## Pushbutton

The next common problem is to read a simple switch connected to a GPIO. In this experiment we will connect a pushbutton to a GPIO, and read the value of the pushbutton with a shell script.

TODO – insert picture of setup and add shell script to export and read the value of the GPIO.

Further challenge, write a program using libsoc to poll the button and print the state of the button when it changes, don’t forget to add a usleep() in the polling loop so that it doesn’t consume 100% of the CPU resource. Added challenge, set up the GPIO to generate an interrupt when it changes state, and write a program to intercept the interrupt and print the new state of the button.

Note: not all of the GPIOs are capable of generating interrupts. The following list shows which GPIOs are capable:

# dragonboard 410c pin layout

#<Pin Name> <SoC Num>

GPIO\_A = 36 #interrupt

GPIO\_B = 12 #interrupt

GPIO\_C = 13 #interrupt

GPIO\_D = 69 #interrupt

GPIO\_E = 115 #interrupt

GPIO\_F = 4

GPIO\_G = 24

GPIO\_H = 25 #interrupt

GPIO\_I = 35 #interrupt

GPIO\_J = 34 #interrupt

GPIO\_K = 28 #interrupt

GPIO\_L = 33

## SPI based sliding potentiometer (Linker Kit)

The linker board from LinkSprite has an on-board 10-bit 4-channel Analog to Digital Converter (ADC). The tutorial that comes with the Linksprite board can be found here:

<http://linksprite.com/wiki/index.php5?title=Linker_Mezzanine_card_starter_kit_for_96board>

The sample code included with the tutorial can be used with a generic spi device driver.

### Modify the Kernel

To setup the generic SPI device, first add to the kernel configuration [note: Debian kernels released by Linaro after April 29/2016, this is the default for the kernel, no need to do this step]

make menuconfig

Select “Device Drivers” then <Enter>

Select “SPI Support” followed by <Enter>

Select “User mode SPI device driver support” Followed by ‘Y’.

Back out of the menus with <esc><esc> and at the end select ‘Yes’ to save the changes.

then spidev needs to be added to the device tree[this still needs to be done]. Add the two blocks of code proceeded with a ‘+’ sign into the file arch/arm64/boot/dts/qcom/apq8016-sbc.dtsi (do not add the ‘+’ character).

diff --git a/arch/arm64/boot/dts/qcom/apq8016-sbc.dtsi b/arch/arm64/boot/dts/qcom/apq8016-sbc.dtsi

index f6d2bcb6dbd1..874150a412a2 100644

--- a/arch/arm64/boot/dts/qcom/apq8016-sbc.dtsi

+++ b/arch/arm64/boot/dts/qcom/apq8016-sbc.dtsi

@@ -208,8 +208,14 @@

/\* On Low speed expansion \*/

label = "LS-SPI0";

status = "okay";

+/\* cs-gpios = <&msmgpio 18 0>; \*/

+ spidev@0 {

+ compatible = "spidev";

+ spi-max-frequency = <100000>;

+ reg = <0>;

+ };

};

leds {

In the file arch/arm64/boot/dts/qcom/msm8916-pins.dtsi near line 211 change the line marked with a + to (do not add the + character)

diff --git a/arch/arm64/boot/dts/qcom/msm8916-pins.dtsi b/arch/arm64/boot/dts/qcom/msm8916-pins.dtsi

index 899f2b28a9c9..1c22b4414edf 100644

--- a/arch/arm64/boot/dts/qcom/msm8916-pins.dtsi

+++ b/arch/arm64/boot/dts/qcom/msm8916-pins.dtsi

@@ -208,7 +208,7 @@

pins = "gpio16", "gpio17", "gpio19";

};

pinmux\_cs {

- function = "gpio";

+ function = "blsp\_spi5";

pins = "gpio18";

};

pinconf {

@@ -218,7 +218,7 @@

};

pinconf\_cs {

pins = "gpio18";

- drive-strength = <2>;

+ drive-strength = <12>;

bias-disable;

output-high;

};

In the file drivers/spi/spi-qup.c near lines 583 and 931 add the SPI\_IO\_C\_MX\_CS\_MODE bits:

diff --git a/drivers/spi/spi-qup.c b/drivers/spi/spi-qup.c

index 68f95acf7971..aed71ef7e3fd 100644

--- a/drivers/spi/spi-qup.c

+++ b/drivers/spi/spi-qup.c

@@ -580,6 +580,7 @@ static int spi\_qup\_io\_config(struct spi\_device \*spi, struct spi\_transfer \*xfer)

else

control &= ~SPI\_IO\_C\_CLK\_IDLE\_HIGH;

+ config |= SPI\_IO\_C\_MX\_CS\_MODE;

writel\_relaxed(control, controller->base + SPI\_IO\_CONTROL);

config = readl\_relaxed(controller->base + SPI\_CONFIG);

@@ -928,7 +929,7 @@ static int spi\_qup\_probe(struct platform\_device \*pdev)

base + QUP\_ERROR\_FLAGS\_EN);

writel\_relaxed(0, base + SPI\_CONFIG);

- writel\_relaxed(SPI\_IO\_C\_NO\_TRI\_STATE, base + SPI\_IO\_CONTROL);

+ writel\_relaxed(SPI\_IO\_C\_NO\_TRI\_STATE|SPI\_IO\_C\_MX\_CS\_MODE, base + SPI\_IO\_CONTROL);

ret = devm\_request\_irq(dev, irq, spi\_qup\_qup\_irq,

IRQF\_TRIGGER\_HIGH, pdev->name, controller);

After you have made changes to the kernel configuration and the device tree, you can use the scripts to recompile and install your new kernel, the rebuild takes about 5 minutes:

../do-rebuild-kernel

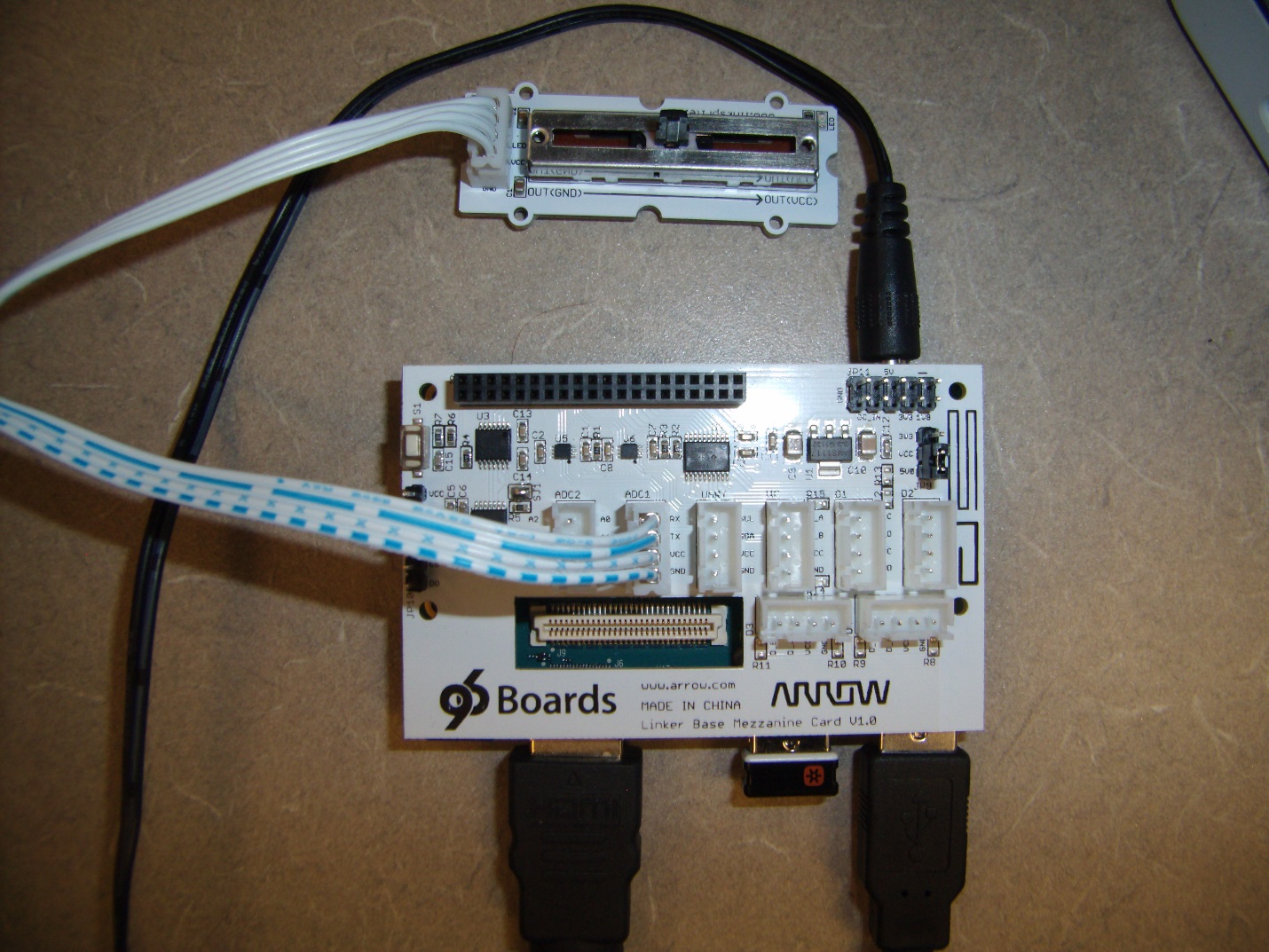
../do-install

sudo reboot now

Once the system reboots you should have a new entry in the file system, /dev/spidev0.0.

### Hardware Setup

Connect the Linker mezzanine board to the DragonBoard 410c, and connect the slide pot to the Linker board as shown:



### Build the Application

In your Makefile add the following:

LDFLAGS = -lsoc

Copy the code from <http://learn.linksprite.com/96-board/sliding-rheostat/> to a local file called slider.c. Edit the file slider.c, change the constant ‘32766’ to ‘0’, and remove all of the GPIO code, then run the commands:

make slider

sudo ./slider

The slider program will output the position of the slider once per second.

### Accessing the slider from a Python program

Make the kernel changes outlined in section above, then run the following Python program

import spidev

spi=spidev.SpiDev()

spi.open(0,0)

spi.max\_speed\_hz=100000

channel\_select=[0x01, 0x80, 0x00]

if \_\_name\_\_==’\_\_main\_\_’:

print(“welcome to the slide pot reader”)

try:

while True:

adc\_data=spi.xfer(channel\_select)

adc=((adc\_data[1]<<8)&0x300)|(adc\_data[2]&0xFF)

print(adc)

except KeyboardInterrupt:

print(“CTRL-C, Exiting”)

## Programming the Arduino (native development)

Install the arduino development system

Attach a shield

Download an Arduino sketch

Compile sketch

Transfer to Arduinio

Run it

These steps are outlined in the “Getting Started” document here: <https://www.96boards.org/products/mezzanine/sensors-mezzanine/>

## Power Measurement with INA219

In this section we will describe how to use a readily available INA219 Breakout Board. The INA219 chip is a complete power measurement system on a chip. This chapter shows how to connect the breakout board. A simple test that causes the power consumption to increase is also described and can be used to verify that the INA219 is correctly connected.

### INA219 Breakout Board

The INA219 Breakout board is a complete power measurement system available from many suppliers at relatively low cost. In this chapter we will use this specific implementation: <https://learn.adafruit.com/adafruit-ina219-current-sensor-breakout/overview>

The instructions to rebuild the kernel presented in the early chapters of this document added the necessary driver and device tree entry for the ina219 power monitor.

If necessary assemble the ina219 breakout board and connect the I2C measurement port to the DragonBoard 410c through the sensors-mezzanine board.

