Getting Started Guide

Sungdong Kim
SAMSUNG

IoTivity Development on ARTIK 530s

GETTING STARTED GUIDE

Version 1.0, December 6, 2018

Table of Contents

Summary

This Getting Started Guide shows you how to set up an IoTivity development environment on an ARTIK(SAMSUNG IoT platform) 530s with Eagleye 530s developer kit. You will also build and run sample server and client applications that verify the build environment is set up properly, and can interact with an Internet of Things (IoT) device, in our case an LED. These sample applications are a baseline and reference for new developers to explore the IoTivity API framework and learn how to write secure server and client applications that can pass the OCF Certification Test Tool (CTT) and implement OCF Introspection (how devices discover and communicate their capabilities to each other, enabling interoperability)

OCF and IoTivity

IoTivity is an open source software project enabling seamless device-to-device connectivity where billions of wired and wireless Internet of Things (IoT) devices can securely connect to each other and to the internet. The Open Connectivity Foundation (OCF) develops specification standards, interoperability guidelines, and a certification program for these devices. IoTivity is an open source reference implementation of the OCF specification. You can learn more about OCF at http://openconnectivity.org and IoTivity at http://iotivity.org.

The IoTivity APIs expose the OCF framework to developers, and are available in several languages and for multiple operating systems. The framework supports dedicated and optimized protocols for IoT devices, with specific considerations for constrained devices, and addressing many types of devices, form-factors, companies, and markets.

The IoTivity framework operates as middleware across supported operating systems and connectivity platforms and has these four essential building blocks:

- **Discovery**: supporting multiple mechanisms for discovering devices and resources in proximity and remotely.
- **Data transmission**: supporting information exchange and control based on a messaging and streaming model.
- **Data management**: supporting the collection, storage, and analysis of data from various resources.
- **Device management**: supporting configuration, provisioning, and diagnostics of devices.

The IoTivity Development on Eagleye 530s developer Kit

This kit provides you with a Eagleye 530s board and additional hardware to get you started with IoT development using IoTivity APIs and interacting with sensor devices. You'll be building and running the sample client and server applications (written in C/C++) on the Eagleye 530s board itself, by following the instructions in this getting started guide. After following these instructions, you will have verified the on-board development environment and tools are set up and working. You can read through the sample applications source code to see how it works and make changes on your own to learn more!

Parts List

The kit contains the following hardware. (You can also assemble a hardware kit from this parts list):

- Eagleye 530s is development kit which incorporates the Samsung ARTIK 530s 1GB system-on-module, with a quad-core 64-bit Arm* A9 processor, HDMI, USB, Ethernet, on-board Wi-Fi*, Bluetooth* Low Energy, Zigbee and Thread support, 4GB eMMC 4.5 flash memory, 1GB RAM, Enterprise-class security with hardware secure element and Secure OS

- Micro-USB to USB cable. Micro USB is connected board and USB is connected PC

Optional parts

- Micro-USB 5V, 2.5A power supply

- Barrel Plug type 5V, 4A power supply

- MicroSD card, with latest ARTIK 530s 1G firmware Ubuntu (a variant of Linux* built for the ARTIK 530s) loaded. (An appendix has instructions for downloading and updating firmware.)

- Ethernet cable for connecting the Internet.

- Optionally, an HDMI monitor, HDMI cable, and a USB keyboard are recommended but not required (and not included in the kit).

Network Security Warning

You'll be connecting the Eagleye 530s to your local network and the Internet. If you're in a corporate environment or using a corporate laptop, it may be against corporate guidelines to do this because of security concerns. Please consult with your network admin. Corporate environments may also require proxy settings to allow tools such as Git to get through the firewall to GitHub and Gerrit repositories. Configuring your proxy settings is beyond the scope of this guide.

The Eagleye 530s setup uses a default login and password shared on every Eagleye 530s running Ubuntu. Secure setup of your development board is out of scope for this guide.

Working directly or indirectly with the Eagleye 530s

We've found it convenient to setup and communicate directly to the Eagleye 530s with an attached USB keyboard and HDMI monitor (not included in the kit). We do provide instructions in an appendix, for working with a "headless" Eagleye 530s via serial port to indirectly connect to

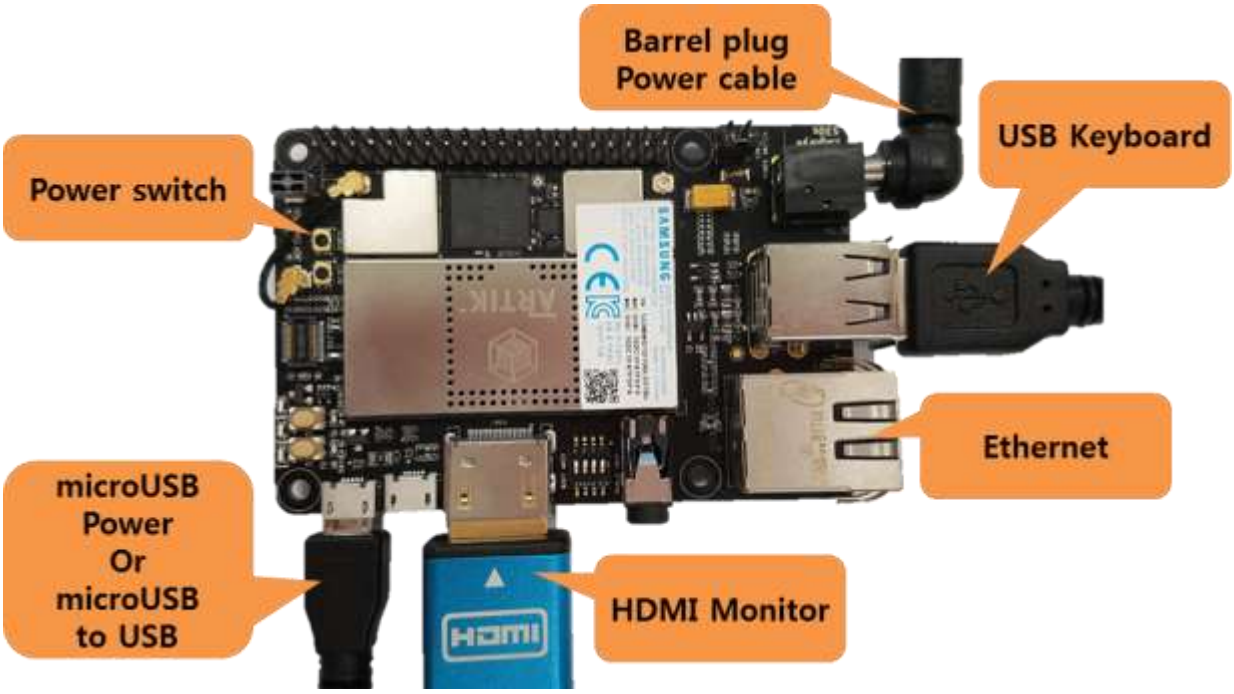
the board from a host computer. In either case, we use the Ubuntu command line interface so we won't need a mouse.

