

Getting Started with M.2 Modules and i.MX 6/7/8 from Linux v5.4



Embedded Artists AB

Jörgen Ankersgatan 12
SE-211 45 Malmö
Sweden

<http://www.EmbeddedArtists.com>

Copyright 2020 © Embedded Artists AB. All rights reserved.

No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language or computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual or otherwise, without the prior written permission of Embedded Artists AB.

Disclaimer

Embedded Artists AB makes no representation or warranties with respect to the contents hereof and specifically disclaim any implied warranties or merchantability or fitness for any particular purpose. Information in this publication is subject to change without notice and does not represent a commitment on the part of Embedded Artists AB.

Feedback

We appreciate any feedback you may have for improvements on this document. Send your comments by using the contact form: www.embeddedartists.com/contact.

Trademarks

All brand and product names mentioned herein are trademarks, services marks, registered trademarks, or registered service marks of their respective owners and should be treated as such.

Table of Contents

1 Document Revision History	5
2 Introduction	6
2.1 Conventions.....	6
3 QuickStart Guide.....	7
3.1 Follow "Getting Started" Guide.....	7
3.2 Mount M.2 Module	7
3.3 Power-up Board.....	9
3.3.1 iMX uCOM Boards with On-Board Solutions	10
3.4 Linux Console – Search for available networks	10
3.5 Access and Configure Bluetooth Devices	11
3.5.1 Additional Links	14
3.6 Wi-Fi: iperf3 Test	14
3.7 Wi-Fi: Check the Linux Boot Log	15
3.8 Wi-Fi: HostAP	16
3.8.1 Setup hostapd for 1CX, 1DX, 1LV, 1MW, 1XA.....	16
3.8.2 Setup for 1YM-PCIe, 1ZM	16
3.8.3 Connecting with a client.....	17
3.8.4 Example - iperf3 on Android	19
3.8.5 Where to go from here?.....	20
3.9 Bluetooth: keyboard	22
4 COM Carrier Board V2 Advanced Features	24
4.1 VBAT Current Measurement.....	24
4.2 VBAT 3.3V or 3.6V	25
4.3 Support for 3.3V IO logic level (if M.2 module supports it).....	25
4.4 Bluetooth UART Interception	25
4.5 Dual UART Debug Channels and JTAG.....	27
4.6 Audio Codec Multiplexing	27
5 Software Update.....	28
5.1 Linux Host Setup.....	28
5.1.1 Introduction.....	28
5.1.2 Required Packages	28
5.1.3 Install the repo tool.....	28
5.1.4 Download Yocto recipes.....	29
5.2 Building Images.....	30
5.2.1 Available Images	30
5.2.2 Machine Configurations.....	30
5.2.3 Initialize Build	30
5.2.4 Starting the Build	31
5.3 Deploying Images.....	31

5.3.1.1	Download the Tool	32
5.3.1.2	Prepare hardware	32
5.3.1.3	OTG boot mode – J2 jumper.....	32
5.3.1.4	Configurations.....	33
5.3.1.5	Download Your Own Images.....	33
5.3.1.6	Run the Tool in Ubuntu.....	33
5.3.1.7	Run the Tool in Windows	34
5.3.1.8	Troubleshoot.....	34
5.4	Building without Yocto	36
5.4.1	Stand-alone Toolchain	36
5.4.2	Build Linux kernel from source code	37
5.4.3	Build u-boot from source code.....	38
5.5	NVRAM	39
5.6	Firmware	39
6	Appendix - Updating Files on Target	40
6.1	U-boot USB Mass Storage Gadget.....	40
6.2	Secure Copy from Target.....	41
6.3	Secure Copy To Target - WinSCP	42
6.3.1	Download and Install	42
6.3.2	Connect to Target.....	42
6.3.3	Copy Files	45
6.4	USB Memory Stick	45

1 Document Revision History

<i>Revision</i>	<i>Date</i>	<i>Description</i>
A	2020-10-30	Initial release

2 Introduction

This document describes how to add wireless functionality with M.2 modules to an *iMX Developer's Kit V2*. Linux commands for controlling wireless functionality are also presented.

This document is **only** valid for our Linux v5.4 release, and further. There is a similar document that is valid up to, and including our Linux v4.14 release.

There are several different *iMX Developer's Kits V2* and this document refers to all of these kits collectively as *iMX Developer's Kits*. Please note that all available *iMX Developer's Kits* may not support all the presented wireless technologies or more specifically the interface used to communicate with a hardware module. The PCIe interface is for example not supported by all i.MX processors.

All interfaces, needed tools and kernel configurations described in this document have been added / enabled in the **prepared images** available at <http://imx.embeddedartists.com/>. To make changes to your own build, see the [iMX Working with Yocto document](#) which can be downloaded on each COM board's product page.

Additional documentation you might need is.

- The *Getting Started* document for the *iMX Developer's Kit* you are using
- *COM Board Datasheet* for the specific COM board you are using
- *COM Carrier Board Datasheet*
- *M.2 Module Datasheet* for the specific M.2 module you are using

2.1 Conventions

A number of conventions have been used throughout to help the reader better understand the content of the document.

Constant width text – is used for file system paths and command, utility and tool names.

```
$ This field illustrates user input in a terminal running on the  
development workstation, i.e., on the workstation where you edit,  
configure and build Linux
```

```
# This field illustrates user input on the target hardware, i.e.,  
input given to the terminal attached to the COM Board
```

```
This field is used to illustrate example code or excerpt from a  
document.
```

This field is used to highlight important information

3 QuickStart Guide

This chapter is a step-by-step guide to get Wi-Fi and Bluetooth connections up in shortest possible time:

1. The first step address basic steps like powering the board and getting access to the console.
2. The second step describes where and how to physically mount the M.2 module.
3. The third step describes how to power up the system and boot with correct setting (given the M.2 module that is mounted).
4. The fourth step describe how to manually setup the Wi-Fi interface from the Linux console.
5. The fifth step describe how to access and configure the Bluetooth interface.

Above are the five simple steps to get up-and-running immediately!

There are a couple of more sections describing different aspects, like performance measurements with perf and how to connect specific Bluetooth devices, like a keyboard.

3.1 Follow "Getting Started" Guide

Follow the instructions in the getting started guide <https://www.embeddedartists.com/getting-started-with-imx-developers-kit-v2/> to get access to the console and to connect the power supply but keep the board powered off for now.

3.2 Mount M.2 Module

Make sure the iMX Developer's Kit is **powered off** and then mount the M.2 Module, as illustrated in the picture below:

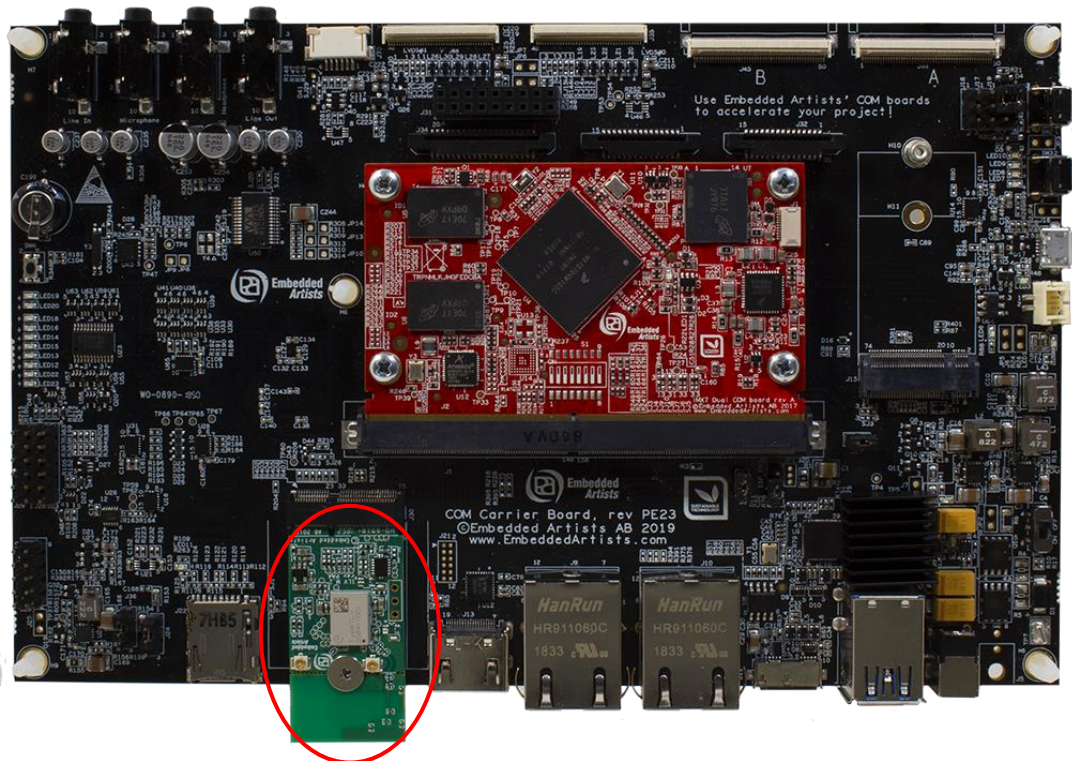


Figure 1 – M.2 Module on COM Carrier Board V2

Do not put too much force/pressure on the M.2 screw (into the M.2 connector stand-off) so that the PCB is bent. Bending the PCB too much will damage the board!
Use your fingers on the bottom side (while screwing) to give a counter-force, keeping the PCB straight.

The picture below illustrates the typical angle (about 25 degrees) to use when inserting the M.2 module into the connector.

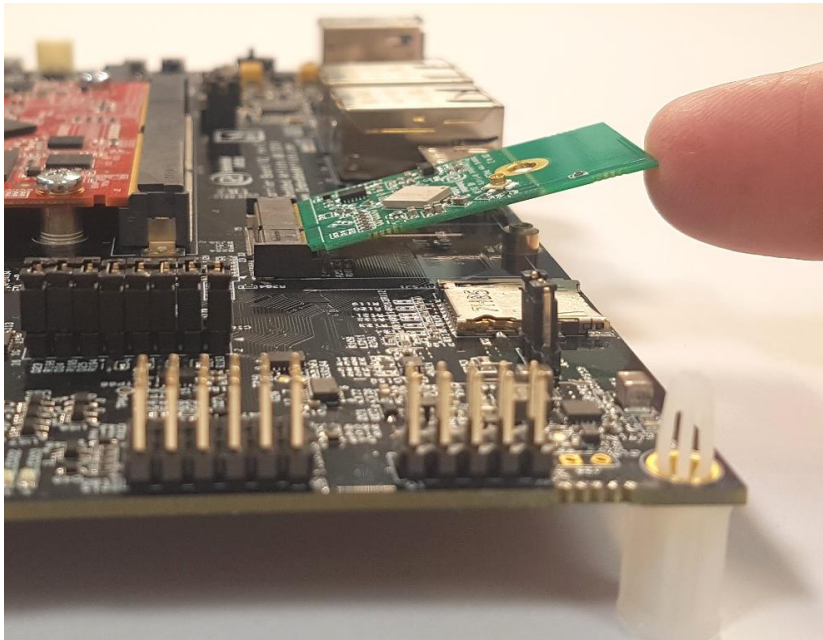


Figure 2 – M.2 Module on COM Carrier Board V2

The picture below illustrates how to use two funders, placed under the grounding stand-off, to avoid bending the board. Make sure to always use this method to avoid damaging the board.

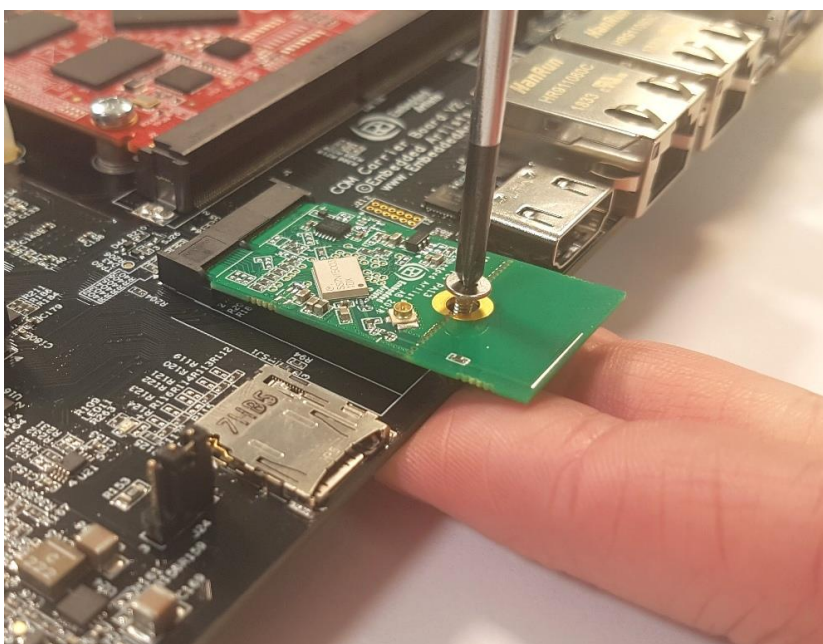


Figure 3 – M.2 Module on COM Carrier Board V2

3.3 Power-up Board

Set the Power on/off switch to On state to power up the board. When the boot process is complete you will be presented with a login prompt. Enter the login credentials below to log in:

Username: root

Password: pass

The M.2 modules use one of two possible interfaces, as listed below

- SDIO interface: 1DX, 1LV, 1MW, 1WZ, 1YM-SDIO, 1ZM
- PCIe interface: 1CX, 1XA, 1YM-PCIE

Prior to Linux 5.4.24 the configuration of interface had to be done by selecting the correct device tree file in the u-boot but that has been simplified with a couple of scripts in Linux 5.4.24.