Equipment required:

96Boards Sensors Mezzanine card:

<https://www.96boards.org/products/mezzanine/sensors-mezzanine/>

I2C jumper cable:

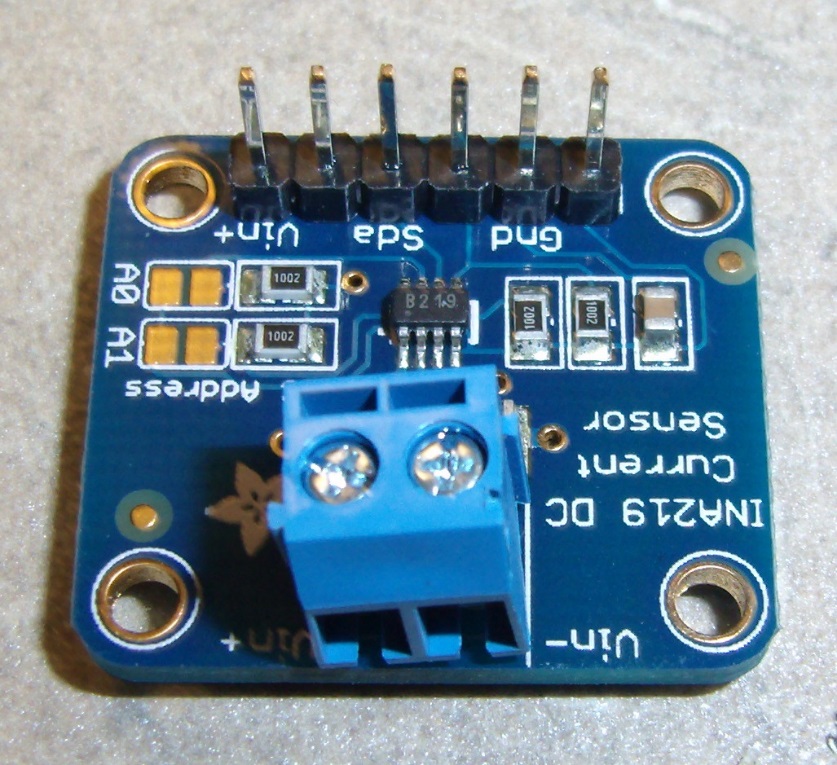
<http://www.seeedstudio.com/depot/Grove-4-pin-Female-Jumper-to-Grove-4-pin-Conversion-Cable-5-PCs-per-PAck-p-1020.html>

INA219 Breakout Board:

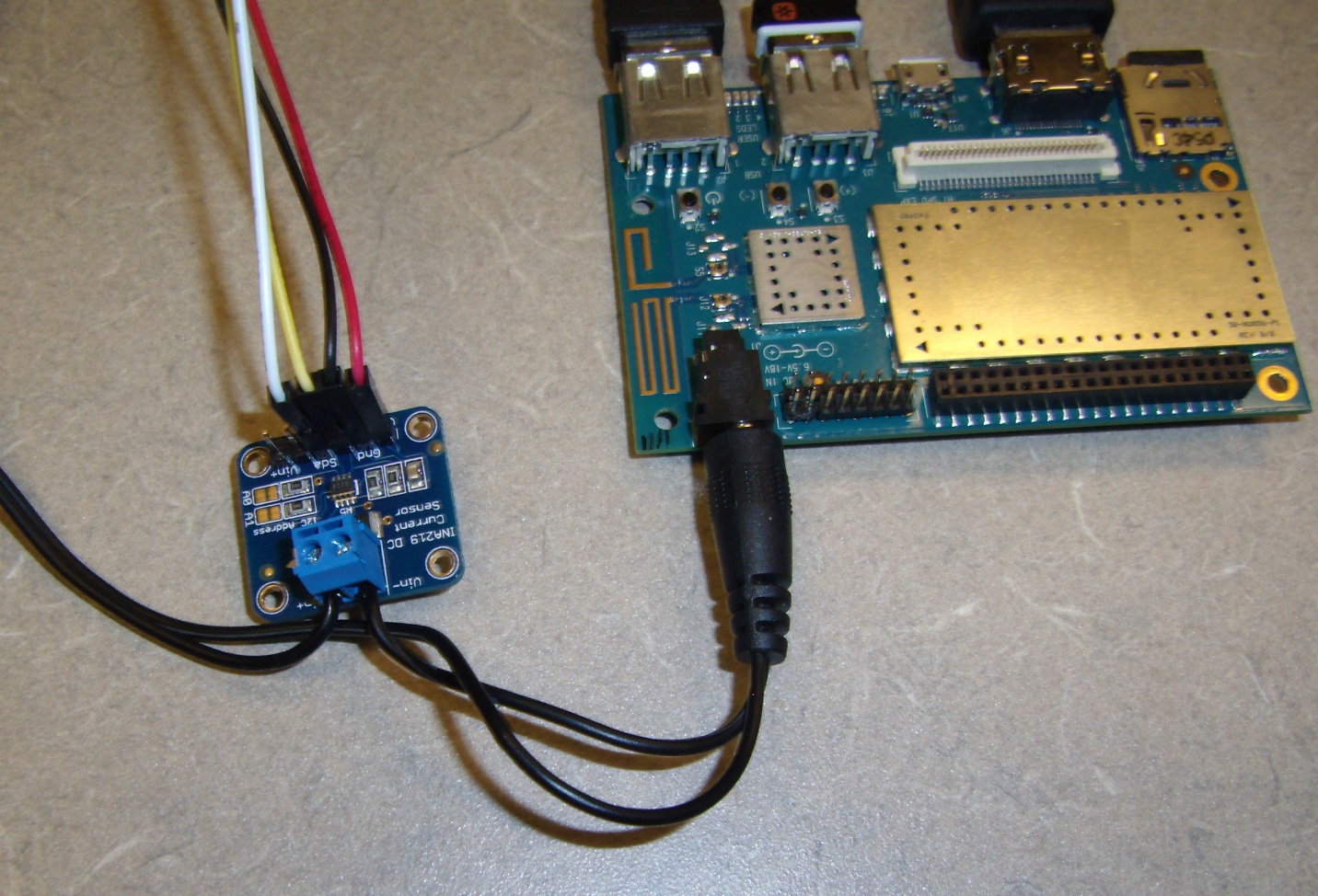
<https://learn.adafruit.com/adafruit-ina219-current-sensor-breakout/overview>