Setting up the Hardware

You have to choose the way to supply power first. because there are three ways(Supply from ①PC to microUSB-to-USB cable, ②microUSB power supply and ③barrel plug power supply)to supply power to Eagleye 530s. ① way is recommended. (Using PuTTY program to access to the Eagleye 530s can be tested more conveniently because it support copying and pasting). Let's get started!

1. Connect the microUSB power or microUSB to USB cable from your PC or barrel plug power on the Eagleye 530s board.
   (Unlike a raspberry pi, an Eagle eye has a power switch, so you can connect the power cable in advance.)
2. If you're using a monitor and keyboard, connect the HDMI cable to your monitor and the HDMI connector on the Eagleye 530s board, and connect the USB keyboard to any of the USB connectors.
   If you're not using a connected monitor and keyboard, we explain in an appendix how to connect to the board from a host computer using serial port.

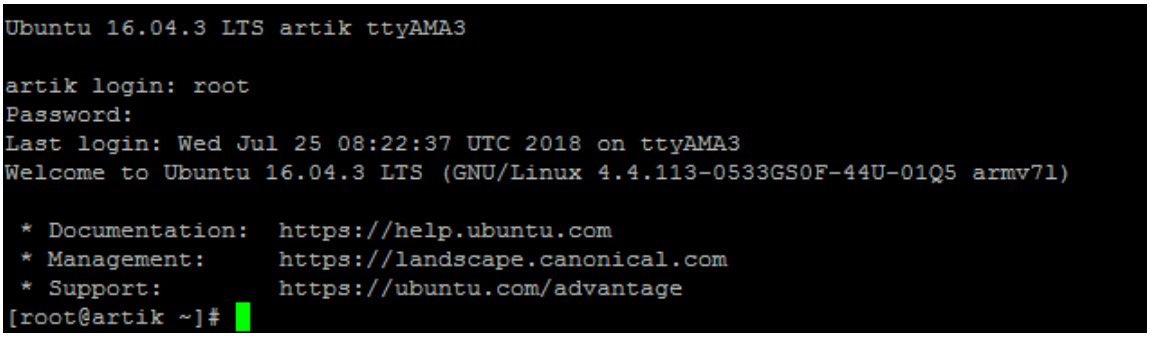
   Here's the Eagleye 530s board fully populated with kit components and cable connections (if you're accessing the board remotely via serial port, then you won't need the monitor or keyboard connection):



3. Now you're ready to press the power switch to boot the board, as explained in the next section.

Booting the Eagleye 530s with Ubuntu

Press the power switch(with a connected monitor) you'll see the Ubuntu OS boot and print log messages. After about 20 seconds, you'll be prompted to login. **The default username is "root" and password is "root".** Type those in, and you'll be at a (hopefully familiar looking) Linux command line prompt:



Password, Keyboard, and Date/Time Setup

Here are some recommended changes you should do after you login for the first time.

1. For security reasons, it's a good idea to change the default login password for the "root" user. Use the "passwd" command, enter the current password ("root") and change it to a new password you'll remember:

```
[root@artik~]# passwd
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
[root@artik~]#
```

2. This is also a good time (pun intended) to verify the date and time is set correctly on the Eagleye 530s , by using the "date" command. Set the date and time if it's not correct, with your local time, something like this for Pacific Daylight Time (PDT):

```
sudo date -s "Jan 6 14:33:00 PDT 2018"
```

An incorrect date and time, can cause system problems maintaining the file system, using security protocols, or with the build system.

You can also install network-time-protocol (ntp) packages that will keep the Eagleye 530s time synchronized automatically:

```
sudo apt-get update

sudo apt-get install ntp
```

3. Reboot the Eagleye 530s to have these changes go into effect:

```
sudo reboot
```

Setting up the IoTivity Development Environment

1. Before you start setting up the IoTivity-specific software development environment, you'll need to login again and then connect network (we explain in an appendix how to connect Wi-Fi network.) and make sure the base Linux OS is up to date by first "updating" information about available packages from the internet, then then upgrading installed packages if updates are available:

```
sudo apt-get update

sudo apt-get upgrade
```

Reply "y" and press enter when asked to continue. You'll see something like this:

```
[root@artik ~]# sudo apt-get upgrade
Reading package lists... Done
Building dependency tree
Reading state information... Done
Calculating upgrade... Done
The following packages were automatically installed and are no longer required:
  busybox-initramfs initramfs-tools initramfs-tools-bin initramfs-tools-core
  klibc-utils libklibc linux-base
Use 'sudo apt autoremove' to remove them.
The following packages will be upgraded:
  apt apt-utils artik-ota artik-plugin artik-plugin-artik530-audio
  artik-plugin-artik530-bluetooth artik-plugin-artik530-fstab
  artik-plugin-artik530-security artik-plugin-artik530-usb
The following packages will be DOWNGRADED:
  hostapd nodejs
147 upgraded, 0 newly installed, 2 downgraded, 0 to remove and 0 not upgraded.
Need to get 58.4 MB of archives.
After this operation, 1125 kB of additional disk space will be used.
Do you want to continue? [Y/n]
```

2. Next, use the same apt-get tool to install the tools and external libraries needed for the IoTivity build environment. As above, reply "y" and press enter when asked to continue. (Note this is a long command line you can cut and paste from below. If you're typing it in, the "\" at the end of a line means the next line is a continuation, or you can type it all in (without the "\" characters as one long line.)

```
sudo apt-get install build-essential git scons libtool automake \
autoconf valgrind doxygen wget unzip cmake chrpath libboost-dev \
libboost-program-options-dev libboost-thread-dev uuid-dev \
libexpat1-dev libglib2.0-dev libsqlite3-dev libcurl4-gnutls-dev
```

This command will install, if not already there, the gcc c++ compiler, Git version control system, build tools (such as scons, libtool, and auto conf), analysis tools (such as valgrind), API documentation tool (doxygen), utilities (such as wget and unzip), and other tools and external libraries needed by the project.

Getting the IoTivity source code

As with many open source projects, IoTivity is distributed as source code that you'll build in your development environment. These next steps connect to the internet to fetch the IoTivity source code and some additional tools.