The first script is `switch_module.sh` that will select device tree file, configure libraries/utilities and enable the correct systemd service based on which M.2 module will be used.

Run the script without parameters to see the full list of available module names

```
# switch_module

Version: 1.0

Usage:
  /usr/sbin/switch_module.sh <module>

Where:
  <module> is one of (case insensitive):
    CYW-SDIO, CYW-PCIE, 1CX, 1DX, 1LV, 1MW, 1XA, 1ZM, 1WZ
    1YM-SDIO, 1YM-PCIE, CURRENT or OFF
```

CYW-SDIO is an alias for 1DX, 1LV, 1MW and 1WZ modules. CYW-PCIE is an alias for 1CX and 1XA. The OFF parameter will disable the systemd service so that it will not automatically start after a reboot but it will keep the rest of the system configuration.

As an example, to enable the 1ZM M.2 module:

```
# switch_module 1ZM
```

The only thing left is to enter the SSID and password information. Open the `/etc/wpa_supplicant.conf` file in an editor (e.g. vi or nano) and replace the part marked in yellow below:

```
ctrl_interface=/var/run/wpa_supplicant
ctrl_interface_group=0
update_config=1

network={
    key_mgmt=NONE
}
```

If the SSID is Hello World and the password is MyPassword then change the file to

```
ctrl_interface=/var/run/wpa_supplicant
ctrl_interface_group=0
update_config=1
```

```
network={
    ssid="Hello World"
    psk="MyPassword"
}
```

Now that the network configuration is completed, reboot to test the new settings. The M.2 module should automatically connect to the desired network.

3.3.1 iMX uCOM Boards with On-Board Solutions

At the time of writing this document, Embedded Artists supplied three iMX uCOM boards with onboard chips. These boards requires extra configuration to use as they use different device tree files.

iMX7 ULP uCOM with onboard 1LV

```
# switch_module 1LV
# fw_setenv fdt_file imx7ulpea-ucm-kit_v2-1lv.dtb
```

iMX8M Mini uCOM with onboard 1MW

```
# switch_module 1MW
# fw_setenv fdt_file imx8mm-ea-ucm-kit_v2-1mw.dtb
```

iMX8M Nano uCOM with onboard 1MW

```
# switch_module 1MW
# fw_setenv fdt_file imx8mn-ea-ucm-kit_v2-1mw.dtb
```

3.4 Linux Console – Search for available networks

When the boot process is complete you will be presented with a login prompt. Enter the login credentials below to log in:

Username: root

Password: pass

The Wi-Fi modules use different device names. Most of them use wlan0 which is what all examples in this document will use. If you have a 1ZM, 1YM-PCIE or 1YM-SDIO then replace wlan0 with mlan0. The places will be marked with cyan.

Run a scan of networks in range:

```
# iw dev wlan0 scan
```

Or

```
# iwlist wlan0 scan
```

Both commands are very verbose so it is probably a good idea to limit the result like this:

```
# iw dev wlan0 scan | grep SSID
      SSID: EA Guest
      SSID: VSG1
      SSID: COMHEM_73ee11
      SSID: DIRECT-4C-HP ENVY Photo 6200
      ...
```

Test the network connection with ping (stop with Ctrl+C):

```
# ping www.sunet.se
PING www.sunet.se (192.36.171.231): 56 data bytes
64 bytes from 192.36.171.231: seq=0 ttl=56 time=16.412 ms
64 bytes from 192.36.171.231: seq=1 ttl=56 time=18.279 ms
64 bytes from 192.36.171.231: seq=2 ttl=56 time=19.125 ms...
```

If your router is not connected to Internet then ping the IP number of the router instead. The IP number can be found like this:

```
# ip route
default via 192.168.0.1 dev wlan0 metric 10
192.168.0.0/24 dev wlan0 proto kernel scope link src 192.168.0.4

# ping 192.168.0.1
PING 192.168.0.1 (192.168.0.1): 56 data bytes
64 bytes from 192.168.0.1: seq=0 ttl=56 time=16.412 ms
64 bytes from 192.168.0.1: seq=1 ttl=56 time=18.279 ms
64 bytes from 192.168.0.1: seq=2 ttl=56 time=19.125 ms...
```

The wireless connection will now work until next reboot. As the wireless network name and password has been saved, all that is needed to connect to the network after a reboot is this:

```
#wpa_supplicant -B -i wlan0 -D nl80211 -c /etc/wpa_supplicant.conf
# udhcpc -i wlan0
```

There are several ways to see the status of your connection and the network it is connection to. Here is one example (not showing the output):

```
# iw dev wlan0 link
```

3.5 Access and Configure Bluetooth Devices

All the Bluetooth devices use the UART interface but the device name for that interface is different depending on the COM Board. The M.2 modules require some device specific instructions to initialize and that has been simplified with the addition of the `/opt/ea/Bluetooth_up.sh` script in Linux 5.4.24.

You must follow the instructions in 3.3 and select the correct M.2 module before trying anything in this section.

To bring up the Bluetooth interfaces run this command:

```
# /opt/ea/bluetooth_up.sh

Setting TTY to N_HCI line discipline
Device setup complete
< HCI Command: ogf 0x3f, ocf 0x0009, plen 4
  C0 C6 2D 00
> HCI Event: 0x0e plen 4
  01 56 0C 00
[ 3065.661301] Bluetooth: hci0: sending frame failed (-49)
Setting TTY to N_HCI line discipline
Device setup complete
Scanning ...

To run a scan again, use hcitool scan
```

The output from the command is different and depends on which module is being used.

Note that there are two error messages that can appear for the 1ZM and 1YM-PCIe modules that can be safely ignored (the second one is found in the output above):

“ Can't init device hci0: Invalid argument (22) “ or

“ Bluetooth: hci0: sending frame failed (-49) ”

The module should now be up and running and to scan for other Bluetooth devices in range:

```
# hcitool scan
Scanning ...
          94:87:0B:35:F2:19          Samsung Galaxy S7
```

Use the interactive `bluetoothctl` command line tool to search for devices, pair with a device (the Galaxy Nexus phone), connect to it and then find some information about it:

```
# bluetoothctl
[bluetooth] power on
[bluetooth] agent on
[bluetooth] scan on
Discovery started
[CHG] Controller D8:FC:93:E4:1E:A6 Discovering: yes
[NEW] Device B0:D0:00:38:00:C6 Galaxy Nexus
[NEW] Device 40:2B:A1:5F:7F:46 MW600
[bluetooth] pair B0:D0:00:38:00:C6
Attempting to pair with B0:D0:00:38:00:C6
[CHG] Device B0:D0:00:38:00:C6 Connected: yes
[CHG] Device B0:D0:00:38:00:C6 Modalias: bluetooth:v000Fp1200d1436
[CHG] Device B0:D0:00:38:00:C6 UUIDs:
        00001105-0000-1000-8000-00805f9b34fb
        0000110a-0000-1000-8000-00805f9b34fb
        0000110c-0000-1000-8000-00805f9b34fb
        00001112-0000-1000-8000-00805f9b34fb
        00001115-0000-1000-8000-00805f9b34fb
        00001116-0000-1000-8000-00805f9b34fb
        0000111f-0000-1000-8000-00805f9b34fb
        0000112f-0000-1000-8000-00805f9b34fb
        00001200-0000-1000-8000-00805f9b34fb
[CHG] Device B0:D0:00:38:00:C6 Paired: yes
Pairing successful
[bluetooth] scan off
[bluetooth] devices
Device B0:D0:00:38:00:C6 Galaxy Nexus
[bluetooth]# connect B0:D0:9C:38:84:C6
Attempting to connect to B0:D0:9C:38:84:C6
[CHG] Device B0:D0:9C:38:84:C6 Connected: yes
Connection successful
[bluetooth]# info B0:D0:00:38:00:C6
Device B0:D0:00:38:00:C6
    Name: Galaxy Nexus
    Alias: Galaxy Nexus
    Class: 0x5a020c
    Icon: phone
    Paired: yes
    Trusted: yes
    Blocked: no
    Connected: yes
    LegacyPairing: no
    UUID: OBEX Object Push (00001105-0000-1000-8000-00805f9b34fb)
    UUID: Audio Source (0000110a-0000-1000-8000-00805f9b34fb)
    UUID: A/V Remote Control Target (0000110c-0000-1000-8000-00805f9b34fb)
    UUID: Headset AG (00001112-0000-1000-8000-00805f9b34fb)
    UUID: PANU (00001115-0000-1000-8000-00805f9b34fb)
    UUID: NAP (00001116-0000-1000-8000-00805f9b34fb)
    UUID: Handsfree Audio Gateway (0000111f-0000-1000-8000-00805f9b34fb)
    UUID: Phonebook Access Server (0000112f-0000-1000-8000-00805f9b34fb)
    UUID: PnP Information (00001200-0000-1000-8000-00805f9b34fb)
    Modalias: bluetooth:v000Fp1200d1436
[bluetooth] quit
```

Using the `sdptool` command it is possible to find even more information (only showing first part here):

```
# sdptool browse B0:D0:00:38:00:C6
Browsing B0:D0:00:38:00:C6 ...
Service Name: Headset Gateway
Service RecHandle: 0x10000
Service Class ID List:
    "Headset Audio Gateway" (0x1112)
    "Generic Audio" (0x1203)
Protocol Descriptor List:
    "L2CAP" (0x0100)
    "RFCOMM" (0x0003)
        Channel: 2
Profile Descriptor List:
    "Headset" (0x1108)
        Version: 0x0102
```

3.5.1 Additional Links

https://wiki.archlinux.org/index.php/Bluetooth_headset

<https://wiki.archlinux.org/index.php/Bluetooth>

3.6 Wi-Fi: iperf3 Test

Ping is a great way to test if the hardware is connected to the network, or not, but to really test the network interface it is better to use a program like `iperf3`. The program works with a client and a server. The client is typically run on the COM board and the server software can either be installed on a computer on the local network (<https://iperf.fr/iperf-download.php>) or one of the online servers can be used (<https://iperf.fr/iperf-servers.php>).

To run the test first start the server by running the program with the `-s` switch. On a server running Linux the command looks like this:

```
$ iperf3 -s
-----
Server listening on 5201
-----
```

To improve the performance of `iperf3`, run the helper script `/opt/ea/optimize_for_iperf.sh`:

```
# /opt/ea/optimize_for_iperf.sh
Done with configuration
Run performance test with:

iperf3 -c 192.168.50.2 -i 5 -t 20 -P 4
iperf3 -c bouygues.iperf.fr -i 5 -t 20 -P 4
iperf3 -c ping.online.net -i 5 -t 20 -P 4
iperf3 -c speedtest.serverius.net -p 5002 -i 5 -t 20 -P 4
iperf3 -c iperf.eenet.ee -p 5201 -i 5 -t 20 -P 4
iperf3 -c iperf.volia.net -p 5201 -i 5 -t 20 -P 4

or another server from https://iperf.fr/iperf-servers.php
```

It will change the CPU frequency governor to performance mode to achieve maximum performance. It will also (for supported M.2 modules) set the minimum power consumption mode, set the driver power management mode to constantly awake, enable frame burst and prevent all scans. The changes are not permanent.

The next step is to run the client on the target hardware:

```
# iperf3 -c 192.168.1.128 -p 5201 -i 1 -P 4
```

The most important parameter is the url or ip number of the server, in this case 192.168.1.128 and the port number reported by the server, in this case 5201. There are a lot of options that can be given to the program. Use the `--help` option to see them all.