Steps

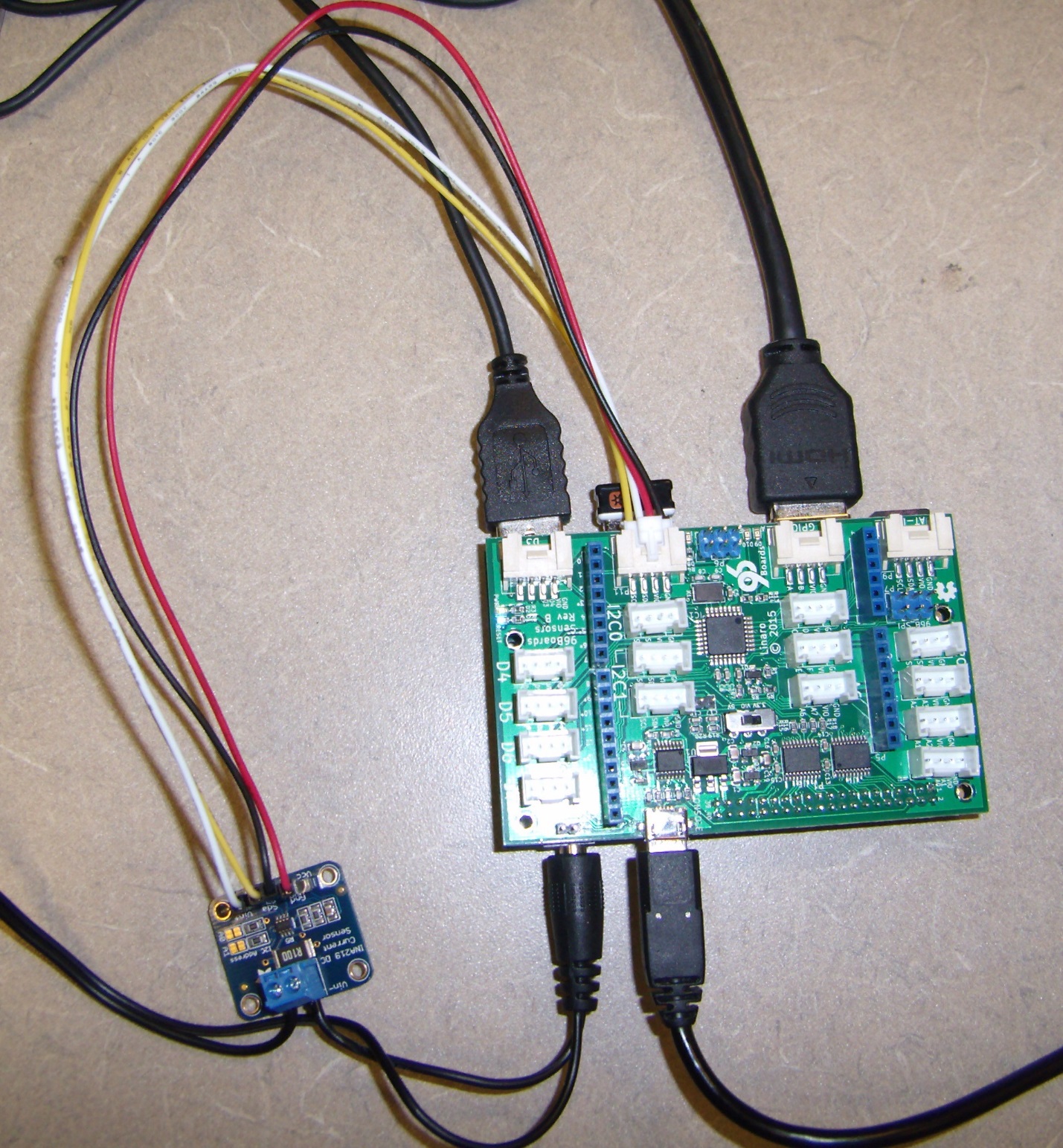
1. Attach the 96Boards Sensors Mezzanine board to the 410c
2. Connect the I2C cable to the mezzanine board on one of the two I2C0 connectors.
3. Assemble the INA219 Board following the AdaFruit instructions here: <https://learn.adafruit.com/adafruit-ina219-current-sensor-breakout/assembly> When assembly is complete your board should look like this:



1. Cut the positive lead to your power supply, strip the wires and connect them to the INA219 Current Sensor breakout. The 12V supply lead should connect to the Vin+ terminal. The 12V lead going to the 410c should be connected to the Vin- terminal. Like this:



1. Connect the power and I2C signals from the 410c board to the INA219 Current Sense Breakout
   1. +5V (red wire)
   2. GND (black wire)
   3. SCL (clock, yellow wire)
   4. SDA (data, white wire)
2. Connect the cable to the mezzanine board on the I2C0 connector.



### Monitoring CPU Speed and Temperature

It is possible to use built-in utilities to monitor the board power consumption.

Setup the required software

# do this once to set up your linux system (it might already be installed)

Under Debian start a terminal session and execute the following command:

# view the core frequencies and the temperatures.

# May need slightly different paths on your system.

sudo watch cat \

/sys/devices/system/cpu/cpu\*/cpufreq/cpuinfo\_cur\_freq \

/sys/devices/virtual/thermal/thermal\_zone\*/temp \

/sys/bus/i2c/drivers/ina2xx/0-0040/hwmon/hwmon0/power1\_input

This will display the speeds of the 4 CPUs, the 2 temperature monitors near the CPUs, and the power consumption as measured by the INA219 board.

You should see this when you are running the watch command:

Every 2.0s: cat /sys/devices/system/cpu/cpu0/cpufre... Fri Dec 18 17:36:04 2015

200000

200000

200000

200000

46000

44000

1800000

The first four numbers are the current clock rates for the 4 A53 CPUS (in Hz, 200Mhz each). The next two numbers are the temperatures from the on chip temperature monitor points (in milli-degrees C, 46C and 44C). The final number is the power consumption in micro-Watts, 1.8Watts).