1. In your home directory, make an "iot" working directory:

```
mkdir ~/iot

cd ~/iot
```

2. Clone the iotivity repository code using Git commands, change to the directory created, and check out the IoTivity 1.3 release branch:

```
git clone https://github.com/iotivity/iotivity.git

cd iotivity

git checkout 1.3-rel
```

3. Download additional tool sources from these repos:

```
git clone https://github.com/01org/tinycbor.git \
extlibs/tinycbor/tinycbor -b v0.4.1

git clone https://github.com/ARMmbed/mbedtls.git \
extlibs/mbedtls/mbedtls -b mbedtls-2.4.2
```

Setting up MRAA library

MRAA is a C Linux library with bindings to C++, Java, Python, and Node.js (JavaScript) that lets you write portable code accessing low speed IO interfaces for sensors and actuators across a variety of hardware platforms. The mraa library is provided as sources that you'll build to create the needed runtime library. MRAA library doesn't support Eagleye 530s but you can control GPIO pin, if use 'raw=ture' option.

1. Clone the mraa source into the "iot" working directory we created earlier:

```
cd ~/iot

git clone https://github.com/intel-iot-devkit/mraa.git
```

2. Make a working directory in the mraa directory we've cloned, change to that directory, build the mraa library, and install it:

```
mkdir mraa/build && cd mraa/build

cmake .. && make && sudo make install
```

Note: You'll see some warnings reported during the cmake run that are expected and OK.

Building the IoTivity Server/Client Sample Applications

Now that all the required sources and tools are on the Eagleye 530s system, it's time to build the IoTivity sample application code.

✓ Currently, Code for controlling Eagleye 530s has not been updated in IoTivity official github yet. So You have to modify the code according to the method below.

```
Cd ~/iot/iotivity/examples/OCFSecure

nano SConscript -c
```

Code for being modified in SConsript file.

line # 87

```
        cpp_defines.append('LED_PIN=38')
```

```
86          elif artik:
87              cpp_defines.append('LED_PIN=38')
88              cpp_defines.append('RAW_GPIO')
```

to

```
        if eagleye:
                cpp_defines.append('LED_PIN=159')
            else:

          cpp_defines.append('LED_PIN=38')
```

```
88          elif artik:
89              if eagleye:
90                  cpp_defines.append('LED_PIN=159')
91              else:
92                  cpp_defines.append('LED_PIN=38')
93              cpp_defines.append('RAW_GPIO')
```

Line # 58

```
        artik = 'artik' in model or 'compy' in model
else:
```

```
57          model = f.read()
58          artik = 'artik' in model or 'compy' in model
59  else:
60      artik = False
```

to

```
        eagleye = 'compy' in model
        artik = 'artik' in model

else:

   eagleye = False
```

```
57          model = f.read()
58          eagleye = 'compy' in model
59          artik = 'artik' in model
60  else:
61      eagleye = False
62      artik = False
```

1. Change back to the iotivity directory and build the sample code using the “scons” tool:

```
cd ~/iot/iotivity

scons examples/OCFSecure -j 2 TARGET_TRANSPORT=IP
```

Here's an explanation of this command:

- **scons** is an open source software construction tool, an improved and more functional substitute for the classic "make" utility.
- The "`examples/OCFSecure`" parameter restricts the building process to the sample code directory and its dependencies.
- The "`-j 2`" flag will utilize two (of the four) processor cores available on the Eagleye 530s, improving performance.
- The "`TARGET_TRANSPORT=IP`" parameter restricts building to only implement the IP transport protocol and not others such as such as Bluetooth Low Energy (BLE) or Near Field Communication (NFC).

2. This build should take less than 10 minutes. Once you see the message: "`scons: done building targets.`", and there are no obvious errors, the build was successful and we can try running the OCFSecure sample server and client applications.

Running the IoTivity Server/Client Sample Applications

A pair of client and server application samples are included in the software for this kit to give you a working example of building a server for an OCF hardware "switch" device (in our case an LED), together with a client that interacts with this server.

Now, let's run the sample applications:

1. Change to the output directory where the sample application executable files were created. (Note the directory name armv7l ends with the lower-case letter "l" not a digit one.) Because the application directly accesses hardware, it needs to run as root (administrator) with the "sudo" command. Note too, you're going to run the server as a background app so we can run the client app in the foreground in the next step:

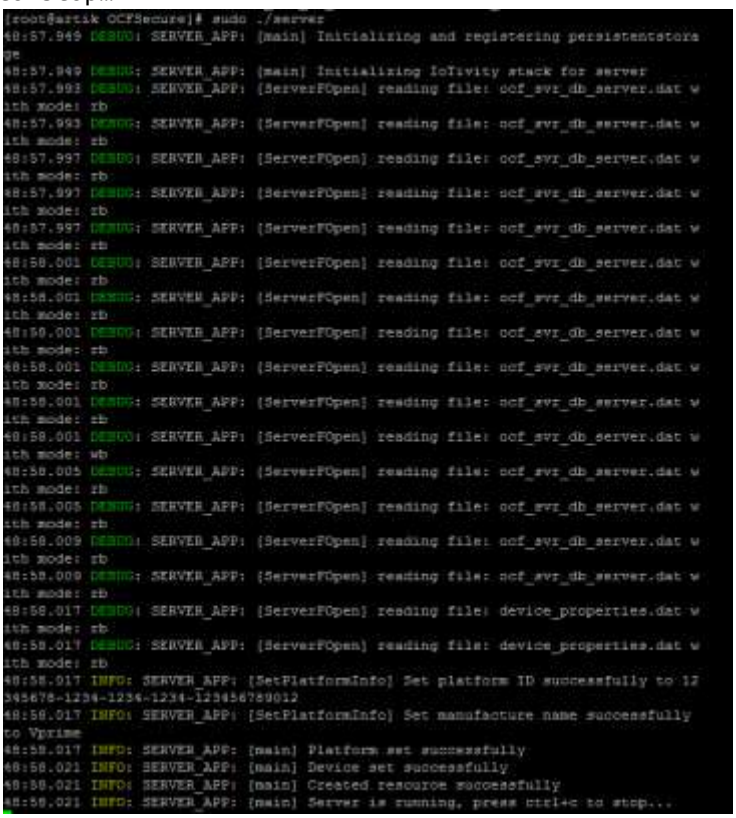
```
cd ~/iot/iotivity/out/linux/armv7l/release/examples/OCFSecure

sudo ./server &
```

If you forgot to add the trailing "&" to tell the shell to run the server in the background, press CTRL-C to end the server, and start the server app again with the trailing "&".

The server application will report that it's initializing and reading some files, creating the SWITCH resource (for the LED), and finally reporting, "`Server is running, press ctrl+c`

to stop…"



2. With the server app running in the background, you can run the client application in the foreground. Since the client doesn't (directly) access the hardware, it doesn't need to run with sudo (admin) access. If you don't see the command prompt, press return, and then type:

```
./client
```

Both the server and client applications will write INFO and DEBUG information to the console, identified as SERVER_APP or CLIENT_APP.

The client application has a very simple interface that lets you interact with resources via IoTivity API calls to the server. Using the client, you identify by <resource number>, the resource you want to interact with, together with a request to either GET (read) or POST (write) information to that resource.