The client prints a lot during the test phase and in the end it prints a summary like this:

[ID]	Interval		Transfer	Bitrate	Retr	
[5]	0.00-10.00	sec	8.76 MBytes	7.35 Mbits/sec	1	sender
[5]	0.00-10.00	sec	8.67 MBytes	7.27 Mbits/sec		receiver
[7]	0.00-10.00	sec	10.5 MBytes	8.81 Mbits/sec	1	sender
[7]	0.00-10.00	sec	10.2 MBytes	8.54 Mbits/sec		receiver
[9]	0.00-10.00	sec	8.36 MBytes	7.02 Mbits/sec	4	sender
[9]	0.00-10.00	sec	8.20 MBytes	6.88 Mbits/sec		receiver
[11]	0.00-10.00	sec	8.55 MBytes	7.17 Mbits/sec	1	sender
[11]	0.00-10.00	sec	8.47 MBytes	7.11 Mbits/sec		receiver
[SUM]	0.00-10.00	sec	36.2 MBytes	30.4 Mbits/sec	7	sender
[SUM]	0.00-10.00	sec	35.5 MBytes	29.8 Mbits/sec		receiver

The last two lines display the bandwidth for send (30.4Mbit/sec) and receive (29.8Mbit/sec). Note that this number is limited by several factors: max bandwidth of the COM board's CPU, any network switches, network card in the PC and the PC performance. The summary above is actually from a test against an online server so the internet connection also limits the speed.

3.7 Wi-Fi: Check the Linux Boot Log

Start by checking that the module has been detected. For 1CX, 1DX, 1LV, 1MW, 1XA and 1WZ:

```
# dmesg | grep brcm
brcmfmac: brcmf_c_preinit_dcmds: Firmware version = wl0: May 14
2018 04:48:55 version 13.10.271.107 (r689896) FWID 01-9d634183
```

For 1YM-PCIe, 1YM-SDIO and 1ZM:

```
# dmesg |grep -i "wlan"
wlan: Loading MWLAN driver
wlan: Enable TX SG mode
wlan: Enable RX SG mode
Wlan: FW download over, firmwarelen=537492 downloaded 537492
WLAN FW is active
wlan: version = SD8987---16.92.10.p73-MX4X16169-GPL- (FP92)
wlan: Driver loaded successfully
```

If the command does not return something similar to the lines above then make sure that 1) the module is inserted correctly and 2) that the correct device was chosen when running `switch_module.sh`.

3.8 Wi-Fi: HostAP

Previous sections have described how to connect to a wireless network as a client. In this section we will instead create our own network that other clients can connect to. The way to do this depends on which M.2 module is being used so the instructions have been split into separate chapters below (3.8.1 and 3.8.2).

The chapters about connecting with a client and using iperf3 are the same for all modules.

3.8.1 Setup hostapd for 1CX, 1DX, 1LV, 1MW, 1XA

For these modules we use `hostapd` (host access point daemon) which enables a network interface card to act as an access point and authentication server. We will also use `udhcpd` (a DHCP daemon) to assign IP addresses to connecting clients.

The code below will be for the 1CX module so replace 1CX with your module. The instructions will result in an unprotected wireless network called **test** and the client will be assigned an IP number in the 192.168.5.100 to 192.168.5.150 range.

```
# switch_module 1CX
# /opt/ea/autostart_hostapd.sh enable
# reboot
```

The new wireless network will automatically be available after the reboot.

To stop using hostapd and go back:

```
# /opt/ea/autostart_hostapd.sh disable
# switch_module 1CX
# reboot
```

What happens in the background?

- The `/etc/hostapd.conf` file is modified to use `wlan1` as network interface. You can modify the `ssid=test` line in this file to set a different network name.
- The `/etc/udhcpd.conf` file is copied and modified to use `wlan1` as network interface and to change to the wanted IP range
- An `hostapd@wlan1` systemd service is enabled (will start after a reboot) that will handle starting of hostapd as well as udhcpd

3.8.2 Setup for 1YM-PCIe, 1ZM

For these modules we use `uaputl` which enables a network interface card to act as an access point and authentication server. We will also use `udhcpd` (a DHCP daemon) to assign IP addresses to connecting clients.

The code below will be for the 1ZM module so replace 1ZM with your module. The instructions will result in an unprotected wireless network called **Test_SSID** and the client will be assigned an IP number in the 192.168.5.100 to 192.168.5.150 range.

```
# switch_module 1ZM
# /opt/ea/autostart_hostapd.sh enable
# reboot
```

The new wireless network will automatically be available after the reboot.

To stop using hostapd and go back:

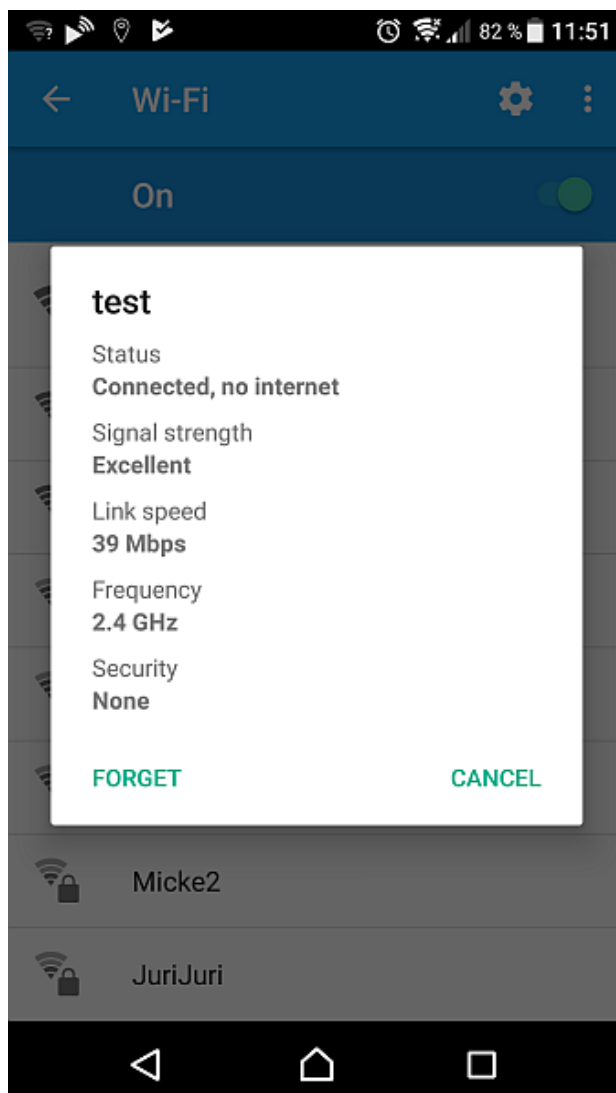
```
# /opt/ea/autostart_hostapd.sh disable
# switch_module 1ZM
# reboot
```

What happens in the background?

- The `/etc/udhcpd.conf` file is copied and modified to use `uap0` as network interface and to change to the wanted IP range
- An `hostapd@uap0` systemd service is enabled (will start after a reboot) that will handle starting of `uaputl` as well as `udhcpd`
- To change network name replace `Test_SSID` in `/opt/ea/start_hostapd.sh`

3.8.3 Connecting with a client

Use a phone/laptop/tablet to search for available networks and the "test"/"Test_SSID" (or the ssid you entered above) network should appear. It might look like this on a phone:



Note that the status "Connected, no internet" appears as there is no route for the traffic from the phone to Internet. It is possible (but out of scope for this document) to route the traffic to, for example, a wired network connection on iMX COM Boards with wired network interface(s).

Some phone models/version show the assigned IP number in this dialog but the example phone does not. If the default settings are used then the assigned IP number will be 192.168.5.100.

To check which IP addresses have been assigned run the following command:

```
# dumpleases

Mac Address          IP Address          Host Name          Expires in
aa:5f:1b:40:22:ad 192.168.1.100      Samsung-Galaxy-S7  expired
```

A bit more information can be found in the system log (here the first 4 lines come when connecting and the last line when the client leaves):

```
# cat /var/log/messages|grep -E "hostapd|udhcpd"

Oct 31 13:41:52 imx8mmea-ucom daemon.info hostapd: wlan1: STA
aa:5f:1b:40:22:ad IEEE 802.11: associated

Oct 31 13:41:52 imx8mmea-ucom daemon.info hostapd: wlan1: STA
aa:5f:1b:40:22:ad RADIUS: starting accounting session
7A3D9F4430B5BFA0

Oct 31 13:41:52 imx8mmea-ucom daemon.err udhcpd[3458]: sending
OFFER of 192.168.5.100

Oct 31 13:41:53 imx8mmea-ucom daemon.err udhcpd[3458]: sending ACK
to 192.168.5.100

Oct 31 13:46:03 imx8mmea-ucom daemon.info hostapd: wlan1: STA
ac:5f:3e:40:1e:ad IEEE 802.11: disassociated
```

One more way to get information about a connected client:

```
# hostapd_cli all_sta

Selected interface 'wlan1'
aa:5f:1b:40:22:ad
flags=[AUTH] [ASSOC] [AUTHORIZED]
aid=0
capability=0x0
listen_interval=0
supported_rates=
timeout_next=NULLFUNC POLL
rx_packets=1015
tx_packets=193
rx_bytes=62988
tx_bytes=37595
inactive_msec=5000
connected_time=216
```

The last few lines give some statistics for the connection.

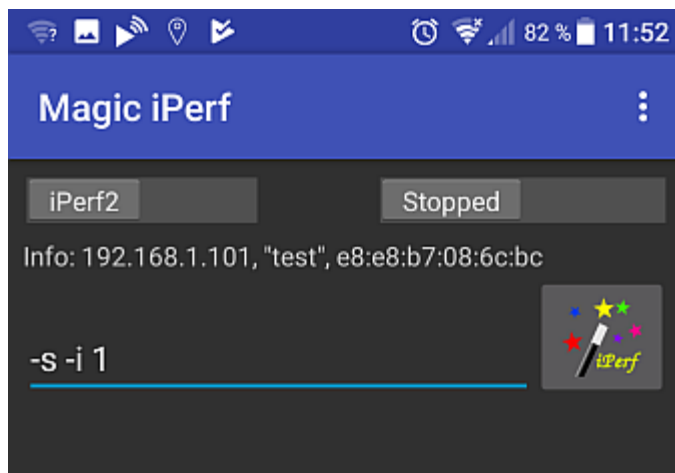
3.8.4 Example - iperf3 on Android

In this example your phone/tablet must have iperf3 software (this example uses the free Magic iPerf app for Android, <https://play.google.com/store/apps/details?id=com.nextdoordeveloper.miperf.miperf>).

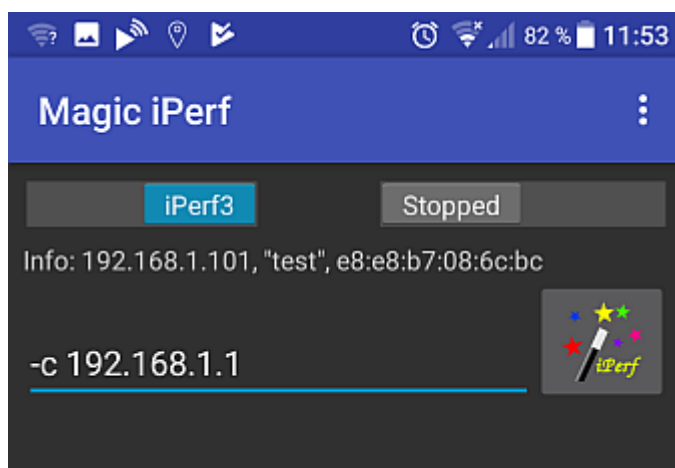
Start the iperf3 server on the target:

```
# iperf3 -s
-----
Server listening on 5201
-----
```

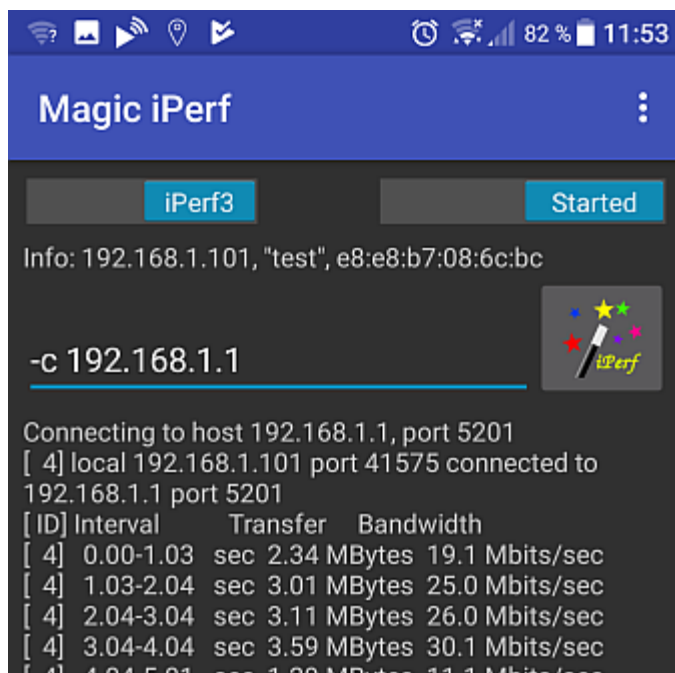
Start the app on the phone. It will look like this the first time:



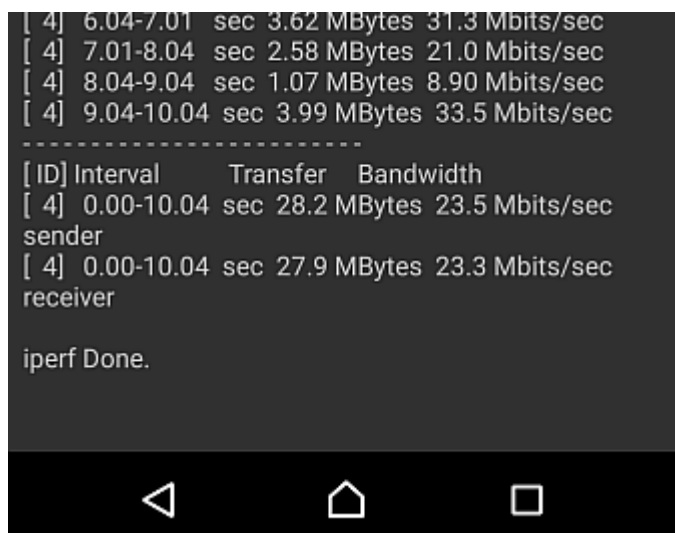
Click the iPerf2 button to change into iperf3 mode then click the text line with "-s -i 1" and change it to "-c 192.168.5.1". It should now look like this:



Click the Stopped button (which will slide to the right and change into a blue Started button) to start the test run:



When the test run ends it will print a summary like this:



The iperf3 server can be stopped with Ctrl+C on the target.

