Using interfaces and peripherals on iMX Developer's Kits

Embedded Artists AB
Rundelsgatan 14
SE-211 36 Malmö
Sweden

http://www.EmbeddedArtists.com

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

Revision	*Date*	*Description*
A	2015-11-27	First release
B	2016-11-16	Added section about QSPI. Updated with new COM boards.
C	2017-04-25	Updated PCIe section with new information about i.MX 7
D	2018-11-14	Added sections about ADC & PWM. Added iperf3. Corrected UART mapping for Quad+DualLite boards.
PE1	2019-02-14	Added information about ea-image-base. Added information about COM Carrier Board V2
E	2019-05-08	Added instructions for iMX8M Quad COM
F	2019-10-04	- Added instructions for iMX8M Mini uCOM - Added instructions for iMX7ULP uCOM
G	2020-10-07	- Added instructions for iMX8M Nano uCOM - Updated with u-boot 2020.04 and Linux 5.4.24 info
H	2021-01-25	- Added information about `libgpiod` in section 4.6.2

2 Introduction

This document describes commands to do basic testing of the peripheral interfaces on Embedded Artists i.MX 6/7/8 based COM boards. Different CPUs have different capabilities so each section starts with a table showing what is supported for each CPU.

Additional documentation you might need is.

- The *Getting Started* document for the board you are using.
- The *Adding Displays to iMX Developer's Kits* document about displays and how to use them

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.
```

3 Test Tools

3.1 Introduction

After following the instructions in the *Getting Started* document it is possible to boot into the u-boot or Linux, but how can the different peripheral interfaces available on the iMX Developer's Kits be tested?

This document aims to give short commands to verify the presence and test the functionality of each peripheral interface.

3.2 Requirements

Each test will describe what it requires in the form of cables or other hardware.

The software on the COM Board should be the `ea-image-base` image built with frame buffer support. It includes all the needed packages and is available as a part of the manufacturing tool package on the Embedded Artists iMX Related Resources page in version 4.14.78 and later. Make sure to use the latest version.

For older Linux version, the software on the COM Board should be the `core-image-base` image built with frame buffer support and with the needed packages. It is available as a part of the manufacturing tool package on the Embedded Artists iMX Related Resources page with version numbers up to and including 4.9.123.

To run tests the board must be booted and a terminal program must be used on a host computer to interact with the board.

3.2.1 Own build

The prepared build comes configured with all required packages. To add the packages to another build, add the following lines to the *conf/local.conf* file:

```
IMAGE_INSTALL_append = " \
    i2c-tools-misc \
    i2c-tools \
    pciutils \
    can-utils \
    iproute2 \
    evtest \
    alsa-utils \
    fbida \
    iperf3 \
"
```

Note that this is not needed when building the `ea-image-base` image as it is already included.

4 Tests

4.1 eMMC

The COM Boards have eMMC flash that is used to persistently store everything needed to boot into Linux.

COM board	eMMC device in u-boot			eMMC device in Linux
	<= 2017.03	2018.03	>=2020.04	
iMX6 SoloX COM	mmc dev 1	mmc dev 0	mmc dev 2	/dev/mmcblk2
iMX6 Quad COM	mmc dev 2	mmc dev 1	mmc dev 3	/dev/mmcblk3
iMX6 DualLite COM	mmc dev 2	mmc dev 1	mmc dev 3	/dev/mmcblk3
iMX6 UltraLite COM	mmc dev 1	mmc dev 0	mmc dev 1	/dev/mmcblk1
iMX7 Dual COM	mmc dev 1	mmc dev 1	mmc dev 2	/dev/mmcblk2
iMX7 Dual uCOM	mmc dev 1	mmc dev 1	mmc dev 2	/dev/mmcblk2
iMX7ULP uCOM	N/A	mmc dev 0	mmc dev 0	/dev/mmcblk0
iMX8M Quad COM	N/A	mmc dev 0	mmc dev 0	/dev/mmcblk0
iMX8M Mini uCOM	N/A	mmc dev 1	mmc dev 2	/dev/mmcblk2
iMX8M Nano uCOM	N/A	mmc dev 1	mmc dev 2	/dev/mmcblk2

No extra hardware is needed for this test.

4.1.1 U-boot

One of the roles of the u-boot is to write new bootloader(s), Linux kernel and file systems to eMMC. To accomplish this there are a set of u-boot commands available:

```
=> mmc
mmc - MMC sub system

Usage:
mmc read addr blk# cnt
mmc write addr blk# cnt
mmc erase blk# cnt
mmc rescan
mmc part - lists available partition on current mmc device
mmc dev [dev] [part] - show or set current mmc device [partition]
mmc list - lists available devices
mmc bootbus dev boot_bus_width reset_boot_bus_width boot_mode
 - Set the BOOT_BUS_WIDTH field of the specified device
mmc bootpart-resize <dev> <boot part size MB> <RPMB part size MB>
 - Change sizes of boot and RPMB partitions of specified device
mmc partconf dev boot_ack boot_partition partition_access
 - Change the bits of the PARTITION_CONFIG field of the specified
device
mmc rst-function dev value
 - Change the RST_n_FUNCTION field of the specified device
   WARNING: This is a write-once field and 0 / 1 / 2 are the only
valid values.
mmc setdsr - set DSR register value
```

Take care when using them as they will potentially corrupt the system!

A couple of safe commands (shown for the iMX6 UltraLite COM board):

```
=> mmc dev 1
mmc1(part 0) is current device

=> mmcinfo
Device: FSL_SDHC
Manufacturer ID: fe
OEM: 14e
Name: MMC04
Tran Speed: 52000000
Rd Block Len: 512
MMC version 4.41
High Capacity: Yes
Capacity: 3.5 GiB
Bus Width: 8-bit

=> mmc part

Partition Map for MMC device 1  --   Partition Type: DOS

Part    Start Sector    Num Sectors     UUID            Type
  1     8192            16384           00000000-01     0c
  2     24576           7364608         00000000-02     83
```

4.1.2 Linux

As Linux boots from the eMMC it is already tested when you log in.

To see available disk space:

```
# df -h
Filesystem            Size      Used Available Use% Mounted on
/dev/root             3.4G     65.2M      3.2G   2% /
devtmpfs            340.0M         0    340.0M   0% /dev
tmpfs               500.1M    216.0K    499.9M   0% /run
tmpfs               500.1M     76.0K    500.1M   0% /var/volatile
```

Create a file

```
# echo Hello World > greeting
```

Show the content of the file

```
# cat greeting
Hello World
```

To list files

```
# ls -la
drwxr-xr-x    2 root     root          1024 Sep 23 15:25 .
drwxr-xr-x    3 root     root          1024 Sep 23 14:52 ..
-rw-r--r--    1 root     root            12 Sep 23 15:25 greeting
```

4.2 Network

The COM Carrier Board has two Gigabit Ethernet connectors. Some CPUs only support one Ethernet interface as shown in the table below. The primary Ethernet connector (the only one accessible in the u-boot) is marked in the table below as "left" or "right". Left means the one closest to the HDMI connector.

COM board	Number of Ethernet interfaces in u-boot	Number of Ethernet interfaces in Linux
iMX6 SoloX COM	1 (primary is left)	2 (primary is left)
iMX6 Quad COM	1 (primary is left)	1 (primary is left)
iMX6 DualLite COM	1 (primary is left)	1 (primary is left)
iMX6 UltraLite COM	1 (primary is right)	2 (primary is right)
iMX7 Dual COM	1 (primary is left)	1 (primary is left)
iMX7 Dual uCOM	1 (primary is left)	1 (primary is left)
iMX7ULP uCOM	Not supported by CPU	Not supported by CPU
iMX8M Quad COM	1 (primary is left)	1 (primary is left)
iMX8M Mini uCOM	1 (primary is left)	1 (primary is left)
iMX7M Nano uCOM	1 (primary is left)	1 (primary is left)

This test requires one or two network cables, a network with a DHCP server and access to Internet. The examples assume that the network is 192.168.5.0/255.255.255.255. Replace the IP addresses below to match the network configuration that the board is connected to.

4.2.1 U-boot

The u-boot has basic network functionality but only for the first/primary interface. To test network connectivity use the Ethernet connector as indicated in the table above.

Use the ping command to test the network. It only handles IP addresses, i.e. no host names. It also requires the `ipaddr` variable to have the current IP address.

```
=> setenv ipaddr 192.168.5.7
=> ping 192.168.5.22
Using FEC0 device
host 192.168.5.22 is alive
```

4.2.2 Linux

The Linux image has full support for both Ethernet interfaces and while booting it will initialize and start the first one (eth0). To see the status of the network interface(s):

```
# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:1A:F1:01:9B:E7
          inet addr:192.168.5.71  Bcast:192.168.5.255
mask:255.255.255.0
          inet6 addr: fe80::21a:f1ff:fe01:9be7/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:27 errors:0 dropped:0 overruns:0 frame:0
          TX packets:29 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:2624 (2.5 KiB)  TX bytes:5643 (5.5 KiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

If the CPU supports a second interface (`eth1`), then it can be started with:

```
# ifup eth1
fec 21b4000.ethernet eth1: Freescale FEC PHY driver [Generic PHY]
(mii_bus:phy_addr=2188000.ethernet:02, irq=-1)
IPv6: ADDRCONF(NETDEV_UP): eth1: link is not ready
udhcpc (v1.22.1) started
Sending discover...
Sending discover...
libphy: 2188000.ethernet:02 - Link is Up - 1000/Full
IPv6: ADDRCONF(NETDEV_CHANGE): eth1: link becomes ready
Sending discover...
Sending select for 192.168.5.72...
Lease of 192.168.5.72 obtained, lease time 691200
/etc/udhcpc.d/50default: Adding DNS 192.168.5.2
```

As can be seen above the interface is detected and DHCP is used to get an IP address.