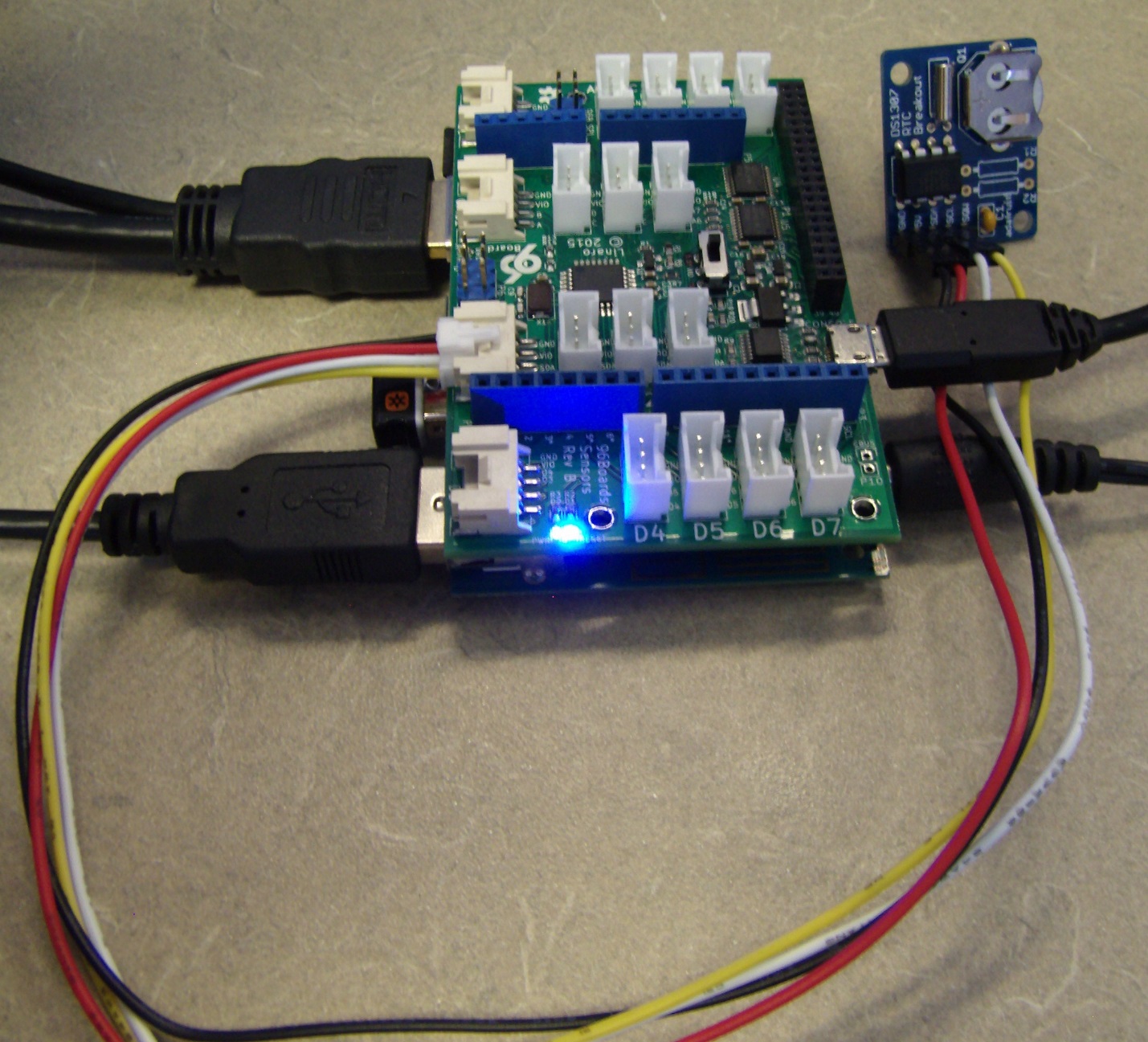
You can increase power consumption on the board by running the following command

stress –c 4 –i 4 –m 2 &

## DS1307 Real Time Clock

The 410c does not have a battery backed up real-time clock. Under common operating conditions, the 410c software simply goes to a network time server and updates the real-time clock soon after power-up. This keeps the time on the board in sync with the real-time. Certain applications may be running in a location where network time is not available and will require the 410c to power up and determine the real-time from a battery backed clock. Battery backed clock modules are readily available one example is DS1307 real-time clock (RTC) breakout board available from Adafruit <https://www.adafruit.com/product/264>, there are many other real-time clock chips that could be used.

In this section we will connect the clock module, make changes to the kernel to access the RTC, and finally set up the system to read the RTC at power-up time. First connect the hardware:



Note: When the AdaFruit DS1307 RTC Breakout kit is assembled the I2C pull-up resistors are NOT installed (R1 and R2), these components are not needed and will cause issues on the I2C bus if installed.

Enable the driver for the DS1307 RTC in the kernel.

make menuconfig

Select “Device Drivers” then <Enter>

Select “Real Time Clock” followed by <Enter>

Select “Dallas/Maxim DS1307/37/38/39/40…” Followed by ‘Y’.

Back out of the menus with <esc><esc> and at the end select ‘Yes’ to save the changes.

Add the block of code proceeded with a ‘+’ sign into the file arch/arm64/boot/dts/qcom/apq8016-sbc.dtsi (do not add the ‘+’ character).

i2c@78b6000 {

/\* On Low speed Expansion \*/

Label = “LS-I2C0”

status = "okay";

+ rtc@68 {

+ compatible = "maxim,ds1307";

+ reg = <0x68>;

+ };

};

Then rebuild and install the new kernel and device tree. After you have made changes to the kernel configuration and the device tree, you can use the scripts to recompile and install your new kernel, the rebuild takes about 5 minutes:

../do-rebuild-kernel

../do-install

sudo reboot now

After the 410c reboots you need to set the time in the real-time clock, then remove the fake clock and install the hardware clock.

sudo hwclock –w # write the time.

sudo hwclock –r # read the time.

sudo update-rc.d -f fake-hwclock remove

sudo update-rc.d hwclock.sh enable

You can test your changes by disabling Wifi, powering the board down, then restarting the board and checking the time. You can also check the bootlogs, you should see the RTC discovered during boot:

[ 4.679390] i2c /dev entries driver

[ 4.691075] i2c\_qup 78b6000.i2c:

[ 4.691075] tx channel not available

[ 4.739080] rtc-ds1307 0-0068: rtc core: registered ds1307 as rtc0

[ 4.739690] rtc-ds1307 0-0068: 56 bytes nvram

# TurtleBot Installation

In the following sections we connect the 410c to a Turtlebot, install the ROS operating Software onto the 410c and control the Turtlebot with the 410c.

The student is expected to take this example recipes and extend the concepts to peripherals specific to their needs. Additionally the student can write applications in their favorite programming language to do more interesting things with the data collected from (or output to) the peripheral.

The ROS software eco system needs two computers to operate completely. The first computer is the ‘controller’ (usually a desktop PC or a laptop), and the second computer is the Robot (in this case it will be a 410c DragonBoard).

In the following sections we will cover setting up the Hardware (connecting the 410c Dragonboard to the robot) setting up the ROS software on the host computer, and finally setting up the software on the robot

## Hardware setup

<TODO>I need to add pictures, parts lists, etc. to this section.

Connect 410c – keyboard/mouse/display/astra camera/etc.

<TODO> Pictures of setup

<TODO> pictures of heatsink assy

Power – the 410c can be connected directly to the 12V output on the Kobuki base.

<TODO> add parts list, pictures and assembly instructions for the 12V power cable.

Hub – I used a powered USB hub and powered it from the 5V power output on the Kobuki base.

<TODO> add part and model number for the hub.