3.8.5 Where to go from here?

- Adding a web server to provide a user interface to the services running on the target. There are several web servers available in yocto, each with their strengths and weaknesses. One interesting feature is a "Captive Portal" which Wikipedia describes as

*a web page accessed with a web browser that is displayed to newly connected users of a Wi-Fi **network** before they are granted broader access to **network** resources.*

There are several guides on how to setup a Captive Portal available on the Internet depending on your choice of web server.

- Look into using iptables (available on the ea-image-base file system) to set up some firewall rules to protect the system from malicious attacks.

3.9 Bluetooth: keyboard

This chapter will show how to connect to a Bluetooth keyboard and how to automatically connect to it after a reboot. Note that the keyboard will appear as an input device and it will not actually work to type in the terminal on the PC. The initialization is the same as explained in section 3.5 but will be repeated for completeness.

Start by initializing Bluetooth (example is for 1ZM):

```
# /opt/ea/Bluetooth_up.sh
Setting TTY to N_HCI line discipline
Device setup complete
< HCI Command: ogf 0x3f, ocf 0x0009, plen 4
  C0 C6 2D 00
> HCI Event: 0x0e plen 4
  01 56 0C 00
Setting TTY to N_HCI line discipline
Device setup complete
Scanning ...

To run a scan again, use hcitool scan
```

Use the interactive `bluetoothctl` command line tool to search for devices:

```
# bluetoothctl
[bluetooth] power on
[bluetooth] agent on
[bluetooth] scan on
Discovery started
[CHG] Controller 44:91:60:9A:7B:3D Discovering: yes
[NEW] Device 5B:09:CC:56:F9:2B 5B-09-CC-56-F9-2B
[NEW] Device 54:1B:BC:EF:AF:5B 54-1B-BC-EF-AF-5B
[CHG] Device 24:4B:03:74:42:FE Name: [TV] UE55JS8505
[NEW] Device 76:09:06:02:00:75 76-09-06-02-00-75
[CHG] Device 76:09:06:02:00:75 LegacyPairing: no
[CHG] Device 76:09:06:02:00:75 Name: Bluetooth 3.0 Keyboard
[CHG] Device 76:09:06:02:00:75 Alias: Bluetooth 3.0 Keyboard
[NEW] Device 5D:B3:02:D1:F9:FC 5D-B3-02-D1-F9-FC
[CHG] Device 76:09:06:02:00:75 LegacyPairing: yes
...
```

The device we are interested in is the 76:09:06:02:00:75 "Bluetooth 3.0 Keyboard". Pair with it:

```
[bluetooth]# pair 76:09:06:02:00:75
Attempting to pair with 76:09:06:02:00:75
[CHG] Device 76:09:06:02:00:75 Connected: yes
[agent] PIN code: 067627
```

In this case the keyboard sends a pass code "067627" that has to be typed on the Bluetooth keyboard followed by the Enter key in order to pair successfully. Check that the pairing was successful:

```
[bluetooth]# paired-devices
Device 76:09:06:02:00:75 Bluetooth 3.0 Keyboard
```

To allow the device to establish the connection by itself it needs to be trusted:

```
[bluetooth]# trust 76:09:06:02:00:75
[CHG] Device 76:09:06:02:00:75 Trusted: yes
Changing 76:09:06:02:00:75 trust succeeded
```

As the last step connect to the keyboard:

```
[bluetooth]# connect 76:09:06:02:00:75
Attempting to connect to 76:09:06:02:00:75
[CHG] Device 76:09:06:02:00:75 Connected: yes
Connection successful

[CHG] Device 76:09:06:02:00:75 ServicesResolved: yes

[Bluetooth 3.0 Keyboard]#
```

Exit the tool:

```
[bluetooth]# quit
```

To test the keyboard:

```
# evtest

No device specified, trying to scan all of /dev/input/event*
Available devices:
/dev/input/event0:      30370000.snvs:snvs-powerkey
/dev/input/event1:      Bluetooth 3.0 Keyboard
Select the device event number [0-1]:
```

Type 1 and then Enter to start the tool. All key presses on the keyboard will be reported. End with Ctrl+C.

The keyboard remains paired until that pairing is broken for example with the remove command in `bluetoothctl`. However, the Bluetooth controller will be turned off after a reboot.

4 COM Carrier Board V2 Advanced Features

There are several advanced and unique features of the COM Carrier Board V2 that has been added to be able to professional evaluation/benchmarking and also debugging. This chapter describes these features.

The picture below illustrates the different connectors and jumpers located on the lower left corner of the COM Carrier Board V2.

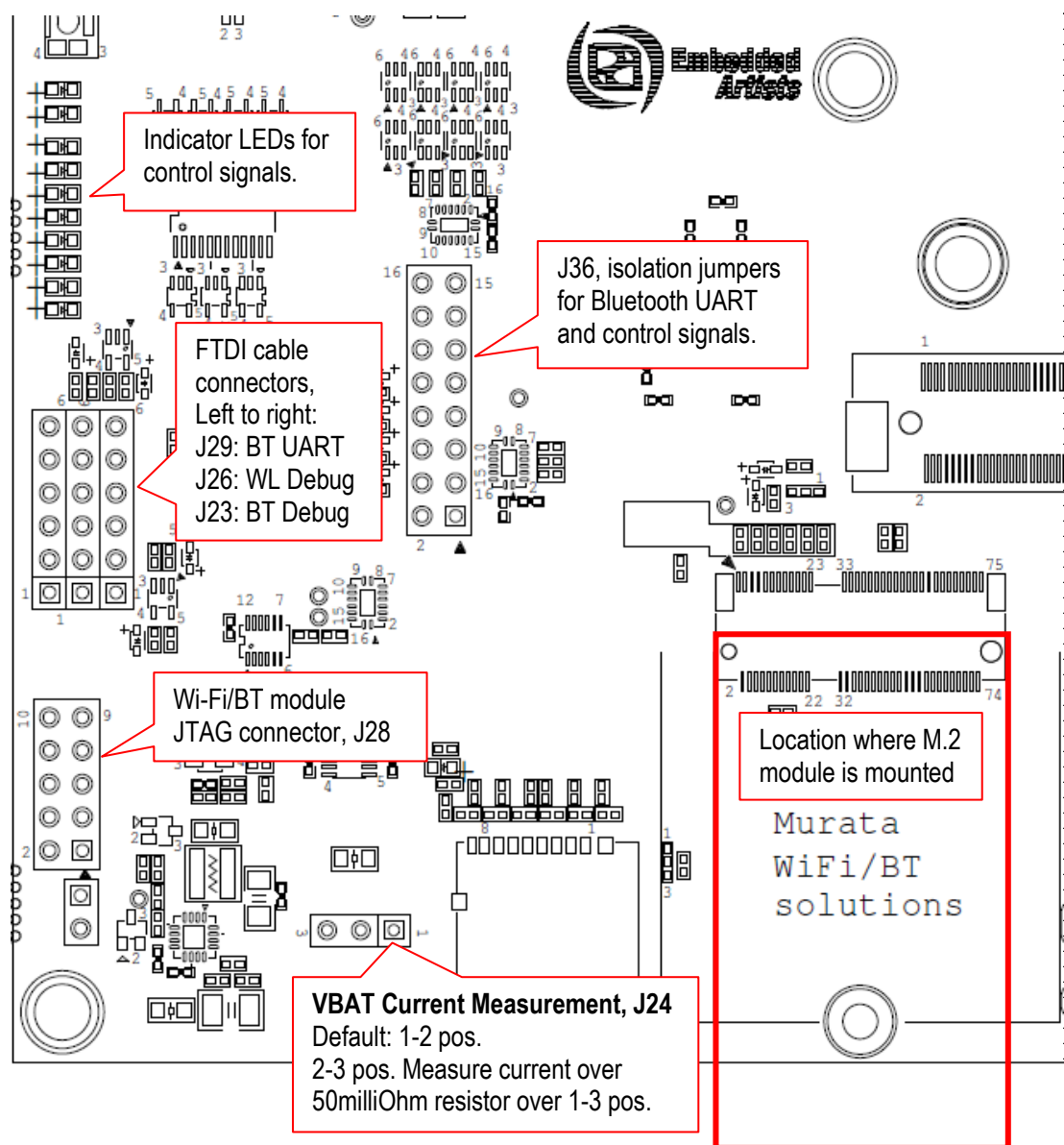


Figure 4 – Lower Left Corner of COM Carrier Board V2

4.1 VBAT Current Measurement

It is possible to measure the VBAT current to the M.2 module exactly.

- Option #1, lift the short jumper (in 1-2 position) and use an external current meter to measure the current exactly, with the resolution possible with the selected meter.

Note: make sure the current meter does not add a voltage drop more than 50-100mV maximum.

- Option #2, move the short jumper to position 2-3. This will add a 50 milliOhm series resistor and it is possible to measure the voltage over this series resistor on pos 1 and 3.

Do not forget to move the short jumper back to position 1-2 after a measurement session.

4.2 VBAT 3.3V or 3.6V

It is possible to set VBAT to either 3.3V or 3.6V during run time. This is controlled via a I2C mapped GPIO. Setting VBAT to 3.6V can improve radio performance on the M.2 module. Note that setting VBAT to 3.6V is outside of the M.2 specification, but if the radio chip/module on the M.2 module is known to handle VBAT set to 3.6V then it can be an option to measure the added performance.

4.3 Support for 3.3V IO logic level (if M.2 module supports it)

The M.2 standard defines the IO voltage logic levels to a mixture of 1.8V and 3.3V. It is possible to set the 1.8V logic signals to 3.3V logic level during run time. This is controlled via a I2C mapped GPIO. Note that before doing this make sure the M.2 module used supports this feature. Not all of them do this.

Also not that this control only affects the controls signals that have 1.8V logic level, not the SDIO bus. The SDIO bus voltage level is controlled via other means.

4.4 Bluetooth UART Interception

It is possible to intercept/overtake the Bluetooth UART communication via connector J29. Insert a UART-to-USB bridge cable from FTDI (TTL-232R-3V3) into J29 and use a PC application to communicate directly with the Bluetooth part of the M.2 module.

Cypress has a tool called CyBluetool that can be used to debug Bluetooth communication problems. The program and instructions on how to use it can be downloaded here:
<https://community.cypress.com/docs/DOC-16475>.