One way to test the network is with the `ping` program. Unlike the u-boot version the Linux version handles host names as well (use `Ctrl-C` to end the program):

```
# 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
64 bytes from 192.36.171.231: seq=3 ttl=56 time=17.355 ms
^C
--- www.sunet.se ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 16.412/17.792/19.125 ms
```

The `ping` command uses the first interface (`eth0`) by default. To specify that it should use another interface use the `-I` option:

```
# ping -I eth1 www.sunet.se
```

When using the second interface (`eth1`) it is possible that the ping program fails. This is most likely because the routing table does not handle the interface. To fix this first look at the current routing table:

```
# route
Kernel IP routing table
Destination  Gateway     Genmask         Flags Metric Ref Use Iface
default      192.168.5.1 0.0.0.0         UG    0      0      0 eth0
192.168.5.0  *           255.255.255.0   U     0      0      0 eth0
192.168.5.0  *           255.255.255.0   U     0      0      0 eth1
```

The default route is only for `eth0`, so remove it and add a default route for `eth1` instead:

```
# route del default
# route add default gw 192.168.5.1 eth1
```

After this change the routing so that eth1 is the default Ethernet interface instead.

4.2.3 Iperf3

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).

If `iperf3` is not available on the file system then it can be added when building the file system. See section 3.2.1 .

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
-----------------------------------------------------------
```

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     Bandwidth       Retr
[  4]   0.00-10.02  sec   146 MBytes   122 Mbits/sec    0    sender
[  4]   0.00-10.02  sec   145 MBytes   122 Mbits/sec         receiver
[  6]   0.00-10.02  sec   151 MBytes   126 Mbits/sec    0    sender
[  6]   0.00-10.02  sec   150 MBytes   126 Mbits/sec         receiver
```

```
[  8]   0.00-10.02  sec   146 MBytes   122 Mbits/sec    0    sender
[  8]   0.00-10.02  sec   145 MBytes   121 Mbits/sec         receiver
[ 10]   0.00-10.02  sec   156 MBytes   131 Mbits/sec    0    sender
[ 10]   0.00-10.02  sec   156 MBytes   130 Mbits/sec         receiver
[SUM]   0.00-10.02  sec   599 MBytes   502 Mbits/sec    0    sender
[SUM]   0.00-10.02  sec   596 MBytes   499 Mbits/sec         receiver
```

The last two lines display the bandwidth for send (502Mbit/sec) and receive (499Mbit/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.

4.3 USB Host

The COM Carrier Board has two USB type A sockets which can be used on all CPUs.

These tests require a USB Memory Stick.

4.3.1 U-boot

The u-boot has USB support for reading/writing USB memories. Connect a USB memory stick to one of the two ports and then issue the following commands:

```
=> usb start
starting USB...
USB0:   Port not available.
USB1:   USB EHCI 1.00
scanning bus 1 for devices... 4 USB Device(s) found
   scanning usb for storage devices... 2 Storage Device(s) found
   scanning usb for ethernet devices... 0 Ethernet Device(s) found

=> usb storage
  Device 0: Vendor: USB      Rev: 1100 Prod: Flash Disk
            Type: Hard Disk
            Capacity: 1912.0 MB = 1.8 GB (3915776 x 512)
  Device 1: Vendor: Kingston Rev: 1.00 Prod: DataTraveler G2
            Type: Removable Hard Disk
            Capacity: 15259.7 MB = 14.9 GB (31252024 x 512)

=> fatls usb 0
 24055271   core-image-base-imx6sxea-com.rootfs.tar.bz2
 79691776   core-image-base-imx6sxea-com.rootfs.ext3
   316520   u-boot-imx6sxea-com.img
  6084744   zimage-imx6sxea-com
    42732   imx6sxea-com-kit.dtb

5 file(s), 0 dir(s)
```

4.3.2 Linux

Linux has support for a wide range of USB devices including mouse, keyboard, memory sticks, hubs etc.