<TODO> add parts list, pictures and assembly instructions for the 5V power cable.

Orbec Astra Cam – the Astra camera needs to be plugged directly into one of the two ports on the 410c, do not connect it through a hub.

Etc.

## Install Ubuntu software onto a Win10 PC

NOTE: At this instant WSL can’t run ROS due to a sockets/getifaddrs problem. See this thread for details: <http://answers.ros.org/question/238646/installing-ros-on-ubuntu-bash-in-windows-10/?sort=latest#sort-top>

The ROS software is intended to run under Ubuntu, typically the host computer will be a dedicated Ubuntu computer, or at least a laptop with Dual-Boot capability so that it will boot into Ubuntu software. As of January 2016, Microsoft Win10 not runs Ubuntu under Win10. In these instructions we will setup Win10 to run Ubuntu, then setup ROS to run under Ubuntu. (Hopefully with no show-stopper bugs). Install instructions paraphrased from here: <https://msdn.microsoft.com/en-us/commandline/wsl/install_guide>

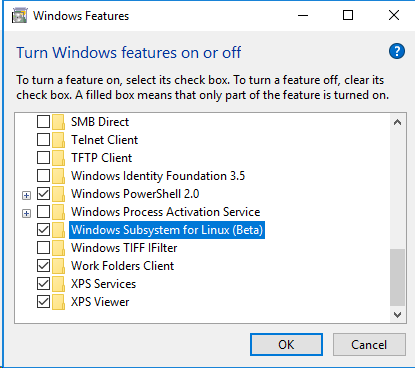
Update Ubuntu version or totally remove it: <https://blogs.msdn.microsoft.com/commandline/2016/10/19/wsl-adds-ubuntu-16-04-xenial-support/>

lsb\_release -a

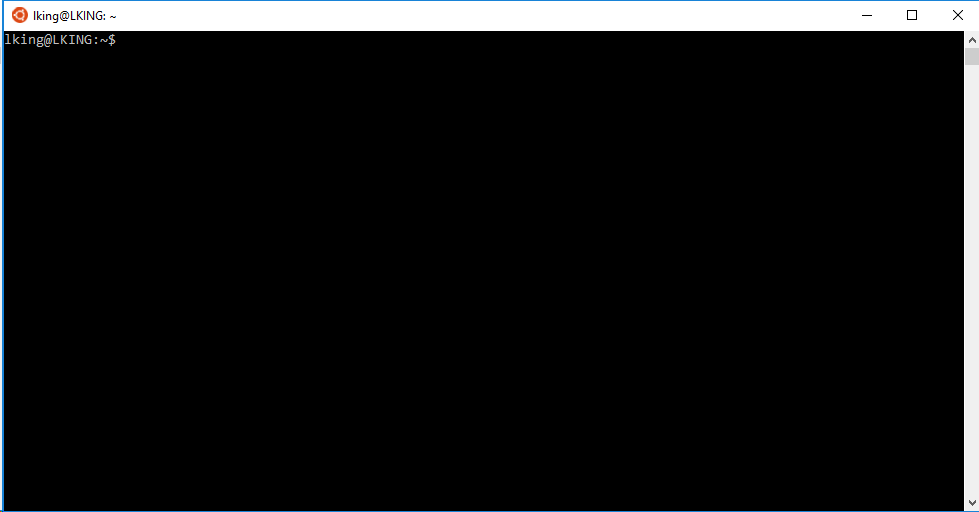
### Enable Ubuntu under Win10

Win10 has a module called WSL (Windows Subsystem for Linux), first step is to turn on WSL.

* In the search box (beside the start button) type ‘Turn Windows features’ and start the control panel application.
* Scroll down and select ‘Windows Subsystem for Linux (Beta)’, then hit OK. A system reboot will be required to turn on this feature.



* After the reboot type ‘bash’ into the search box (beside the start button) and start bash. Bash will ask a few questions and download Canonical Ubuntu form the windows store. This will take a while to install.
* After Ubuntu is installed you will have a ‘new’ version of bash. Start a new bash window to enter the commands in the next sections. Your bash window should look something like this:



The first thing is you need to **run bash as an administrator**, otherwise the network code fetches won’t work.

* Click on the start button
* Scroll down until you see the bash icon (under the letter ‘B’)
* Right click on the bash icon and select ‘More -> Run as Administrator’

Run the command and remember the output

hostname

Edit /etc/hosts and add the hostname output to the end of the very first line in the file. The first line in the file should look something like this:

127.0.0.1 localhost <your-hostname>

Alternatively online instructions say that you can simply delete the file /etc/hosts, then shutdown all bash windows. When you restart a bash window it will automatically generate a new /etc/hosts file with the name of your system in it. I haven’t tested this myself.

### Install an X-server

We need to be able to run the graphical applications that are part of ROS on the host PC under Win 10. This requires an X11 server. Go to <https://sourceforge.net/projects/xming/> and download and install Xming. Once Xming launches you should see an X icon in your system tray.

Next we need to tell the bash windows where to find the X11 display.

echo "export DISPLAY=:0.0" >> ~/.bashrc

and finally a little trickery to get everything running under WSL. Change from using Unix sockets for X to using tcp ports which work under WSL

sudo sed -i 's$<listen>.\*</listen>$<listen>tcp:host=localhost,port=0</listen>$' /etc/dbus-1/session.conf

## Install ROS Software onto the Host computer

Now run the following commands to install ROS into the PC.

sudo apt-get update

sudo apt-get upgrade -y

sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb\_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'

sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key 421C365BD9FF1F717815A3895523BAEEB01FA116

sudo apt-get update

At this point we need to install the Kinetic version of ROS on the host computer, and the Kinetic version of ROS on the robot. If you are trying this under Win10 at this time you need to install Indigo instead of kinetic. Apparently there shouldn’t be a problem having the host and the target running different versions of ROS.

sudo apt-get install ros-kinetic-desktop-full –y

sudo apt-get install ros-kinetic-kobuki-dashboard –y

sudo apt-get install ros-kinetic-turtlebot-simulator -y

sudo apt-get install git –y

sudo apt-get install libudev-dev –y

echo source /opt/ros/kinetic/setup.bash >>~/.bashrc

source ~/.bashrc

### Build the turtlebot software for the host PC

In order to simulate the robot you first need to install and build all of the software. This step is necessary even though you are not running the hardware on the host PC, you will still be able to run the simulator.

mkdir -p ~/workspace/turtlebot/src

cd ~/workspace/turtlebot/src

git clone <https://github.com/turtlebot/turtlebot_interactions.git>

git clone <https://github.com/orbbec/ros_astra_camera>

#sudo ros\_astra\_camera/install.sh

cd ..