CyBluetool requires direct control of the UART and this is normally controlled by Linux. The UART is different for different COM boards:

COM boards	Bluetooth UART	Serial for u-boot command
iMX6 SoloX COM	/dev/ttymx1	serial1
iMX6 Quad COM	/dev/ttymx4	serial4
iMX6 DualLite COM	/dev/ttymx4	serial4
iMX6 UltraLite COM	/dev/ttymx1	serial1
iMX7 Dual COM	/dev/ttymx1	serial1
iMX7 Dual uCOM	/dev/ttymx1	serial1
iMX7 ULP uCOM	/dev/ttyLP2	serial2
iMX7 ULP uCOM with onboard 1LV	Not externally accessible	
iMX8M COM	/dev/ttymx1	serial1
iMX8M Mini uCOM	/dev/ttymx0	serial0

iMX8M Mini uCOM with onboard 1MW	Not externally accessible
iMX8M Nano uCOM	/dev/ttymx0 serial0
iMX8M Nano uCOM with onboard 1MW	Not externally accessible

To disable that control:

- 1) Start the terminal program on the PC
- 2) Power on the board
- 3) Press space as soon as text appears in the terminal program to stop in the u-boot
- 4) Run this command:

```
=> setenv cmd_custom fdt set serial1 status disabled\;fdt set /modem-reset status disabled
```

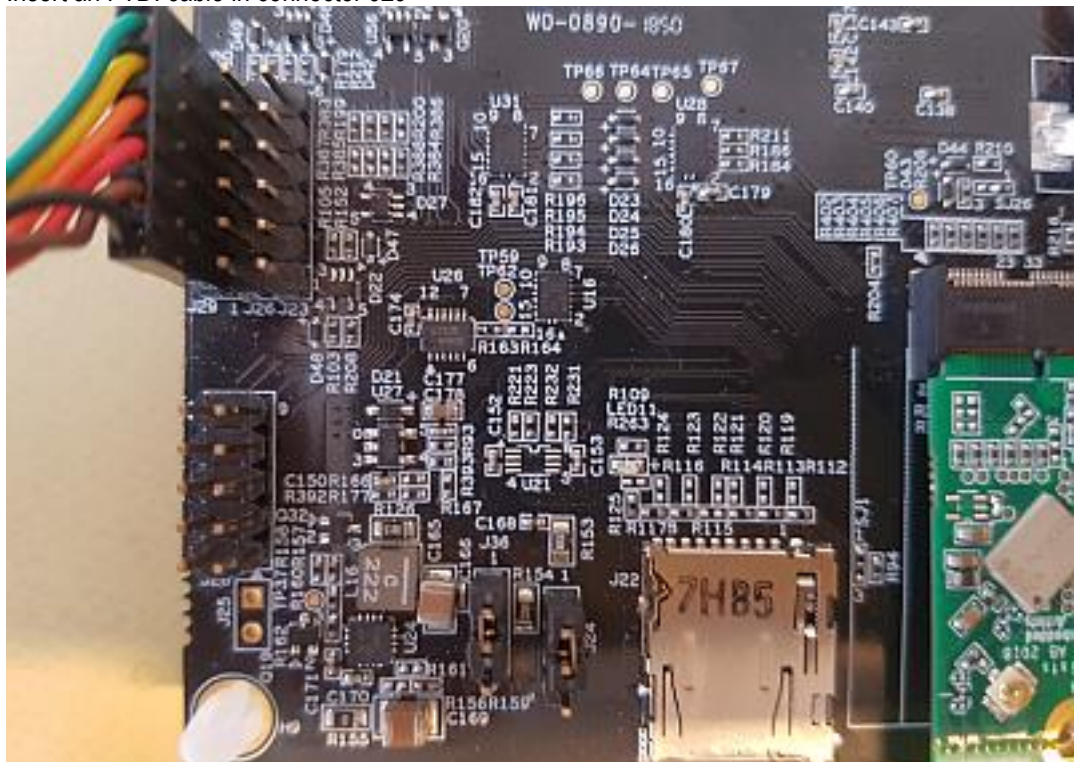
Note that this is one line and the => should not be typed.

The "serial1" part of the command is COM board specific. Replace with the correct one for your board according to the table above.

- 5) Run this command to save the setting:

```
=> saveenv
```

- 6) Power off the board
- 7) Insert an FTDI cable in connector J29



- 8) Power On the board and let it boot into Linux
- 9) Now that Linux is no longer in control of the UART the reset signal must be manually controlled. To enable Bluetooth run the following commands to control BT_REG_ON:

```
# echo 496 > /sys/class/gpio/export
# echo high > /sys/class/gpio/gpio496/direction
```

- 10) This step is only if you are using an i.MX6 Quad or i.MX6 DualLite COM board. These two boards require the CTS signal to be pulled low:

```
# echo 165 > /sys/class/gpio/export
# echo low > /sys/class/gpio/gpio165/direction
```

- 11) Follow the instruction in the CyBluetool document.

The gpio commands are not persistent and must be executed again after a reboot. The U-boot commands are however persistent and will disable the Linux access to the UART after each reboot. To return to normal use of the UART stop in the u-boot and run these two commands:

```
=> setenv cmd_custom
=> saveenv
```

4.5 Dual UART Debug Channels and JTAG

In a cooperation with Murata, Cypress and Embedded Artists a number of pins on the M.2 connector have been defines to carry UART debug channels as well as JTAG signals to the chipset on the M.2 module. Note that not all M.2 modules support all debug channels.

Connector J26 carry the Wi-Fi UART debug channel. Connect a UART-to-USB bridge cable from FTDI (TTL-232R-3V3) into J26 and use a PC terminal application to get access to the debug interface.

Connector J23 carry the Bluetooth UART debug channel. Connect a UART-to-USB bridge cable from FTDI (TTL-232R-3V3) into J26 and use a PC terminal application to get access to the debug interface.

J28 is a JTAG debug interface to the chipset on the M.2 module.

Note that using these debug interfaces typically requires understanding and access to the firmware

4.6 Audio Codec Multiplexing

Audio interface routing between i.MX processor on COM board, M.2 module and audio codec are three corners in a triangle. With multiplexing, any corner can connect to any other corner. There are three options, as listed below. Control is done in run time with two I2C mapped GPIOs.

- Option #1, connect M.2 module audio interface to audio codec on COM Carrier Board V2.
- Option #2, connect M.2 module audio interface to i.MX processor on the COM board.
- Option #3, connect audio interface from i.MX processor (on the COM board) to audio codec on COM Carrier Board V2. This is the default when no M.2 module is used.

5 Software Update

The fastest way to get started is to download a prepared set of files for the COM board you are using from <http://imx.embeddedartists.com>.

Instructions on how to flash that software is available in the [iMX Working with Yocto document](#).

This section is an abbreviated version of that document.

5.1 Linux Host Setup

5.1.1 Introduction

The Yocto build system requires a Linux host machine. You can either run this host as a standalone / native computer or as a virtual machine on, for example, a Microsoft Windows PC. The minimum available hard disk space is 50 GB, but it is recommended that the host machine has at least 120 GB to be able to build the largest image / distribution.

Several Linux distributions are supported by the Yocto project. Please refer to the [Supported Linux Distributions](#) section in the Yocto reference manual for a complete list.

The instructions in this document have been tested on an Ubuntu 14.04. The [iMX Working with Yocto document](#) explains how to setup a virtual machine.

5.1.2 Required Packages

The Yocto Project requires several packages to be installed on the host machine. If you are using any of the distributions in section 5.1.1 follow the instructions below. If you, however, are using another distribution refer to the [Required Packages for the Host Development System](#) section in the Yocto reference manual.

```
$ sudo apt-get install gawk wget git-core diffstat unzip texinfo  
gcc-multilib build-essential chrpath socat  
  
$ sudo apt-get install libsdl1.2-dev xterm sed cvs subversion  
coreutils texi2html docbook-utils python-pysqlite2 help2man make  
gcc g++ desktop-file-utils libgl1-mesa-dev libglu1-mesa-dev  
mercurial autoconf automake groff curl lzop asciidoc  
  
$ sudo apt-get install u-boot-tools
```

5.1.3 Install the repo tool

The `repo` tool has been developed to make it easier to manage multiple Git repositories. Instead of downloading each repository separately the `repo` tool can download all with one instruction. Download and install the tool by following the instructions below.

1. Create a directory for the tool. The example below creates a directory named `bin` in your home folder.

```
$ mkdir ~/bin
```

2. Download the tool

```
$ curl http://commondatastorage.googleapis.com/git-repo-  
downloads/repo > ~/bin/repo
```

3. Make the tool executable

```
$ chmod a+x ~/bin/repo
```

4. Add the directory to the PATH variable. The line below could be added to your .bashrc file so the path is available in each started shell/terminal

```
$ export PATH=~/bin:$PATH
```

5.1.4 Download Yocto recipes

The Yocto project consists of many recipes used when building an image. These recipes come from several repositories and the repo tool is used to download these repositories.

In step 3 below a branch must be selected of the ea-yocto-base repository. The table below lists the branches that support the M.2 modules.

Branch name	Description
ea-5.4.24	u-boot: 2020.04. Linux: 5.4.24.

Table 1 - ea-yocto-base branches

1. Create a directory for the downloaded files (ea-bsp in the example below)

```
$ mkdir ea-bsp
$ cd ea-bsp
```

2. Configure Git if you haven't already done so. Change "Your name" to your actual name and "Your e-mail" to your e-mail address.

```
$ git config --global user.name "Your name"
$ git config --global user.email "Your e-mail"
```

3. Initialize repo. The file containing all needed repositories is downloaded in this step. Change <selected branch> to a branch name according to Table 1.

```
$ repo init -u https://github.com/embeddedartists/ea-yocto-base -b
<selected branch>
```

4. Start to download files

```
$ repo sync
```

All files have now been downloaded into the ea-bsp directory. Most of the files will actually be available in the sub-directory called sources.

5.2 Building Images

Yocto is using the BitBake tool to generate complete Linux images/distributions, that is, all needed to boot and run a Linux system. This is typically boot loader(s), Linux kernel, and root file system with selected utilities and applications.

5.2.1 Available Images

The recipes that have been downloaded contain many different images. The table below describes the ones relevant when working with the M.2 modules.

Image name	Description
meta-toolchain	Builds an installable toolchain (cross-compiler)
ea-image-base	Only available on branch ea-4.14.78 and later. Based on core-image-base and added packages for peripheral testing and verification

5.2.2 Machine Configurations

A machine configuration must be specified before a build can be started.

The table below contains the machine configurations available for Embedded Artists boards. It is also possible to find the configuration files in the directory `~/ea-bsp/sources/meta-ea/conf/machine`.

Machine	Description
imx6sxea-com	Machine configuration for Embedded Artists i.MX 6 SoloX COM Board / Kit
imx6qea-com	Machine configuration for Embedded Artists i.MX 6 Quad COM Board / Kit
imx6dlea-com	Machine configuration for Embedded Artists i.MX 6 DualLite COM Board / Kit
imx6ulea-com	Machine configuration for Embedded Artists i.MX 6 UltraLite COM Board / Kit
imx7dea-ucom	Machine configuration for Embedded Artists i.MX 7 Dual uCOM Board / Kit
imx7dea-com	Machine configuration for Embedded Artists i.MX 7 Dual COM Board / Kit
imx7ulpea-ucom	Machine configuration for Embedded Artists i.MX 7 ULP uCOM Board / Kit
imx8mqea-com	Machine configuration for Embedded Artists i.MX8M Quad COM Board / Kit
imx8mmea-ucom	Machine configuration for Embedded Artists i.MX8M Mini uCOM Board / Kit
imx8mnea-ucom	Machine configuration for Embedded Artists i.MX8M Nano uCOM Board / Kit

5.2.3 Initialize Build

Before starting the build it must be initialized. In this step the build directory and local configuration files are created.

A distribution must be selected when initializing the build, see [iMX Working with Yocto](#) for different alternatives. For headless setups (i.e. without a display) use `fsl-imx-fb`. Note that the i.MX8 boards do not support `fsl-imx-fb` so use `fsl-imx-wayland` instead.

In the example below the machine `imx6sxea-com`, the build directory `build_dir` and the `fsl-imx-fb` distribution (see [iMX Working with Yocto](#) for other distributions) is selected.

```
$ DISTRO=fsl-imx-fb MACHINE=imx6sxea-com source ea-setup-release.sh -b build_dir
```

Restart a Build

If you need to restart a build in a new terminal window or after a restart of the host computer you don't need to run the `ea-setup-release.sh` script again. Instead you run the `setup-environment` script. If you don't run the `setup-environment` script you won't have access to needed tools and utilities, such as `bitbake`.

```
$ source setup-environment build_dir
```

5.2.4 Starting the Build

Everything has now been setup to start the actual build. The example below shows how the `ea-image-base` image is being built. Please note that depending on the capabilities of your host computer building an image can take many hours.

```
$ bitbake ea-image-base
```

When the build has finished the images will be available in the directory specified below. Please note that this directory will be different if you are using another build directory or machine configuration.

`~/ea-bsp/build_dir/tmp/deploy/images/imx6sxea-com.`

Go to chapter 5.3 for instructions of how to deploy images to the target hardware.

5.3 Deploying Images

UUU (Universal Update Utility) is version 3 of MFGTool (NXP's Manufacturing Tool) but it has been rewritten, is publicly available on GitHub (<https://github.com/NXPmicro/mfgtools>) and it can be run on both Windows and Linux while the older versions of MFGTool were limited to Windows only.