It is possible to see which USB devices are currently connected:

```
# lsusb
Bus 001 Device 004: ID 0951:1624 Kingston Technology DataTraveler G2
Bus 001 Device 003: ID 8087:07dc Intel Corp.
Bus 001 Device 002: ID 0424:2513 Standard Microsystems Corp. 2.0 Hub
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

When a new USB device is connected some status messages will be printed in the console. The following comes when inserting a USB memory stick:

```
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. Section 4.5.2 describes an alternative way to find the device name.

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
```

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
```

There are many different USB devices and the level of support varies with the kind of device. Keyboards will work without any extra work – just plug in and start typing. A mouse will work but without a graphical desktop it will be difficult to use it.

4.4 SD/MMC and uSD Cards

The COM Carrier Board (see column "<V2" in the table below) has two slots for external memory cards – a slot for the uSD card on the top of the carrier board and a slot for the full size SD/ MMC cards on the bottom side. Note that only one of the slots can be used at a time.

The COM Carrier Board V2 (see column "V2" in the table below) has one slot for external memory cards on the top of the carrier board. Note that the uSD card slot is no longer available for the iMX6 UltraLite COM board as it is used for the M.2 connector instead.

The i.MX8M Quad COM board does not support uSD card as the pins are used for the M.2 connector instead.

COM board	MMC/uSD device in u-boot			MMC/uSD device in Linux	
	<= 2017.03	2018.03	>=2020.04	< V2	V2
iMX6 SoloX COM	mmc dev 0	mmc dev 1	mmc dev 3	/dev/mmcblk1	/dev/mmcblk3
iMX6 Quad COM	mmc dev 0	mmc dev 0	mmc dev 2	/dev/mmcblk1	/dev/mmcblk2
iMX6 DualLite COM	mmc dev 0	mmc dev 0	mmc dev 2	/dev/mmcblk1	/dev/mmcblk2
iMX6 UltraLite COM	mmc dev 0	mmc dev 1	N/A	/dev/mmcblk0	N/A
iMX7 Dual COM	mmc dev 0	mmc dev 0	mmc dev 1	/dev/mmcblk0	/dev/mmcblk1
iMX7 Dual uCOM	mmc dev 0	mmc dev 0	mmc dev 1	/dev/mmcblk0	/dev/mmcblk1
iMX7ULP uCOM	N/A	N/A	N/A	N/A	N/A
iMX8M Quad COM	N/A	N/A	N/A	N/A	N/A
iMX8M Mini uCOM	N/A	mmc dev 0	mmc dev 1	N/A	/dev/mmcblk1
iMX7M Nano uCOM	N/A	mmc dev 0	mmc dev 1	N/A	/dev/mmcblk1

A uSD or a full size memory card (SD/MMC) is required to run the tests.

4.4.1 U-boot

The u-boot has support for reading/writing memory cards.

To show information about the memory card it must first be selected with the "mmc dev" command.

```
=> mmc rescan
=> mmc dev 0
=> mmc info
Device: FSL_SDHC
Manufacturer ID: 9
OEM: 4150
Name: AF UD
Tran Speed: 50000000
Rd Block Len: 512
SD version 2.0
High Capacity: No
Capacity: 981.5 MiB
Bus Width: 4-bit
```

To list the content of the sdcard:

```
=> fatls mmc 0
     1491   btngraph.gif
     1396   btnlast.jpg
     1266   btnminus.gif
     1049   btnnext.jpg
     1375   btnplus.gif
```

4.4.2 Linux

A new memory card will be detected automatically when it is inserted and a message like this one will be printed in the console (example is for iMX6 UltraLite COM board so mmc number is 0):

```
mmc0: host does not support reading read-only switch. assuming
write-enable.
mmc0: new high speed SD card at address b368
mmcblk0: mmc1:b368 AF UD 981 MiB
 mmcblk0: p1
```

To use the memory card it must first be mounted:

```
# mkdir /mnt/sdcard
# mount /dev/mmcblk0p1 /mnt/sdcard
```

The card is now mounted:

```
# df -h
Filesystem            Size      Used Available Use% Mounted on
/dev/root             3.4G     65.2M      3.2G   2% /
devtmpfs            340.0M         0    340.0M   0% /dev
tmpfs               500.1M    216.0K    499.9M   0% /run
tmpfs               500.1M     76.0K    500.1M   0% /var/volatile
/dev/mmcblk1p1      981.1M      3.8M    977.4M   0% /mnt/sdcard
```

To see the content:

```
# ls /mnt/scard
DEFCXX.JS          TXTPSET.XML        digi4.gif          metaserv.js
DEFIO01.JS         TXTPSIOA.XML       error.gif          metaset.htm
DEFIO02.JS         TXTPSIOD.XML       excanvas.js        metaset.js
DEFIO03.JS         TXTPSIOS.XML       g_hor1.jpg         metasys.htm
```