If you press a return, the client displays the list of discovered resources. Press return again if you don't see the discovered /switch resource when the client first starts up. As shown below, the /switch resource was discovered as resource number 21:

```
============================================================
Discovered Resources:
        #0: /oic/res    @224.0.1.187:5683       resource type: nil
        #1: /oic/sec/doxm       @fe80::996e:7dc6:37f3:5311%eth0:57495   resource type: oic.r.doxm
        #2: /oic/sec/pstat      @fe80::996e:7dc6:37f3:5311%eth0:0       resource type: oic.r.pstat
        #3: /oic/sec/acl2       @192.168.1.24:57791     resource type: oic.r.acl2
        #4: /oic/sec/cred       @192.168.1.24:57791     resource type: oic.r.cred
        #5: /oic/sec/crl        @192.168.1.24:57791     resource type: oic.r.crl
        #6: /oic/sec/csr        @192.168.1.24:57791     resource type: oic.r.csr
        #7: /oic/sec/roles      @192.168.1.24:57791     resource type: oic.r.roles
        #8: /oic/d      @192.168.1.24:57791     resource type: oic.wk.d
        #9: /oic/p      @192.168.1.24:0 resource type: oic.wk.p
        #10: /introspection     @192.168.1.24:0 resource type: oic.wk.introspection
        #11: /oic/sec/doxm      @fe80::996e:7dc6:37f3:5311%eth0:54464   resource type: oic.r.doxm
        #12: /oic/sec/pstat     @fe80::996e:7dc6:37f3:5311%eth0:0       resource type: oic.r.pstat
        #13: /oic/sec/acl2      @192.168.1.24:43330     resource type: oic.r.acl2
        #14: /oic/sec/cred      @192.168.1.24:43330     resource type: oic.r.cred
        #15: /oic/sec/crl       @192.168.1.24:43330     resource type: oic.r.crl
        #16: /oic/sec/csr       @192.168.1.24:43330     resource type: oic.r.csr
        #17: /oic/sec/roles     @192.168.1.24:43330     resource type: oic.r.roles
        #18: /oic/d     @192.168.1.24:43330     resource type: oic.wk.d
        #19: /oic/p     @192.168.1.24:0 resource type: oic.wk.p
        #20: /introspection     @192.168.1.24:0 resource type: oic.wk.introspection
        #21: /switch    @192.168.1.24:43330     resource type: oic.r.switch.binary

Request Methods:        1: GET  4: POST
Usage:<resource number> <request method> :
```

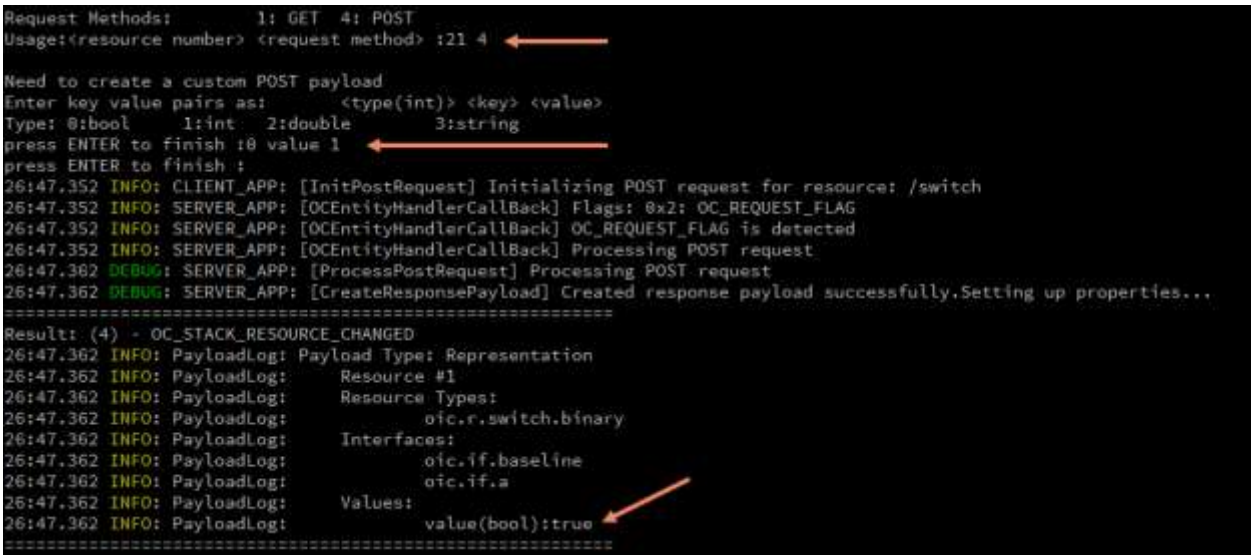
3. Send a GET request to the /switch resource by typing "21 1" and press return: 21 for the /switch resource and 1 for a GET request. After some INFO and DEBUG messages go by, you'll see the result as:

```
============================================================
Result: (0) - OC_STACK_OK
05:47.637 INFO: PayloadLog: Payload Type: Representation
05:47.637 INFO: PayloadLog:     Resource #1
05:47.637 INFO: PayloadLog:     Resource Types:
05:47.637 INFO: PayloadLog:             oic.r.switch.binary
05:47.637 INFO: PayloadLog:     Interfaces:
05:47.637 INFO: PayloadLog:             oic.if.baseline
05:47.637 INFO: PayloadLog:             oic.if.a
05:47.637 INFO: PayloadLog:     Values:
05:47.647 INFO: PayloadLog:             value(bool):false
============================================================
```

This response indicates the result was OC_STACK_OK and displays the payload returned. You can see the value of a Boolean property named "value" is false, indicating the LED is off.

(If you don't see this response on your screen, but you do see the "Discovered Resources" list, the response may have just scrolled off your screen. If you're connecting via ssh, try making your terminal screen taller, or simply scroll the windows up to see the output.)

4. Next, try changing the property "value" to turn the LED on, by sending a POST request to the /switch resource. Start by typing "21 4" and enter. You'll be prompted to create a custom payload to POST (send) to the server. Select a property type from the list (in our case Boolean, so 0), and enter the property key (in our case "value") and the property value (1 for true), and press enter. If the custom payload required multiple key/value pairs to POST, we could enter more, but for our example that's all, so press enter to finish:



Notice that the response message indicates the result is OC_STACK_RESOURCE_CHANGED and the values that were affected, namely the Boolean property "value" is now true (and the LED is on). You can verify this by repeating the GET request you previously did.

5. Exit the client application by pressing CTRL-C. Then, exit the server app running in the background, by using the "fg" command to bring the background job to the foreground and press CTRL-C to exit the server.

Shutting down the Eagleye 530s

Like all computers, just pulling the power plug on the Eagleye 530s is a bad way to turn it off, and can cause data loss on the microSD card. You must shut down the Eagleye 530s properly and give the operating system a chance to cleanly close down system services and the file system.