UUU can be used to write images to the board. This tool is sending files and instructions over USB and the board must be set in OTG boot mode for it to work.

Prerequisites:

- Ubuntu 16.04 or above, 64-bit
- Windows 10, 64-bit
- Windows 7, 64-bit - note that there might be problems with drivers and that it might not even work with the driver fixes applied even if the documentation says it does. The Windows 7 specific instructions can be found here: <https://github.com/NXPmicro/mfgtools/wiki/WIN7-User-Guide>

Useful links:

- UUU on GitHub: <https://github.com/NXPmicro/mfgtools>

- UUU release page: <https://github.com/NXPmicro/mfgtools/releases>

5.3.1.1 Download the Tool

Download the zip file for the board you are using from <http://imx.embeddedartists.com/>

Unpack this zip file somewhere on your computer. Below is a description of some of the content in the zip file.

- `uuu (root)`: Contains a README file.
- `uuu/uuu.exe`: The Windows version of the tool
- `uuu/*.uuu`: The different download configurations.
- `uuu/files/`: Contains pre-compiled versions of images. The tool will look in this directory when selecting images to download to the board.

The UUU zip file includes the Windows version of tool itself (i.e. `uuu.exe`). The README file contains a link to where the latest binaries can be downloaded (<https://github.com/NXPmicro/mfgtools/releases>). Download either `uuu.exe` (for Windows) or `uuu` for Linux and save the file in the same folder as the README file.

5.3.1.2 Prepare hardware

Begin by reading the *Getting Started* document for the board you are using. It shows how to setup the board and also gives an overview of the hardware.

5.3.1.3 OTG boot mode – J2 jumper

To download images using `uu` the board must be put into OTG boot mode.

This is accomplished by closing the J2 jumper on the Carrier board; see Figure 5 to locate the jumper. Please note that in the figure the jumper is in open state which means that the COM board will boot from eMMC.



Figure 5 - J2 jumper (opened state) on a COM Carrier Board V2

Note: When you want to boot the software from eMMC you have to remove jumper J2.

5.3.1.4 Configurations

Several configurations (*.uuu files) for the tool have been prepared in order to help you download specific images.

- `bootloader.uuu` – will install only the bootloaders. This should only be used if you want to restore the bootloaders or download your own bootloaders to the board.
- `bootloader_combined.uuu` – will install only the bootloaders. This is a faster alternative to `bootloader.uuu` but it requires a binary where SPL and the u-boot have been combined (see below). This should only be used if you want to restore the bootloaders or download your own bootloaders to the board.
- `kernel.uuu` – will install kernel and dtb files. This should only be used if you want to update the kernel or dtb files.
- `full_tar.uuu` – will install bootloaders, Linux kernel and root file system. The root file system will be installed from a tar.bz2 file.
- `raw_sdcard_example.uuu` – will overwrite the eMMC with the content of an sdcard file. The sdcard file is copied directly to the eMMC overwriting everything including bootloaders, Linux kernel and file system.

If you want to create the combined binary to use with `bootloader_combined.uuu` run the following commands in Linux:

```
$ dd if=SPL of=spl_and_uboot.bin bs=1024
$ dd if=u-boot.img of=spl_and_uboot.bin bs=1024 seek=68
```

5.3.1.5 Download Your Own Images

The UUU zip file that you download from <http://imx.embeddedartists.com/> contain the latest build from Embedded Artists.

The simplest way to download your own images is to replace the existing file(s) with your own file(s). If you keep the file names intact the *.uuu configurations will download your version of the file.

5.3.1.6 Run the Tool in Ubuntu

On Linux open a terminal, navigate to the folder where the UUU zip file was unpacked, make sure that the tool is executable and then execute the tool:

```
$ cd ~/uuu_imx8mq_com_5.4.24
$ chmod +x ./uuu
$ sudo ./uuu full_tar.uuu
```

The terminal will show a progress bar like this while it is running:

```
andli@lenovo:~/uuu_imx8mq_com_4.14.78$ sudo ./uuu full_tar.uuu
uuu (Universal Update Utility) for nxp imx chips -- libuuu_1.2.91-0-g3799f4d

Success 0      Failure 0

2:24 20/23 [=====47%] FBK: ucp files/ea-image-base-imx8mqea-com.tar.bz2 t:-
```

After a successful run it will look like this:

```
andli@lenovo:~/uuu_imx8mq_com_4.14.78$ sudo ./uuu full_tar.uuu
uuu (Universal Update Utility) for nxp imx chips -- libuuu_1.2.91-0-g3799f4d

Success 1      Failure 0

2:24 23/23 [Done] FBK: DONE
```

If a problem occurs then the program will terminate and print an error message like this

```
andli@lenovo:~/uuu_imx8mq_com_4.14.78$ sudo ./uuu full_tar.uuu
uuu (Universal Update Utility) for nxp imx chips -- libuuu_1.2.91-0-g3799f4d

Success 0    Failure 1

2:24  1/ 1  [HID(W):LIBUSB_ERROR_IO ] SDP: boot -f files/u-boot-imx8mqea-com.bin
andli@lenovo:~/uuu_imx8mq_com_4.14.78$
```

5.3.1.7 Run the Tool in Windows

On Windows open a Command Prompt, navigate to the folder where the UUU zip file was unpacked and then run the tool:

```
C:\> cd c:\temp\uuu_imx8mq_com_5.4.24
C:\temp\uuu_imx8mq_com_5.4.24> uuu.exe full_tar.uuu
```

The terminal will show a progress bar like this while it is running:

```
c:\temp\uuu_imx8mq_com_4.14.78>uuu full_tar.uuu
uuu (Universal Update Utility) for nxp imx chips -- libuuu_1.2.91-0-g3799f4d

Success 0    Failure 0

1:23  20/23  [=====> 34% ] FBK: ucp files/ea-image-base-imx8mqea-com.tar.bz2 t:-
```

After a successful run it will look like this:

```
c:\temp\uuu_imx8mq_com_4.14.78>uuu full_tar.uuu
uuu (Universal Update Utility) for nxp imx chips -- libuuu_1.2.91-0-g3799f4d

Success 1    Failure 0

1:23  23/23  [Done ] FBK: DONE

c:\temp\uuu_imx8mq_com_4.14.78>
```

If a problem occurs then the program will terminate and print an error message like this

```
c:\temp\uuu_imx8mq_com_4.14.78>uuu full_tar.uuu
uuu (Universal Update Utility) for nxp imx chips -- libuuu_1.2.91-0-g3799f4d

Success 0    Failure 1

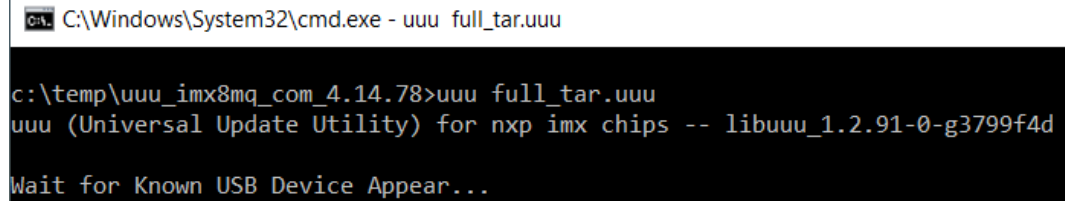
1:23  20/23  [Bulk(R):LIBUSB_ERROR_PIPE ] FBK: ucp files/ea-image-base-imx8mqea-com.tar.bz2 t:-

c:\temp\uuu_imx8mq_com_4.14.78>
```

5.3.1.8 Troubleshoot

Some common problems and solutions:

- **The first time you run UUU on your computer it fails.**
This is likely because of USB driver installation. Let the driver install, reset the hardware and then run the uuu command again. In Windows it is three different drivers that are needed so this procedure might have to be repeated three times - each time the procedure gets a little bit further.
- UUU appears to hang with a "Wait for Known USB Device Appear..." message like this:



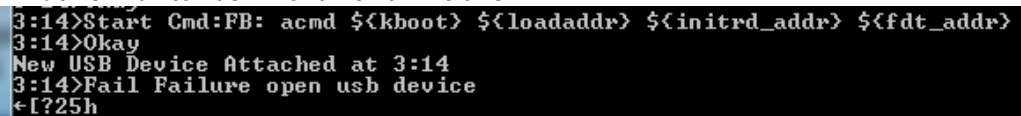
```
C:\Windows\System32\cmd.exe - uuu full_tar.uuu

c:\temp\uuu_imx8mq_com_4.14.78>uuu full_tar.uuu
uuu (Universal Update Utility) for nxp imx chips -- libuuu_1.2.91-0-g3799f4d

Wait for Known USB Device Appear...
```

This means that the hardware is either not connected to the computer with the USB cable or it is not in the OTG boot mode. Check **Fel! Hittar inte referensskälla.** to **Fel! Hittar inte referensskälla.** again and then run the uuu command again.

- **Windows 7 fail to flash with an error like this:**



```
3:14>Start Cmd:FB: acmd ${kboot} ${loadaddr} ${initrd_addr} ${fdt_addr}
3:14>Okay
New USB Device Attached at 3:14
3:14>Fail Failure open usb device
←[?25h
```

It could be due to a driver problem. Follow instructions here:

<https://github.com/NXPmicro/mfgtools/wiki/WIN7-User-Guide>

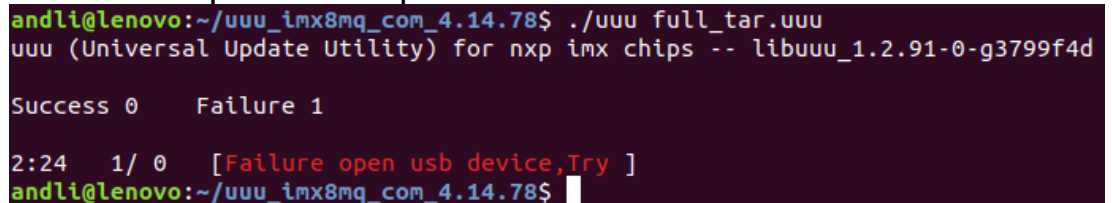
- **Windows 7 terminal does not appear as in the screenshots**

This is because Windows 7 does not support what the UUU tool calls "VT mode" so it defaults to verbose mode which has a lot more printouts and no progress bar.

- **Running raw_sdcard_example.uuu complains about a missing .sdcard file**

That file is not supplied in the downloaded zip file but you will find it in the "deploy" folder after you complete your own yocto build.

- **UUU in Ubuntu reports failure to open usb device:**



```
andli@lenovo:~/uuu_imx8mq_com_4.14.78$ ./uuu full_tar.uuu
uuu (Universal Update Utility) for nxp imx chips -- libuuu_1.2.91-0-g3799f4d

Success 0      Failure 1

2:24 1/ 0 [Failure open usb device, Try ]
andli@lenovo:~/uuu_imx8mq_com_4.14.78$
```

This happens if the UUU program is not executed with the correct rights. Either use "sudo uuu" or setup udev rules so that sudo rights are not needed. The instructions for how to

create the udev rules are built into the tool so run "uuu -udev" and then follow the steps:

```
andli@lenovo:~/uuu_imx8mq_com_4.14.78$ ./uuu -udev
SUBSYSTEM=="usb", ATTRS{idVendor}=="1fc9", ATTRS{idProduct}=="012f", MODE="0666"
SUBSYSTEM=="usb", ATTRS{idVendor}=="1fc9", ATTRS{idProduct}=="0129", MODE="0666"
SUBSYSTEM=="usb", ATTRS{idVendor}=="15a2", ATTRS{idProduct}=="0076", MODE="0666"
SUBSYSTEM=="usb", ATTRS{idVendor}=="15a2", ATTRS{idProduct}=="0054", MODE="0666"
SUBSYSTEM=="usb", ATTRS{idVendor}=="15a2", ATTRS{idProduct}=="0061", MODE="0666"
SUBSYSTEM=="usb", ATTRS{idVendor}=="15a2", ATTRS{idProduct}=="0063", MODE="0666"
SUBSYSTEM=="usb", ATTRS{idVendor}=="15a2", ATTRS{idProduct}=="0071", MODE="0666"
SUBSYSTEM=="usb", ATTRS{idVendor}=="15a2", ATTRS{idProduct}=="007d", MODE="0666"
SUBSYSTEM=="usb", ATTRS{idVendor}=="15a2", ATTRS{idProduct}=="0080", MODE="0666"
SUBSYSTEM=="usb", ATTRS{idVendor}=="1fc9", ATTRS{idProduct}=="0128", MODE="0666"
SUBSYSTEM=="usb", ATTRS{idVendor}=="1fc9", ATTRS{idProduct}=="0126", MODE="0666"
SUBSYSTEM=="usb", ATTRS{idVendor}=="1fc9", ATTRS{idProduct}=="0135", MODE="0666"
SUBSYSTEM=="usb", ATTRS{idVendor}=="1fc9", ATTRS{idProduct}=="0134", MODE="0666"
SUBSYSTEM=="usb", ATTRS{idVendor}=="1fc9", ATTRS{idProduct}=="012b", MODE="0666"
SUBSYSTEM=="usb", ATTRS{idVendor}=="0525", ATTRS{idProduct}=="b4a4", MODE="0666"
SUBSYSTEM=="usb", ATTRS{idVendor}=="0525", ATTRS{idProduct}=="b4a4", MODE="0666"
SUBSYSTEM=="usb", ATTRS{idVendor}=="0525", ATTRS{idProduct}=="b4a4", MODE="0666"
SUBSYSTEM=="usb", ATTRS{idVendor}=="066f", ATTRS{idProduct}=="9afe", MODE="0666"
SUBSYSTEM=="usb", ATTRS{idVendor}=="066f", ATTRS{idProduct}=="9bff", MODE="0666"
SUBSYSTEM=="usb", ATTRS{idVendor}=="0525", ATTRS{idProduct}=="a4a5", MODE="0666"
SUBSYSTEM=="usb", ATTRS{idVendor}=="18d1", ATTRS{idProduct}=="0d02", MODE="0666"

1: put above udev run into /etc/udev/rules.d/99-uuu.rules
   sudo sh -c "uuu -udev >> /etc/udev/rules.d/99-uuu.rules"
2: update udev rule
   sudo udevadm control --reload-rules
andli@lenovo:~/uuu_imx8mq_com_4.14.78$
```

5.4 Building without Yocto

5.4.1 Stand-alone Toolchain

To be able to build your own application or, for example, u-boot and the Linux kernel outside of Yocto you need a toolchain. The toolchain consists of cross-compiler, linker, and necessary libraries. As mentioned in section 5.2.1 there is an image named **meta-toolchain** that will create the necessary toolchain.