A good way to stress test the card is to copy a large file to the sdcard and compute a checksum of it (this should be done on a PC with a known good sdcard reader). It is then possible to calculate that checksum again on the target and make sure it matches.

```
# md5sum /mnt/blob.bin
790bfdcdfac22a08ff27450d60be8a8f  /mnt/blob.bin
```

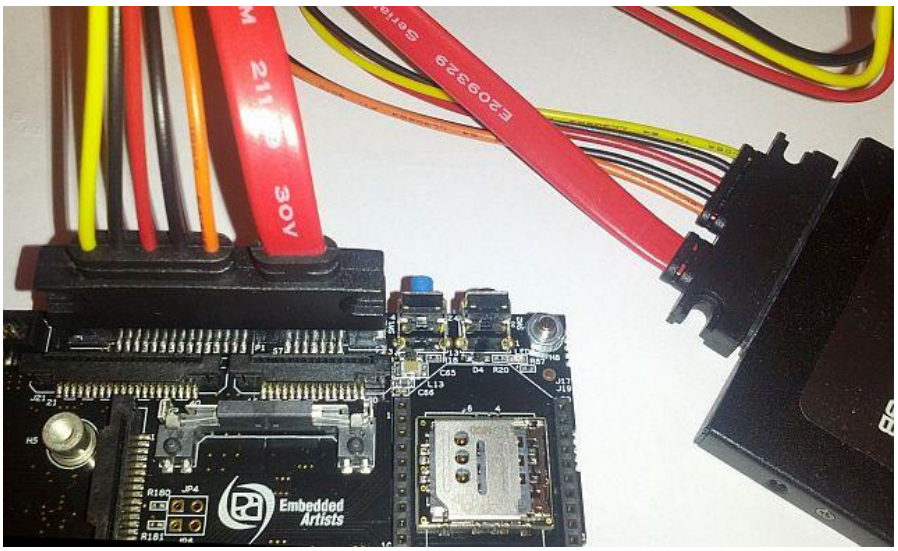
As with the USB memory stick, don't forget to unmount it before physically removing it from the COM Carrier Board:

```
# umount /mnt/scard
```

4.5 SATA

COM board	SATA in u-boot	SATA in Linux
iMX6 SoloX COM	Not Supported by CPU	Not Supported by CPU
iMX6 Quad COM	Not Enabled	Yes
iMX6 DualLite COM	Not Supported by CPU	Not Supported by CPU
iMX6 UltraLite COM	Not Supported by CPU	Not Supported by CPU
iMX7 Dual COM	Not Supported by CPU	Not Supported by CPU
iMX7 Dual uCOM	Not Supported by CPU	Not Supported by CPU
iMX7ULP uCOM	Not Supported by CPU	Not Supported by CPU
iMX8M Quad COM	Not Supported by CPU	Not Supported by CPU
iMX8M Mini uCOM	Not Supported by CPU	Not Supported by CPU
iMX8M Nano uCOM	Not Supported by CPU	Not Supported by CPU

This test requires a SATA disk (can be SSD) and a cable to connect it to the Carrier Board.



For COM Carrier Board V2 the SATA disk interface is on the M.2 connector:

4.5.1 U-boot

The u-boot can be (but is not in the default build) configured with SATA support.

4.5.2 Linux

A SATA disk may have many partitions and to see which devices have been assigned to the disk:

```
# ls -l /dev/disk/by-id/
ata-ADATA_SP550_1F3520275635 -> ../../sda
ata-ADATA_SP550_1F3520275635-part1 -> ../../sda1
ata-ADATA_SP550_1F3520275635-part2 -> ../../sda2
…
```

The interesting lines above are starting with 'ata' and shows that the SATA disk's two partitions are available as sda1 and sda2.

To use one of the partitions it must first be mounted:

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

The card is now mounted:

```
# df -h
Filesystem            Size      Used Available Use% Mounted on
/dev/root            73.5M     57.1M     12.3M  82% /
devtmpfs            340.0M         0    340.0M   0% /dev
tmpfs               500.1M    224.0K    499.9M   0% /run
tmpfs               500.1M     88.0K    500.1M   0% /var/volatile
/dev/sda1            28.8G     32.0K     28.8G   0% /mnt/sata
```

To see the content:

```
# ls /mnt/sata
test_marker4.txt
```

As with the USB memory stick, don't forget to unmount it before physically removing it from the COM Carrier Board:

```
# umount /mnt/sata
```

4.6 GPIO

Many of the available GPIO pins are available on the expansion connectors. Check exactly which pins are available on the COM board used, because they all differ in number of available GPIO pins.