From the command line use this command:

```
sudo shutdown -h now
```

Then wait until the green LED near the micro-USB power connector on the Eagleye 530s stays off. At this point it's best to unplug the power supply from the wall rather than unplug the micro-USB connector – connectors like these have a limited service lifetime.

Wrap-up and Next Steps

With that, you're done setting up the IoTivity development environment on the Eagleye 530s and verified the sample applications can be built and run. Over time, we plan to add additional samples along with updated documentation to go with them, to give you more examples of using the IoTivity APIs.

Now it's up to you to explore the source code for the sample code (in the ~/iot/iotivity/examples/OCFSecure directory). Read through the source comments and the included README file for more information.

If you're looking for some ideas, try modifying the server app to add access to another device, for example returning the temperature or reading the light and color value detected by the on-board sensors.

OCF Membership Resources

The OCF basic membership level is a no-cost way to get read only rights for members-only materials and access the OCF Certification Test Tools (CTT) for pre-testing purposes. Visit the OCF membership page to learn more. If your company is already an OCF member, you can also get access to additional resources and participate in the many OCF work and task groups that are helping to create this interoperable and secure "network of everything" environment.

Appendix A: Headless access to the Eagleye 530s via Serial

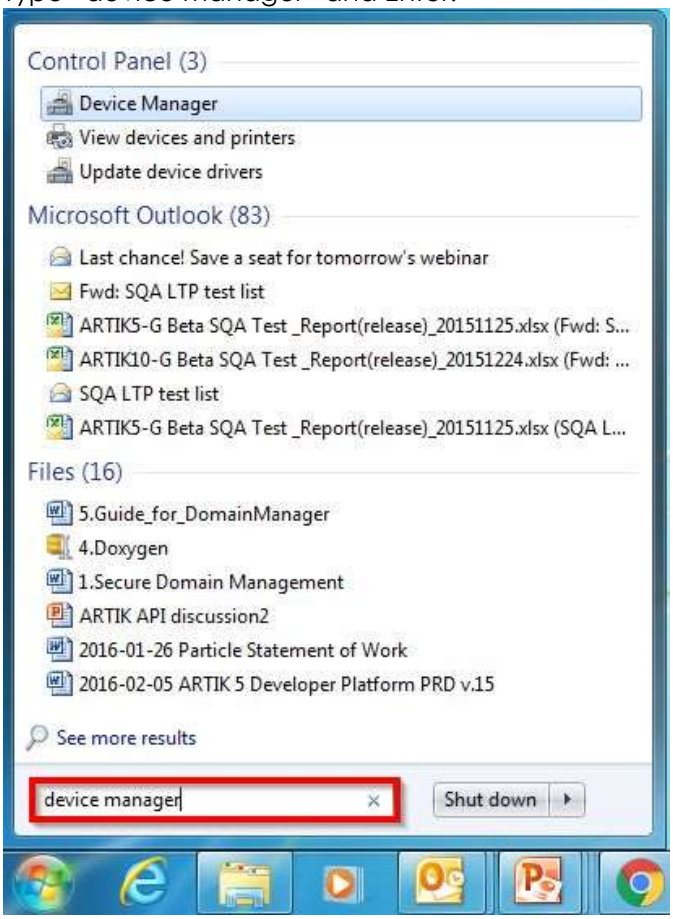
If it's not convenient to connect a monitor and keyboard to the Eagleye 530s , you can also get command line access via serial to the board from another computer

Finding the serial port number of the Eagleye 530s

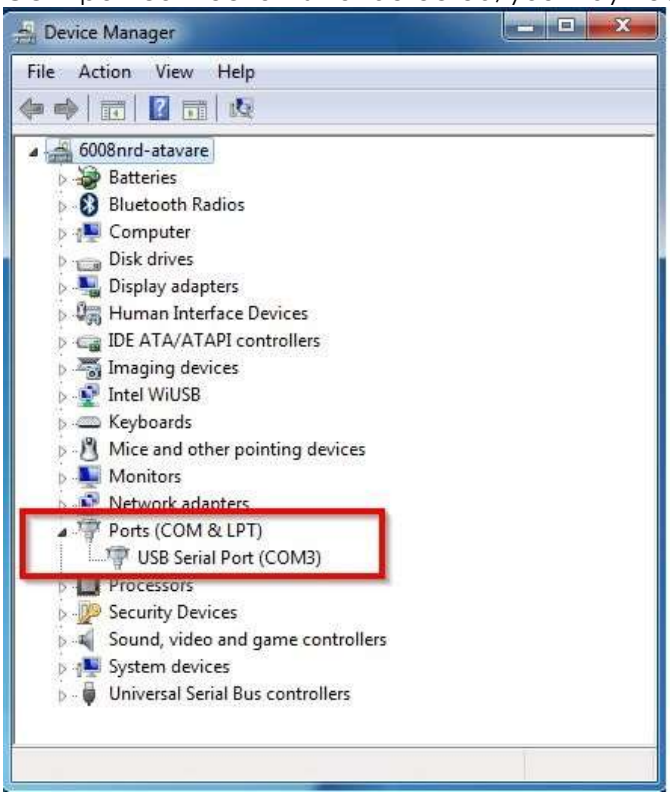
You'll need to determine the Serial(COM) port number assigned to your board. Start with the USB cable unattached, then watch Device Manager to see which device appears when you plug it in.

1. Open Device Manager in the Control Panel.

2. Type "device manager" and Enter.



3. Check the COM port number when you connect a USB serial cable to the PC. (If the COM port connection is not detected, you may need to power on the ARTIK board.)