1. Build the image

```
$ bitbake meta-toolchain
```

2. The build will result in a file located at <build dir>/tmp/deploy/sdk. The exact name of the file depends on the host computer and the target board, but in our example it is called:

```
fsl-imx-fb-glibc-x86_64-meta-toolchain-cortexa9hf-neon-
toolchain-4.14-sumo.sh
```

3. Install the toolchain

```
$ cd <build dir>/tmp/deploy/sdk
$ sudo ./fsl-imx-fb-glibc-x86_64-meta-toolchain-cortexa9hf-neon-
toolchain-4.14-sumo.sh
```

4. If you select the default settings the toolchain will in this example be installed in /opt/fsl-imx-fb/ 4.14-sumo

5. Before building an application run the command below to setup environment variables

```
$ source /opt/fsl-imx-fb/4.14-sumo/environment-setup-cortexa9hf-  
neon-poky-linux-gnueabi
```

6. You can verify that the environment variables has been correctly setup by running the command below that will show the version of the GCC compiler used.

```
$ $CC --version  
arm-poky-linux-gnueabi-gcc (GCC) 7.3.0  
...
```

NOTE 1: Setting up environment variables in step 5 may overwrite other variables you already have in your environment. It is, for example, not recommended to do this in the same terminal where you run bitbake to build Yocto images.

It is recommended to build this toolchain on your host computer and where you will do the development and that you build for the same board as for example i.MX6UL/i.MX7Dual have a different (ARM7) cross compiler than other i.MX6 processors (ARM9).

5.4.2 Build Linux kernel from source code

You can build the Linux kernel outside of Yocto by following the instructions in this section. Please note that it is recommended that the kernel is built by Yocto when you are generating your final distribution images since there can be dependencies between the root file system and the kernel.

The instructions in this section assume that you have built and installed the toolchain as described in section 5.4.1 above.

Setup the **environment variables** for the toolchain. We are using the same installation path as described in section 5.4.1 above. If you have installed the toolchain in a different path use that path in the instructions below.

```
$ source /opt/fsl-imx-fb/4.14-sumo/environment-setup-cortexa9hf-  
neon-poky-linux-gnueabi
```

Get the **source code** from the Embedded Artists GitHub repository. In this example we are checking out branch **ea_5.4.24**.

```
$ git clone https://github.com/embeddedartists/linux-imx.git  
$ cd linux-imx  
$ git checkout ea_5.4.24
```

Use Embedded Artists **kernel configurations**.

```
$ make ea_imx_defconfig
```

(Optional) If you want to change kernel configurations you can at this point run the **menuconfig** tool.

```
$ make menuconfig
```

Build the kernel.

```
$ make
```

When the build process has finished the kernel will be available here:

arch/arm/boot/zImage - for iMX6 and iMX7 boards

arch/arm64/boot/Image - for iMX8 boards

Device tree files are available in the following directory:

arch/arm/boot/dts/ - for iMX6 and iMX7 boards

arch/arm64/boot/dts/freescale/ - for iMX8 boards

A compiled device tree file has the file extension `dtb`.

Updating the system

To update the system use manufacturing tool as described in 5.3 . For alternative ways of updating see the [iMX Working with Yocto document](#).

5.4.3 Build u-boot from source code

You can build u-boot outside of Yocto by following the instructions in this section.

The instructions in this section assume that you have built and installed the toolchain as described in section 5.4.1 above.

Setup the **environment variables** for the toolchain. We are using the same installation path as described in section 5.4.1 above. If you have installed the toolchain in a different path use that path in the instructions below.

```
$ source /opt/fsl-imx-fb/4.14-sumo/environment-setup-cortexa9hf-  
neon-poky-linux-gnueabi
```

Get the **source code** from the Embedded Artists GitHub repository. In this example we are checking out branch **ea_v2020.04**.

```
$ git clone https://github.com/embeddedartists/uboot-imx.git  
$ cd uboot-imx  
$ git checkout ea_v2020.04
```

Use the Embedded Artists **configuration** for the COM board you are using. In the example below the configuration for the iMX6 SoloX COM board is used.

```
$ make mx6sxea-com_config
```

Build the bootloader.

```
$ make
```

When the build process has finished the u-boot image (and in some cases SPL) will be available directly in the `uboot-imx` directory.

Updating the system

Use the manufacturing tool as described in section 5.3 to update the system with the new u-boot image and possibly SPL if your board requires it.

5.5 NVRAM

<TBD>Describe NVRAM and how it is updated/modified

5.6 Firmware

<TBD>Describe how to update the firmware for the M.2 modules, if needed

6 Appendix - Updating Files on Target

There are four parts that make up a running system: u-boot, Linux kernel, device tree files and a root file system. The u-boot and root file system have to be flashed using MfgTool/UUU as described in section 5.3. The Linux kernel and the device tree files are however much easier to update and this appendix will show a couple of different ways to do this.

- U-boot USB Mass Storage Gadget - Quick, no extra software or network connection needed
- Secure Copy From Target - Requires network but no extra software
- Secure Copy To Target - Requires network and PC software. Target must be modified to allow incoming connection. The advantage is an application with a GUI.
- USB memory stick - Only requires a Memory Stick but using it involves a lot of plugging/copying/unplugging

Note that the sections below focus on updating the Linux kernel and device tree files but the same commands can be used to update any file on the target.

6.1 U-boot USB Mass Storage Gadget

The u-boot has an `ums` command that can export an mmc device as a USB Mass Storage making it accessible from a PC. The mmc device numbering is different for each board and changes with versions of the u-boot so the table below shows the command to use for version 2020.04.

COM board	Command
iMX6 SoloX COM	<code>ums 0 mmc 2</code>
iMX6 Quad COM	<code>ums 0 mmc 3</code>
iMX6 DualLite COM	<code>ums 0 mmc 3</code>
iMX6 UltraLite COM	<code>ums 0 mmc 1</code>
iMX7 Dual COM	<code>ums 0 mmc 2</code>
iMX7 Dual uCOM	<code>ums 0 mmc 2</code>
iMX7ULP uCOM	<code>ums 0 mmc 0</code>
iMX8M Quad COM	<code>ums 0 mmc 0</code>
iMX8M Mini uCOM	<code>ums 0 mmc 2</code>
iMX8M Nano uCOM	<code>ums 0 mmc 2</code>

To use the `ums` command, power on, stop in the u-boot, connect the micro USB cable in J11 and to the PC and then execute the command for your board. For iMX6 SoloX it will look like this:

```
=> ums 0 mmc 0
UMS: LUN 0, dev 0, hwpart 0, sector 0x0, count 0x72c000
/
```

There will be a spinning character indicating that the command is running. After a few seconds the PC should detect the USB drive(s) and make them available.

The number of drives that appear and if they can be accessed, or not, depends on the operating system. Windows and Linux are both able to access the first drive as it is FAT formatted. It

corresponds to the mmc partition with the Linux kernel and device tree files on it. As Linux has ext3/ext4 file system support it can also access the second drive which corresponds to the root file system of the target.

Add/remove files on the PC and then use the "safe unmounting" option in Windows/Linux to make sure all changes have been written. Finally stop the ums command in the u-boot by typing Ctrl+C in the terminal.

```
=> ums 0 mmc 0
UMS: LUN 0, dev 0, hwpart 0, sector 0x0, count 0x72c000
CTRL+C - Operation aborted
=>
```

Some things to note

- The u-boot is single threaded so the transfer speed of files to/from the USB drive is quite low. That does not matter when updating the Linux kernel and/or device tree files but for large files it is probably quicker to boot into Linux first (and do the file transfer from there).
- When accessing the root file system from a PC running Linux you may have to run the commands as root (for example using `sudo`) as all files are owned by root.
- The ums support was verified on the ea_v2018.03 branch of the u-boot and connected to a PC running Windows 7, Windows 10 and Ubuntu 18.04.

6.2 Secure Copy from Target

The default file system on all iMX Developer's Kits come with the `scp` tool preinstalled. If the target is connected to the same network as you build computer and the build computer has an ssh server running then `scp` can be used to transfer files from the build server.

The first step is to mount the mmc partition that holds the Linux kernel and device tree files. The mmc device numbering is different for each board so the table below shows the command to use.

COM board	eMMC device in Linux	Command
iMX6 SoloX COM	/dev/mmcblk2	mount /dev/mmcblk2p1 /mnt/mmc
iMX6 Quad COM	/dev/mmcblk3	mount /dev/mmcblk3p1 /mnt/mmc
iMX6 DualLite COM	/dev/mmcblk3	mount /dev/mmcblk3p1 /mnt/mmc
iMX6 UltraLite COM	/dev/mmcblk1	mount /dev/mmcblk1p1 /mnt/mmc
iMX7 Dual COM	/dev/mmcblk2	mount /dev/mmcblk2p1 /mnt/mmc
iMX7 Dual uCOM	/dev/mmcblk2	mount /dev/mmcblk2p1 /mnt/mmc
iMX7ULP uCOM	/dev/mmcblk0	mount /dev/mmcblk0p1 /mnt/mmc
iMX8M Quad COM	/dev/mmcblk0	mount /dev/mmcblk0p1 /mnt/mmc
iMX8M Mini uCOM	/dev/mmcblk2	mount /dev/mmcblk2p1 /mnt/mmc
iMX8M Nano uCOM	/dev/mmcblk2	mount /dev/mmcblk2p1 /mnt/mmc

Using iMX8M Quad COM as an example:

```
# mkdir /mnt/mmc
# mount /dev/mmcblk0p1 /mnt/mmc
# ls /mnt/mmc/
Image
boot.scr
fsl-imx8mq-ea-com-kit_v2-m4.dtb
fsl-imx8mq-ea-com-kit_v2-ov5640.dtb
fsl-imx8mq-ea-com-kit_v2-pcie.dtb
fsl-imx8mq-ea-com-kit_v2.dtb
```

The scp command looks like this:

```
scp <username>@<server>:<path> <destination>
```

So to copy the kernel from server with IP address 192.168.0.10 as user bob it could look like this:

```
# scp bob@192.168.0.10:/home/bob/linux-imx/arch/arm64/boot/Image /mnt/mmc/
```

After copying the files from the build server make sure to unmount the mmc partition before rebooting:

```
# umount /mnt/mmc
```

6.3 Secure Copy To Target - WinSCP

WinSCP is a Windows program that is very useful to transfer files from the PC to the target hardware. It can also be used to transfer the files from a yocto build to the PC where the files can be flashed using UUU as described in section 5.3 .