Note that a pin must be declared as a GPIO before it can be used and manipulated as a GPIO. Error messages are not always given if a declaration is missing in the device tree file.

Also note that if a pin is already declared and used in the device tree file, then it is not possible to use it as a general GPIO.

4.6.1 U-boot

Not applicable.

4.6.2 Linux – libgpiod

The `libgpiod` project is a C library and tools for interacting with GPIO character devices. It has been added to `ea-image-base` builds from 2021-01-22.

The `gpiodetect` command can be used to list all GPIO chips that are available. In the example below GPIO chips 0 – 5 are built-in to the processor while GPIO chip 5 is a GPIO expander attached to the I2C bus on COM Carrier Board V2.

```
# gpiodetect
gpiochip0 [30200000.gpio] (32 lines)
gpiochip1 [30210000.gpio] (32 lines)
gpiochip2 [30220000.gpio] (32 lines)
gpiochip3 [30230000.gpio] (32 lines)
gpiochip4 [30240000.gpio] (32 lines)
gpiochip5 [1-0020] (16 lines)
```

The `gpioinfo` command can be used to get more information about a GPIO chip. The command will list all available GPIO lines (pins), the name of the line, how it is configured and if it is already used. In the example below info about GPIO chip 5 is listed. Lines 13-15 show the lines that are connected to the button and LEDs on COM Carrier Board V2 that are described in section 4.6.4 below.

```
# gpioinfo 5
line   0:  "BT_REG_ON"  "?"             output active-high [used]
line   1:  "WL_REG_ON"  "reset"         output active-low  [used]
line   2:  "VBAT_VSEL"  "hog_VBAT_VSEL" output active-high [used]

...

line  13:  "USER_BTN"   unused          output  active-high
line  14:  "USER_LED1"  unused          output  active-high
line  15:  "USER_LED2"  unused          output  active-high
```

The `gpioset` command can be used to set a GPIO line. In the example below `USER_LED1` will turn on.

```
# gpioset 5 14=1
```

The `gpioget` command can be used to read the value of a GPIO line. In the example below the value of `USER_BTN` will be read.

```
# gpioget 5 13
```

More information about `libgpiod` can be found on this link: https://embeddedbits.org/new-linux-kernel-gpio-user-space-interface/

4.6.3 Linux - sysfs

Assuming that a pin (in this example GPIO6_IO13) is configured as GPIO in the device tree file and is not already in used then it can be examined and manipulated from the command line in Linux.

The GPIO pins are controlled with special files in sysfs.

To use a pin it must first be configured as GPIO and to do that the pin's port and pin number must be converted into a number with this formula:

Num = (Port - 1)*32 + Pin

So for GPIO6_IO13 the number is (6 - 1)*32 + 13 = 173.

To configure the pin:

```
# echo 173 > /sys/class/gpio/export
```

If that is successful then a new folder should have been created (see gpio173 below):

```
# ls /sys/class/gpio/
export       gpiochip0    gpiochip160  gpiochip32   gpiochip96
gpio173      gpiochip128  gpiochip192  gpiochip64   unexport
```

A closer inspection of the exported GPIO:

```
# ls /sys/class/gpio/gpio173/
active_low  direction   power       uevent
device      edge        subsystem   value
```

To configure the pin as an input and read the current value (0 or 1):

```
# echo in > /sys/class/gpio/gpio173/direction
# cat /sys/class/gpio/gpio173/value
0
```

To configure the pin as an output and set it high:

```
# echo out > /sys/class/gpio/gpio173/direction
# echo 1 > /sys/class/gpio/gpio173/value
```

or

```
# echo high > /sys/class/gpio/gpio173/direction
```

To configure the pin as output and set it low:

```
# echo out > /sys/class/gpio/gpio173/direction
# echo 0 > /sys/class/gpio/gpio173/value
```

or

```
# echo low > /sys/class/gpio/gpio173/direction
```

4.6.4 COM Carrier Board V2 - sysfs

COM Carrier Board V2 has a button and two LEDs that are connected to an I2C controlled GPIO expander. From a user perspective this does not matter as they can be manipulated in the same way as a normal GPIO pin on the CPU.