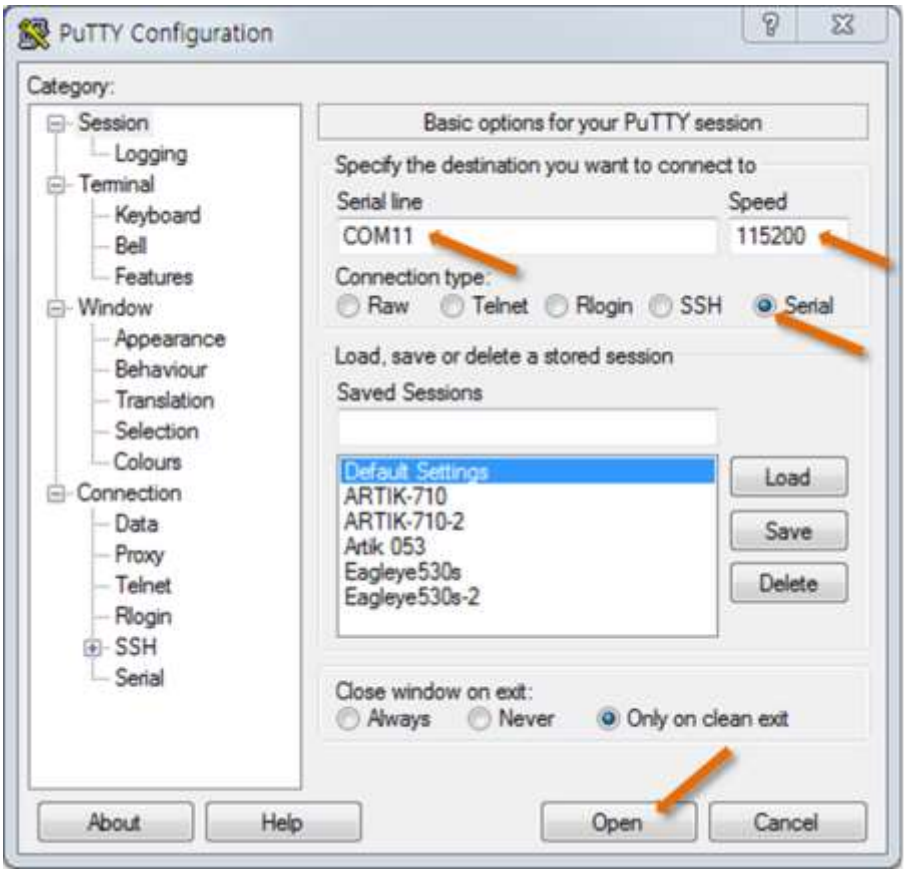
Using PuTTY on Windows, Linux and macOS

One of the most commonly used clients is PuTTY. If you're using Windows, can be downloaded from http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html. Look for the MSI Windows Installer links for your OS version (most likely 64-bit windows).
If you're using Linux or macOS, open Command line terminal and run the command below to install PuTTY.

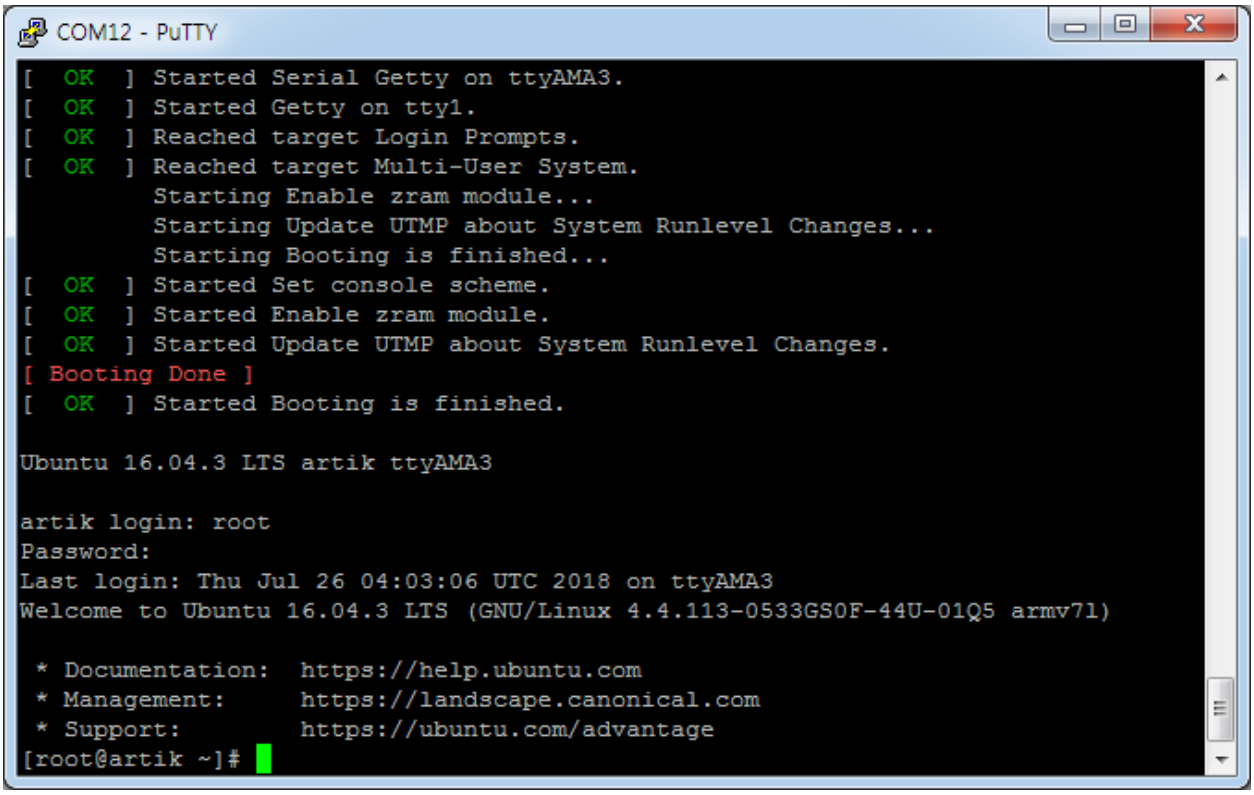
```
Sudo apt install putty
```

You can also read their online documentation for full information about PuTTY.

After installing PuTTY, launch the PuTTY application and input the Eagleye 530s USB serial port number in the Serial line field and 115200 in the Speed field, verify the Serial connection type is selected, and click open. (If you want to save current status, fill in the blank located below 'Saved Session' with the name you want and click 'Save' button. You can load the previously saved status by selecting the name and click the 'load' button.)



Next you'll be prompted for the username (root) and password (root, unless you changed it), and you'll be connected:



Appendix B: Firmware update

If you want to update firmware on Eagleye 530s, you have to prepare microSD card(over 2GB is recommended) preloaded with a bootable Ubuntu Lite image (without a graphical desktop). The following instructions will show how you can create your own microSD card with the same bootable image.