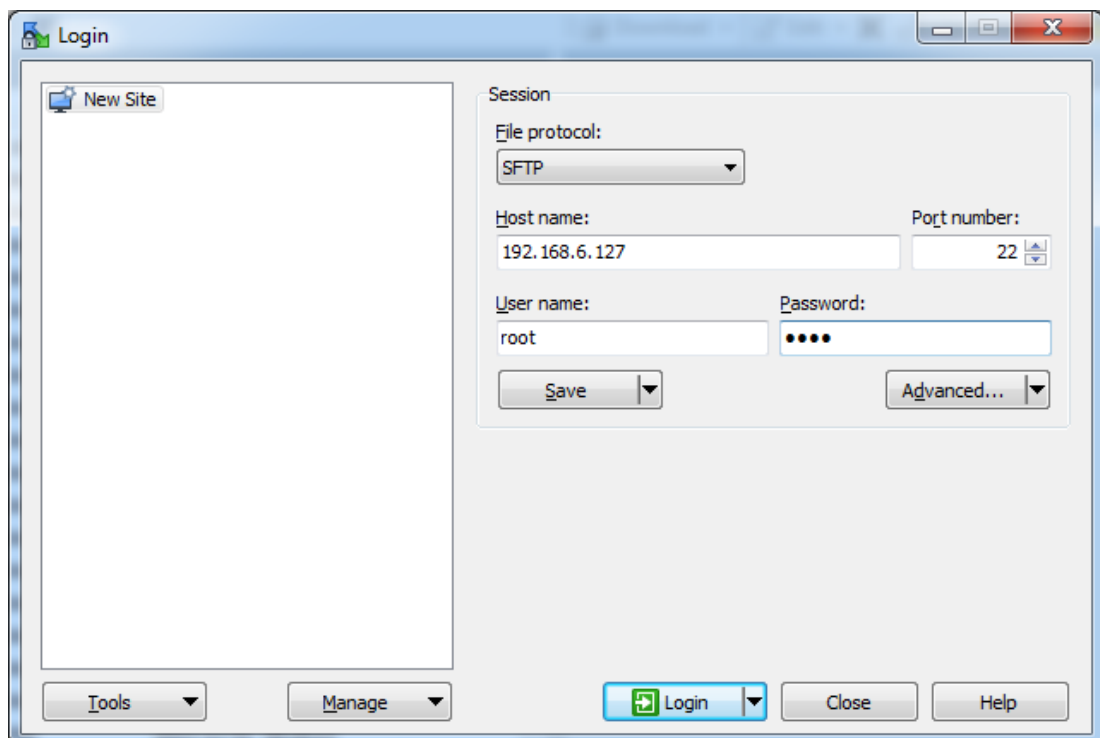
6.3.1 Download and Install

Download the program from <https://winscp.net/eng/download.php> and install it.

6.3.2 Connect to Target

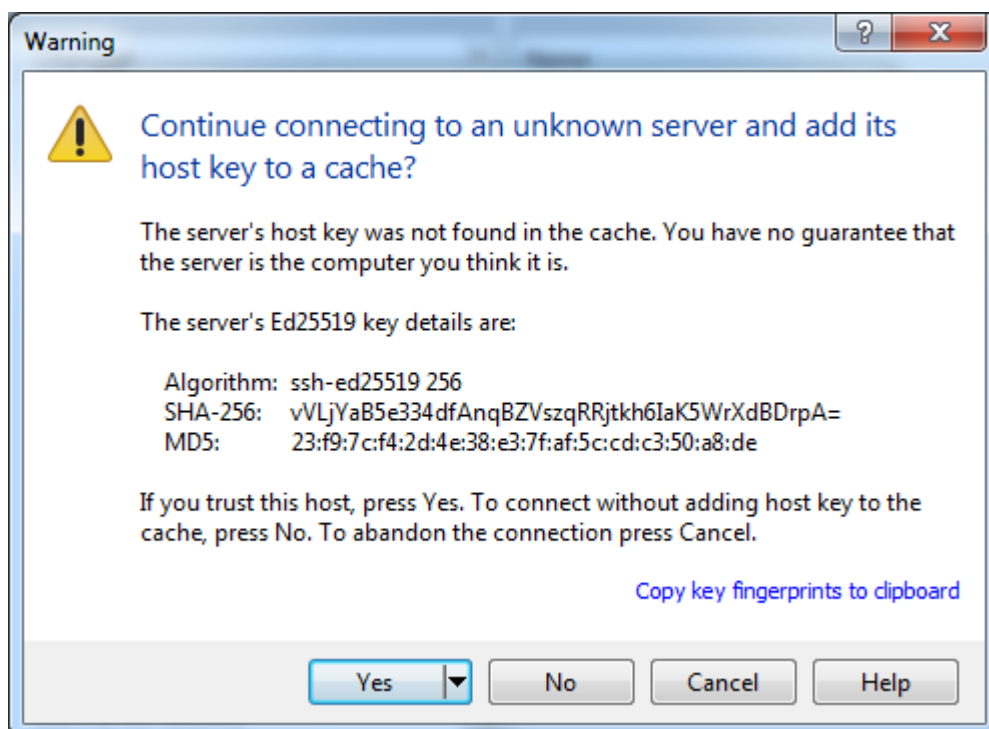
When WinSCP is started it asks for connection information. Enter the following information

Field	Value
User name	Root
Password	Pass
Host name	The IP number of the target (found by running "ifconfig" on the target)
File protocol	SCP or SFTP



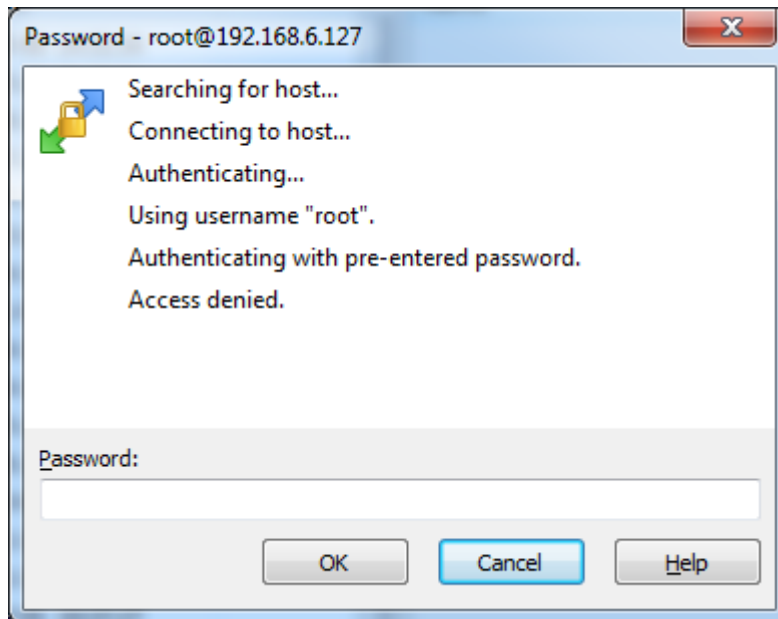
Press the Login button to connect to the target.

The first time connecting to the target (or after a re-flash of the target) this dialog will appear:



Press the Yes button to accept it.

If you get a dialog like the one below asking for the password and you are sure that you have entered the correct one (pass) then it is probably because the target is configured not to allow the root user to connect.



Allowing root access is a huge security threat and it should never be allowed in a production system. To allow it during development in a controlled environment is ok and to do it run the following command on the target and then reboot to apply the change:

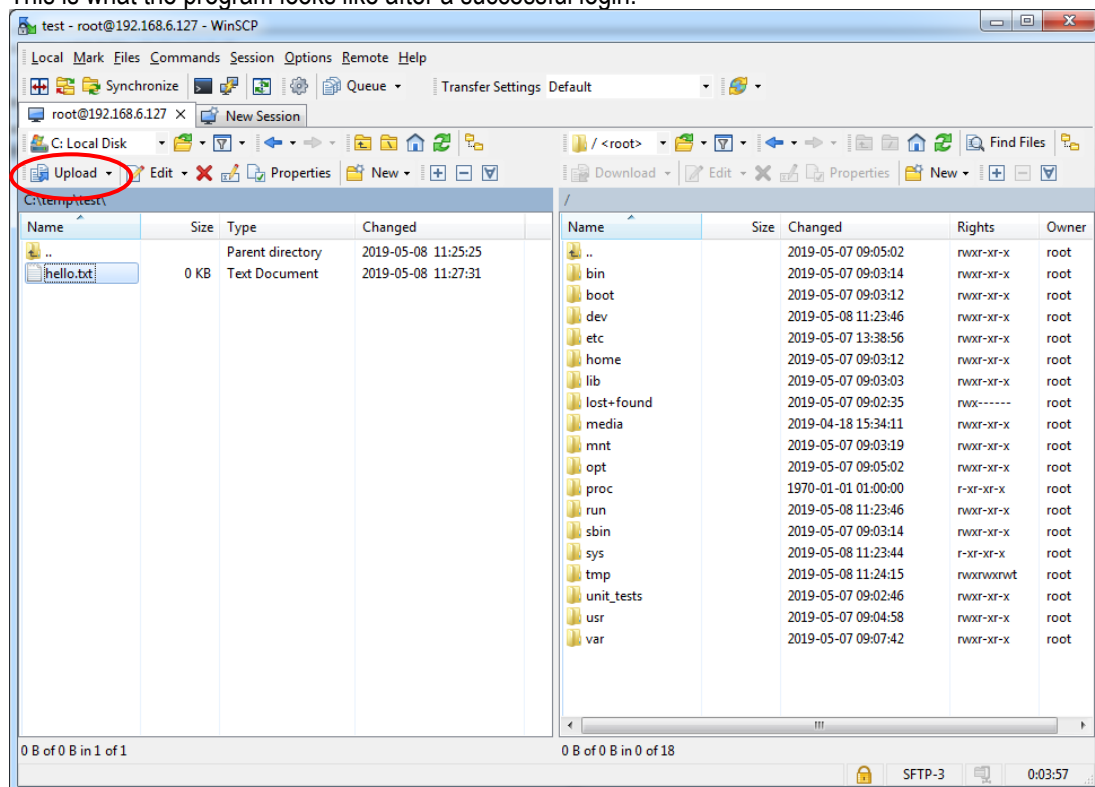
```
# sed -i 's/#PermitRoot/PermitRoot/' /etc/ssh/sshd_config
```

If you prefer to edit the file manually then open `/etc/ssh/sshd_config` and search for `#PermitRootLogin yes` and then remove the preceding `#` sign.

Note that the command is persistent but needs to be executed again if you re-flash the file system.

6.3.3 Copy Files

This is what the program looks like after a successful login:



Select a file in the left side (the PC), select a destination folder on the target (right side) and then click the Upload button to transfer the file.

Note that to copy files directly to the mmc partition, use the `mount` and `unmount` commands as described in section 6.2 above. You will then find the mounted partition under the `mnt/mmc/` folder in WinSCP.

6.4 USB Memory Stick

If the build machine is not accessible over network then copying the files on the build server onto a USB Memory Stick could be an option.

When the USB Memory Stick is inserted on the target some status messages will be printed in the console.

```
usb 1-1.3: new high-speed USB device number 5 using ci_hdrc
usb-storage 1-1.3:1.0: USB Mass Storage device detected
scsi1 : usb-storage 1-1.3:1.0
scsi 1:0:0:0: Direct-Access      Kingston DataTraveler G2  1.00 PQ:
0 ANSI: 2
sd 1:0:0:0: [sda] 31252024 512-byte logical blocks: (16.0 GB/14.9
GiB)
sd 1:0:0:0: [sda] Write Protect is off
sd 1:0:0:0: [sda] Incomplete mode parameter data
sd 1:0:0:0: [sda] Assuming drive cache: write through
sd 1:0:0:0: [sda] Incomplete mode parameter data
sd 1:0:0:0: [sda] Assuming drive cache: write through
sda: sda1
```

```
sd 1:0:0:0: [sda] Incomplete mode parameter data
sd 1:0:0:0: [sda] Assuming drive cache: write through
sd 1:0:0:0: [sda] Attached SCSI removable disk
```

The interesting part above is the “sda: sda1” which indicates which device (`sda1`) that the USB memory stick is assigned to.

To be able to access the memory stick it must first be mounted:

```
# mkdir /mnt/usb
# mount /dev/sda1 /mnt/usb
```

The memory stick is now available in the `/mnt/usb` directory on the file system:

```
# ls /mnt/usb/
core-image-base-imx6sxea-com.rootfs.ext3
core-image-base-imx6sxea-com.rootfs.tar.bz2
imx6sxea-com-kit.dtb
u-boot-imx6sxea-com.img
zImage-imx6sxea-com
```

If the mmc partition has been mounted as described in section 6.2 above then files can be copied from the directory to the mmc partition like this:

```
# cp /mnt/usb/imx6sxea-com-kit.dtb /mnt/mmc/
```

Make sure to unmount the mmc after completing the copying.

Before physically removing the memory stick from the COM Carrier Board, it should be unmounted to make sure that all pending write operations are committed to prevent data loss:

```
# umount /mnt/usb
```

Mouser Electronics

Authorized Distributor

Click to View Pricing, Inventory, Delivery & Lifecycle Information:

[Embedded Artists:](#)

[EAR00373](#)