COM board	Label	GPIO
Button (SW4)	USER_BTN	509
LED19	USER_LED1	510
LED20	USER_LED2	511

The GPIO numbers are not assigned with the same formula as was described in 4.6.2 . To see which number is assigned to the button and the LEDs:

```
# grep USER /sys/kernel/debug/gpio
 gpio-509 (USER_BTN            )
 gpio-510 (USER_LED1           )
 gpio-511 (USER_LED2           )
```

To read the button state (0 = button pressed, 1 = button not pressed):

```
# echo 509 > /sys/class/gpio/export
# echo in > /sys/class/gpio/gpio509/direction
# cat /sys/class/gpio/gpio509/value
 1
```

A simple way to do repeated reading of the button once per second (abort with Ctrl+C):

```
# watch -n 1 cat /sys/class/gpio/gpio509/value
Every 1s: cat /sys/class/gpio/gpio509/value    2019-02-08 08:04:25

1
```

The LEDs are on if the value is 1 and off if it is 0 so to turn LED19 on and then off:

```
# echo 510 > /sys/class/gpio/export
# echo high > /sys/class/gpio/gpio510/direction
# echo low > /sys/class/gpio/gpio510/direction
```

4.6.5 COM Carrier Board V2 – libgpiod

See section 4.6.2 for more information about how to use `libgpiod` with the USER_BTN, USER_LED1, and USER_LED2.

4.7 I2C

Each CPU supports a number of I2C bus interfaces. In most cases they are all available on the standard EACOM locations. In some cases, more I2C interfaces exist on the CPU and can be enabled in the device tree file. The table below is a mapping between the bus numbering in u-boot / Linux and the I2C channel name (A/B/C) used on the COM Carrier board. Note that the bus numbering in u-boot/Linux is not necessarily the same as the peripheral number on the processor.

COM board	I2C-A	I2C-B	I2C-C
iMX6 SoloX COM	i2c-0	i2c-1	i2c-2
iMX6 Quad COM	i2c-0	i2c-2	i2c-1
iMX6 DualLite COM	i2c-0	i2c-2	i2c-1
iMX6 UltraLite COM	i2c-0	i2c-1	Not Supported
iMX7 Dual COM	i2c-0	i2c-1	i2c-2
iMX7 Dual uCOM	i2c-0	i2c-1	i2c-2
iMX7ULP uCOM	i2c-0 / bus 5 *)	i2c-1 / bus 7 *)	Not Supported
iMX8M Quad COM	i2c-0	i2c-1	Not Supported
iMX8M Mini uCOM	i2c-0	i2c-1	i2c-2
iMX7M Nano uCOM	i2c-0	i2c-1	i2c-2

*) For iMX7ULP uCOM the i2c bus numbering is not the same in u-boot and Linux. The first name in the column corresponds to device name in Linux and the second the device name in u-boot.

Which I2C devices that can be found varies depending on COM board, COM Carrier Board and connected peripherals like displays (typically the touch controller uses I2C).

The 7-bit address of all I2C devices on the COM Carrier board can be found in the schematics for the COM Carrier board.

No extra hardware is needed to run these tests.

4.7.1 U-boot

The u-boot has I2C commands:

```
=> i2c
i2c - I2C sub-system

Usage:
i2c bus [muxtype:muxaddr:muxchannel] - show I2C bus info
crc32 chip address[.0, .1, .2] count - compute CRC32 checksum
i2c dev [dev] - show or set current I2C bus
i2c loop chip address[.0, .1, .2] [# of objects] - looping read of device
i2c md chip address[.0, .1, .2] [# of objects] - read from I2C device
i2c mm chip address[.0, .1, .2] - write to I2C device (auto-incrementing)
i2c mw chip address[.0, .1, .2] value [count] - write to I2C device (fill)
i2c nm chip address[.0, .1, .2] - write to I2C device (constant address)
i2c probe [address] - test for and show device(s) on the I2C bus
i2c read chip address[.0, .1, .2] length memaddress - read to memory
i2c write memaddress chip address[.0, .1, .2] length - write memory to i2c
i2c reset - re-init the I2C Controller
i2c speed [speed] - show or set I2C bus speed
```

To see the available busses:

```
=> i2c bus
Bus 0:  mxc0
Bus 1:  mxc1
Bus 2:  mxc2
```

Bus 0 is safe to use. The other busses may not be initialized and could produce errors when probed. To list all devices on bus 0:

```
=> i2c dev 0
Setting bus to 0

=> i2c probe
Valid chip addresses: 08 1A 4D 55 56
```

The scan found the following devices:

- 0x08 – PMIC on the COM board
- 0x1a – Audio Codec on the COM Carrier Board
- 0x4d – AR1021 Touch Controller, typically on COM Display Adapter (or on COM Carrier Board, rev A)
- 0x55 – EEPROM on the COM Board
- 0x56 – EEPROM, typically on COM Display Adapter (or on COM Carrier Board, rev A)

Note that your result may be different depending on which combination of COM board, COM Carrier Board and external peripherals you use.