Creating the microSD card for firmware update

Download the latest firmware Image

Because Eagleye 530s is an ARTIK 530s system-on-module based board, it is recommended to install an ARTIK 530s specialized Ubuntu. Official Ubuntu OS images are available from the https://developer.artik.io/documentation/downloads.html web page. On your Windows, macOS, or Linux host, use your web browser to download the ZIP file for Ubuntu Stretch Lite:

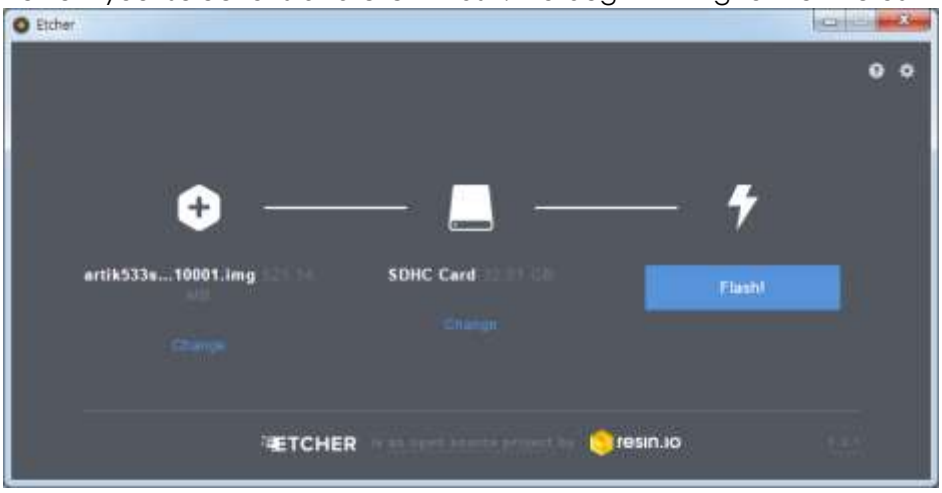
Write the image to the microSD card

You'll need special image burning software to write this file to the microSD card in a way that creates a bootable image. As described in the ARTIK 530s image update guide, we recommend using a free open-source graphical SD card writing tool called Etcher. This tool runs on Windows, macOS, and Linux systems and is available for download from http://etcher.io.

Here's a summary of image burning instructions:

1. Download Etcher and install it on your host computer.
2. Connect a microSD card to your host computer with an SD or USB carrier or an external adapter. If you're using an SD-card carrier, verify the write-protect switch is not enabled.
3. Open Etcher and select the .ZIP file you downloaded with the Ubuntu image as the source.
4. Select the SD card you're writing to as the destination. Note you'll be completely overwriting the card's contents. If Etcher can't find an SD card to write to, you may need to reformat the microSD card. (Instructions for doing a low-level format for an SD card can be found on the SD Association website.)

5. Review your selections and click "Flash!" to begin writing to the microSD card.

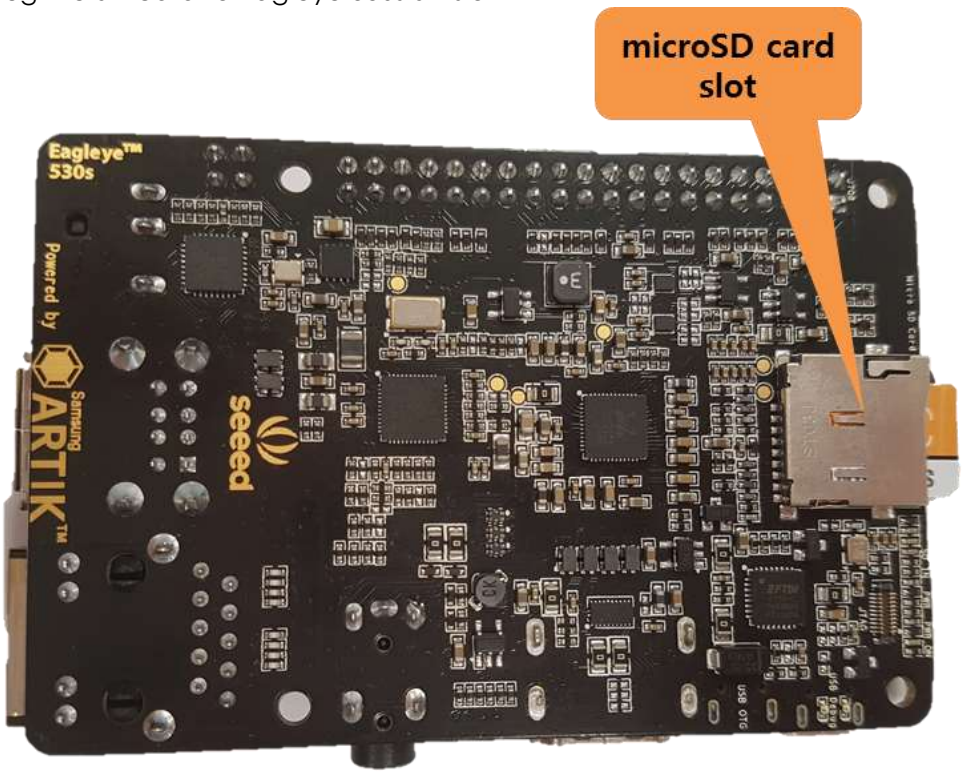


6. When done, exit Etcher, unmount or eject the microSD card, and remove it from your computer.Write the image to the microSD card

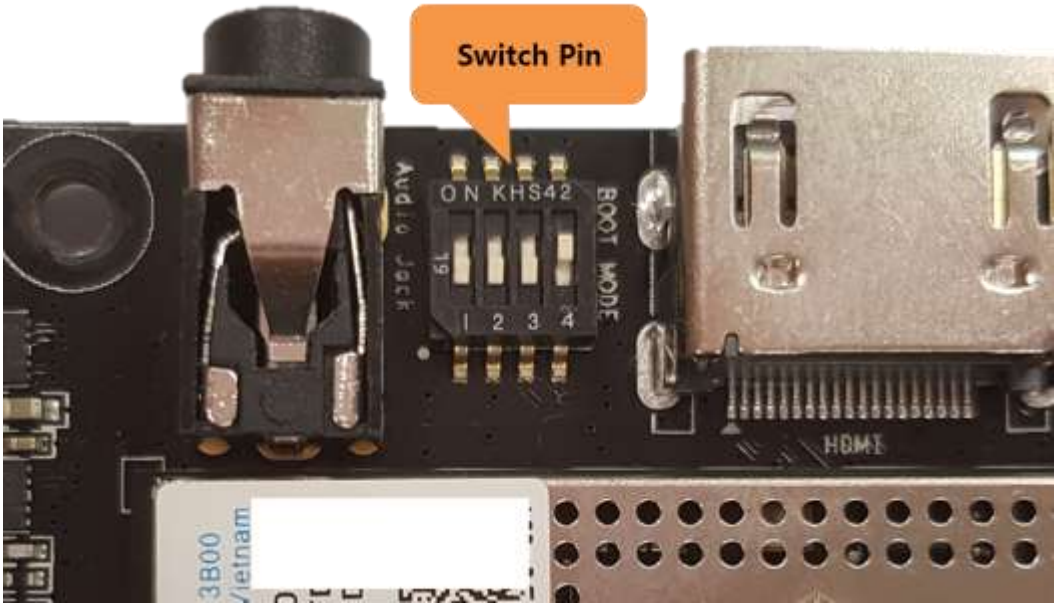
Firmware update with microSD card

Since the microSD card for firmware update has been completed, Let' update using it.