#catkin\_make --pkg astra\_camera

catkin\_make install -DCMAKE\_BUILD\_TYPE=Release

## Install ROS Software onto the 410c

These instructions are adapted from the instructions original found here: <https://developer.qualcomm.com/project/turtlebot-2e>. These instructions work assuming that you are running Ubunty ‘trusty’ on your 410c. If you are running a newer ‘stretch’ version of Ubuntu you will need to rebuild ROS from source (see below)

Follow the software setup instructions in chapters <X> and install all of the development software. Add a swap file so you don’t run out of memory in some of the steps below. See the steps above for using a SDCard, at least 3GB is required. I used 8GB.

Here are the additional packages you will need to get the Turtlebot running.

sudo apt-get update

sudo apt-get install dirmngr –y

Setup your computer to accept software from packages.ros.org. ROS Indigo ONLY supports Saucy (13.10) and Trusty (14.04) for debian packages.

sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb\_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'

Set up your keys

sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key 0xB01FA116

Now that you have pointed to the ROS packages, you need to tell apt about them and then install ROS-kinetic (~750 packages).

sudo apt-get update

sudo apt-get install ros-kinetic-desktop-full –y

#sudo apt-get install ros-kinetic-turtlebot ros-kinetic-turtlebot-apps ros-kinetic-turtlebot-interactions ros-kinetic-turtlebot-simulator ros-kinetic-kobuki-ftdi ros-kinetic-ar-track-alvar-msgs

Next install some more turtlebot related packages (~200 packages):

#sudo apt-get install ros-kinetic-turtlebot-apps -y

sudo apt-get install libudev-dev –y

sudo apt-get install ros-kinetic-turtlebot\* -y

#sudo apt-get install ros-kinetic-slam-gmapping -y

#sudo apt-get install ros-kinetic-librealsense -y

#sudo apt-get install ros-kinetic-kobuki\* -y

#sudo apt-get install ros-kinetic-ros-tutorials –y

#sudo apt-get install python-rosdep –y

#sudo apt-get install python-rosinstall -y

Initialize rosdep and git

sudo apt-get install python-rosdep

sudo rosdep init

sudo rosdep fix-permissions

rosdep update

git config –-global user.email “you@example.com”

git config –-global user.name “Your Name”

Update your .bashrc so that each time you login your environmental variables will be set correctly. Add the following to the end of your ~/.bashrc file. ~~Choose ‘astra’ or ‘astra\_s’ depending on which camera you have installed on your turtlebot.~~

export TURTLEBOT\_3D\_SENSOR=astra

#export TURTLEBOT\_3D\_SENSOR=astra\_s

export TURTLEBOT\_BATTERY=None

export ROS\_PARALLEL\_JOBS='-j4 –l4'

export ROS\_OS\_OVERRIDE=ubuntu:16.04

source /opt/ros/kinetic/setup.bash

source ~/workspace/turtlebot/devel/setup.bash

Note: the calls to setup.bash will fail until after you have performed some of the steps below.

Check that your environment looks reasonable:

source ~/.bashrc

printenv | grep ROS

printenv | grep TURTLE

### Rebuilding all of ROS from source

You can skip this section if you are using the precompiled binaries you have downloaded in the steps above. Move down to the Using ROS section.

Install ROS python packages

#source /opt/ros/kinetic/setup.bash

sudo apt-get install python-rosinstall -y

sudo apt-get install python-rostopic –y

sudo apt-get install python-empy python-nose libboost-all-dev –y

sudo apt-get install libboost-doc autopkgtest

Build the console\_bridge package

mkdir -p ~/workspace/ROS/cb\_ws

cd ~/workspace/ROS/cb\_ws

git clone https://github.com/ros/console\_bridge

mkdir build

cd build

cmake -DCMAKE\_INSTALL\_PREFIX=/opt/ros/kinetic ../console\_bridge

sudo mkdir /opt/ros

sudo chown linaro.linaro /opt/ros

make install

export console\_bridge\_DIR=/opt/ros/kinetic/lib/console\_bridge/cmake

# get some additional packages that ROS depends on.

sudo apt-get install doxygen libgtest-dev libprotobuf-dev -y

sudo apt-get install libpoco-dev libpoco-doc –y

sudo apt-get install libeigen3-dev libeigen3-doc –y

sudo apt-get install zlib1g-dev qtbase5-dev gstreamer1.0 –y

sudo apt-get install python-qt-binding –y

#rebuild all of ROS from source

sudo apt-get install python-rosdep python-rosinstall-generator python-wstool python-rosinstall build-essential

# initialize rosdep

sudo rosdep init

rosdep update

# make a workspace

mkdir -p ~/workspace/ROS/ros\_catkin\_ws

cd ~/workspace/ROS/ros\_catkin\_ws

#fetch the source (choose desktop\_full, desktop, or ros\_comm)

rosinstall\_generator desktop --rosdistro kinetic --deps --wet-only --tar > kinetic-desktop-wet.rosinstall

wstool init –j4 src kinetic-desktop-wet.rosinstall

# resolve dependancies

rosdep install --from-paths src --ignore-src --rosdistro kinetic –y --os=ubuntu:trusty

# use catkin\_make to build ROS

./src/catkin/bin/catkin\_make\_isolated --install --install-space /opt/ros/kinetic -DCMAKE\_BUILD\_TYPE=Release

## Using ROS

Setup ROS environment:

sudo update-locale LANG=C LANGUAGE=C LC\_ALL=C LC\_MESSAGES=POSIX

source /opt/ros/kinetic/setup.bash

Create a Catkin Workspace:

mkdir -p ~/workspace/turtlebot/src

cd ~/workspace/turtlebot/src

catkin\_init\_workspace

cd ~/workspace/turtlebot

catkin\_make

use git clone to get packages from turtlebot/src then build

cd ~/workspace/turtlebot/src

git clone <https://github.com/turtlebot/turtlebot_interactions.git>

git clone <https://github.com/orbbec/ros_astra_camera.git>

git clone [https://github.com/orbbec/ros\_astra\_launch.git](https://github.com/orbbec/ros_astra_launch)