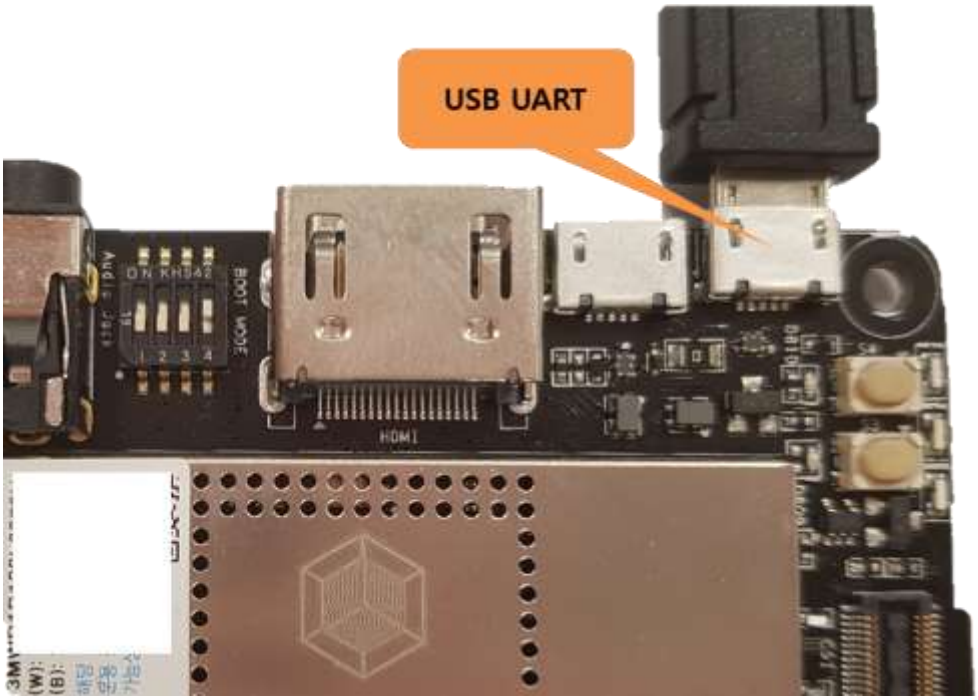
1. Plug the SD card to Eagleye 530s SD slot



2. Change boot switch Pin4 to on.

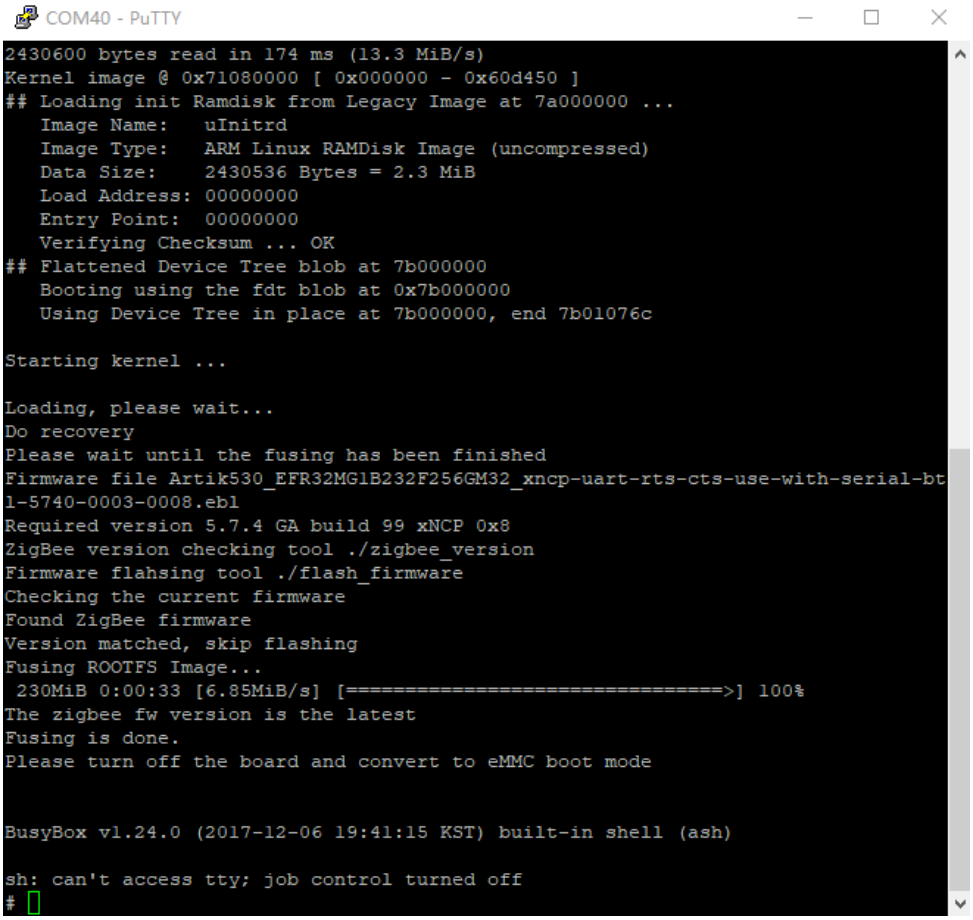


3. Connect the USB cable from PC to USB UART



4. Press the Power button for more than 1 second and we will see the Green/Red/Blue LEDs are turned on

5. We will see below message and the firmware is downloaded successfully.



6. Please shutdown the board, remove SD card and switch boot switch Pin4 back to off.

Appendix C: Connect Wi-Fi Network

The below instructions discuss setting up a wired or wireless local area network (LAN). The Ethernet LAN (hard-wired) port is always available. A wireless LAN (WLAN) circuit is separately available, and can be configured by Connection Manager.

1. Enter commands as follows. 'connmanctl' to get the > prompt.
2. 'scan wifi' to scan for available access points (wait until finished).
3. 'services' to list them.
4. 'agent on' if you want to Connection manager to prompt you for a password.
5. 'connect wifi_xxxx' to pick the desired access point(You can use the tab key to start, and to auto-complete, your entry)
6. Respond to the agent query for a password if needed.
7. 'quit' when finished.
8. Connection happens automatically from now on, you won't need to repeat this process in the future.

```
[root@artik ~]# connmanctl
Error getting VPN connections: The name net.connman.vpn was not provided by
any connmanctl> scan wifi
Scan completed for wifi
connmanctl> scan wifi
connmanctl> services
*AO Wired                ethernet_000000000000_cable
    seeed                wifi_722c1f37ca11_XXXXXXXX_managed_psk
    ReSpeaker1DD346      wifi_722c1f37ca11_XXXXXXXX_managed_none
connmanctl> agent on
Agent registered
connmanctl> connect wifi_722c1f37ca11_XXXXXXXX_managed_psk
Agent RequestInput wifi_722c1f37ca11_XXXXXXXX_managed_psk
  Passphrase = [ Type=psk, Requirement=mandatory ]
Passphrase? 2018seeed
connmanctl> quit
[root@artik ~]#
```