Make the packages

cd ~/workspace/turtlebot

catkin\_make --pkg astra\_camera –DFILTER=ON

catkin\_make install -DCMAKE\_BUILD\_TYPE=Release

The catkin\_make steps should complete cleanly, examine the output messages carefully.

install the astra camera usb driver rules and the Kobuki device rules

# <http://wiki.ros.org/astra_camera>

sudo cp ros\_astra\_camera/56-orbbec-usb.rules /etc/udev/rules.d/

sudo service udev reload

sudo service udev restart

# unplug and plug in the astra camera to the USB port

source ~/workspace/turtlebot/devel/setup.bash

rosrun kobuki\_ftdi create\_udev\_rules

# unplug and plug in the kobuki base to the USB port

### Check that you are connected to the kobukibase

The following two commands will startup the ROS operating system and check that you can read the bumpers on the Kobuki Base. <http://edu.gaitech.hk/turtlebot/turtlebot-first-tests.html>

roslaunch turtlebot\_bringup minimal.launch &

rostopic echo /mobile\_base/events/bumper

pressing on the bumper will output the events.

### Upgrade the Astra Camera firmware

Depending on which Astra camera you have you may need to update the camera firmware.

~/workspace/turtlebot/src/ros\_astra\_camera/fwugrade/astra\_s\_fwupload\_1\_09\_3

Unplug the camera and plug it in back in to use the new firmware.

### Check that the camera works

You can run this test on the 410c if you have a monitor and keyboard connected. The command needs to be run from a terminal window on the 410c screen. Alternatively if you have set up a host computer and made the connection (see below) then you can run the image\_view commands on the host computer.

roslaunch turtlebot\_bringup 3dsensor.launch

#roslaunch turtlebot\_rviz\_launchers view\_robot.launch

rosrun image\_view image\_view image:=/camera/rgb/image\_rect\_color

rosrun image\_view image\_view image:=/camera/depth/image

## Connect the Host to the Robot over the Network

<http://wiki.ros.org/turtlebot/Tutorials/indigo/Network%20Configuration>

On the turtlebot in a terminal window run the following command:

ip addr

and note the IP Address <turtlebot-ip-addr> of the wlan0 device.

On the host PC in a Windows CMD window run the command:

ipconfig

And note the ip address <host-ip-addr>.

Start a bash terminal window on the host PC and enter the command:

ssh linaro@<turtlebot-ip-addr>

Enter the turtlebot password, this will start a shell on the turtlebot giving you remote control off the robot from your host computer.

On the Turtlebot execute the following commands:

echo export ROS\_MASTER\_URI=http://localhost:11311 \

>> ~/workspace/turtlebot/devel/setup.bash

echo export ROS\_HOSTNAME=<turtlebot-ip-addr> \

>> ~/workspace/turtlebot/devel/setup.bash

On the Host PC execute the following commands to permanently setup the .bashrc file.

echo export ROS\_MASTER\_URI=http://<turtlebot-ip-addr>:11311 \

>> ~/.bashrc

echo export ROS\_HOSTNAME=<host-ip-addr> >> ~/.bashrc

source ~/.bashrc

### Test the connection

On the turtlebot run the following command:

roslaunch turtlebot\_bringup minimal.launch --screen &

and on the host PC run the following command

rostopic list

if all is setup correctly you should get a list of about 30 topics.

You can also try these tests: <http://wiki.ros.org/turtlebot/Tutorials/indigo/Network%20Configuration>

## Update the ROS software on your Turtlebot

Do this only once (if you didn’t do it earlier):

sudo apt-get install python-rosdep

sudo pip install –U rosdep

sudo rosdep init

sudo rosdep fix-permissions

Then to update ros just run:

rosdep update

## Make the Turtlebot Move

Start the Turtlebot follower program (this gets the robot to move, but I do not yet understand why the second command doesn’t inherently run the first command). Run the following commands on the robot:

roslaunch turtlebot\_follower follower.launch

### Changing Follower Parameters

Run the dynamic reconfigure gui on your workstation computer (or on the 410c with a screen/keyboard/mouse attached).

rosrun rqt\_reconfigure rqt\_reconfigure

Select camera/follower on the reconfigure gui

Play with the parameters using the sliders; you will see the TurtleBot's following behavior change as you move the sliders.

### Send the Turtlebot back to the Dock

roslaunch turtlebot\_bringup minimal.launch &

roslaunch kobuki\_auto\_docking minimal.launch &

roslaunch kobuki\_auto\_docking activate.launch

## Adding an Arduino powered Arm to the Turtlebot

<http://wiki.ros.org/rosserial_arduino/Tutorials/Arduino%20IDE%20Setup>

## ROS Resources

<https://github.com/tfoote/ros_ifc6410_guide>

<http://wiki.ros.org/turtlebot/Tutorials/indigo/Network%20Configuration>

<http://edu.gaitech.hk/getting_started/get-started.html>

# Installing Amazon Alexa on the 410c

<https://developer.qualcomm.com/sites/default/files/attachments/alexa-service-db410c.pdf>

# Resources

A list of handy places on the NET to find helpful information:

Documentation, operating systems, schematics, online tech support, forums:

[www.96Boards.org](http://www.96Boards.org)

Connect the Dragonboard 410c to IBM Watson IoT

<https://developer.ibm.com/recipes/tutorials/dragonboard410c-recipe/>

Linker mezzanine board and tutorials

<http://linksprite.com/wiki/index.php5?title=Linker_Mezzanine_card_starter_kit_for_96board>

96Boards Sensors mezzanine Board and using Arduino

<http://www.seeedstudio.com/depot/96Boards-Sensors-p-2617.html>

STMicro Robotics mezzanine board

<https://www.96boards.org/products/mezzanine/stm32sensor/>

<https://github.com/mbedmicro/mbed>

Snappy Ubuntu Core for 410c

<https://insights.ubuntu.com/2016/02/24/canonical-to-offer-powerful-arm-64-bit-iot-developer-environment/>

Amazon Web Services on 410c – Getting started

<https://aws.amazon.com/iot/getting-started/>

Microsoft Azure – Getting started

<https://azure.microsoft.com/en-us/develop/iot/get-started/>

Google Brillo - Weave

<https://developers.google.com/brillo/>

Rebuilding Android on Ubuntu 16.04

<https://nathanpfry.com/how-to-setup-ubuntu-16-04-lts-xenial-xerus-to-compile-android-roms/>

Install ROS from source

<http://wiki.ros.org/kinetic/Installation/Source>