4.7.2 Linux

To see which I2C busses are available use either

```
# ls /dev/i2c*
/dev/i2c-0  /dev/i2c-1  /dev/i2c-2
```

or

```
# i2cdetect -l
i2c-0   i2c         21a0000.i2c       I2C adapter
i2c-1   i2c         21a4000.i2c       I2C adapter
i2c-2   i2c         21a8000.i2c       I2C adapter
```

To scan for all devices on i2c-1:

```
# i2cdetect -y 0
root@imx6sxea-com:~# i2cdetect -y 0
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- UU -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- UU -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- UU -- --
50: -- -- -- -- -- UU 56 -- -- -- -- -- -- -- -- --
```

```
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

The scan above uses UU to indicate that a device was not probed as it was marked as being in use by a driver. The address to device mapping is described in the u-boot section above.

4.8 UART

4.8.1 COM Carrier Board

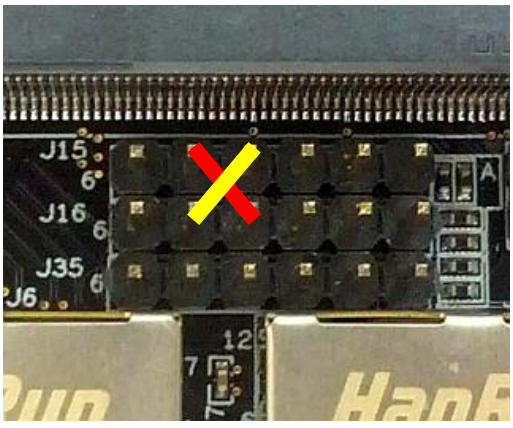
The COM Carrier Board has three UARTs:

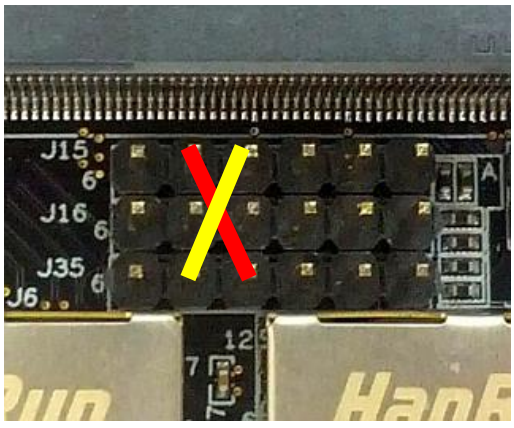
COM board	NXP Name	Linux	On COM Carrier Board
iMX6 SoloX COM	UART1	/dev/ttymxc0 (console)	Pin list J35 (UART-A)
	UART2	/dev/ttymxc1	Pin list J15 (UART-B)
	UART5	/dev/ttymxc4	Pin list J16 (UART-C)
iMX6 Quad COM	UART1 *	/dev/ttymxc0 *	Pin list J16 (UART-C)
	UART4 *	/dev/ttymxc3 (console) *	Pin list J35 (UART-A)
	UART5	/dev/ttymxc4	Pin list J15 (UART-B)
iMX6 DualLite COM	UART1 *	/dev/ttymxc0 *	Pin list J16 (UART-C)
	UART4 *	/dev/ttymxc3 (console) *	Pin list J35 (UART-A)
	UART5	/dev/ttymxc4	Pin list J15 (UART-B)
iMX6 UltraLite COM	UART1	/dev/ttymxc0 (console)	Pin list J35 (UART-A)
	UART2	/dev/ttymxc1	Pin list J15 (UART-B)
	UART3	/dev/ttymxc2	Pin list J16 (UART-C)
iMX7 Dual COM	UART1	/dev/ttymxc0 (console)	Pin list J35 (UART-A)
	UART2	/dev/ttymxc1	Pin list J15 (UART-B)
	UART3	/dev/ttymxc2	Pin list J16 (UART-C)
iMX7 Dual uCOM	UART1	/dev/ttymxc0 (console)	Pin list J35 (UART-A)
	UART2	/dev/ttymxc1	Pin list J15 (UART-B)
	UART3	/dev/ttymxc2	Pin list J16 (UART-C)

*) The console uses UART4 on iMX6 Quad and iMX6 DualLite starting with u-boot version 2018.03 and Linux version 4.14.78. Older u-boot and Linux versions use UART1

UART-A is used by the console so it is tested by connecting a terminal program to the port and then boot into the u-boot.

Two jumper cables are needed to run these tests. For iMX6 Quad and DualLite COM boards running an older version of u-boot and Linux use UART-C as console, connect as in the right image below. For all other COM boards or connect as shown in the left image below.



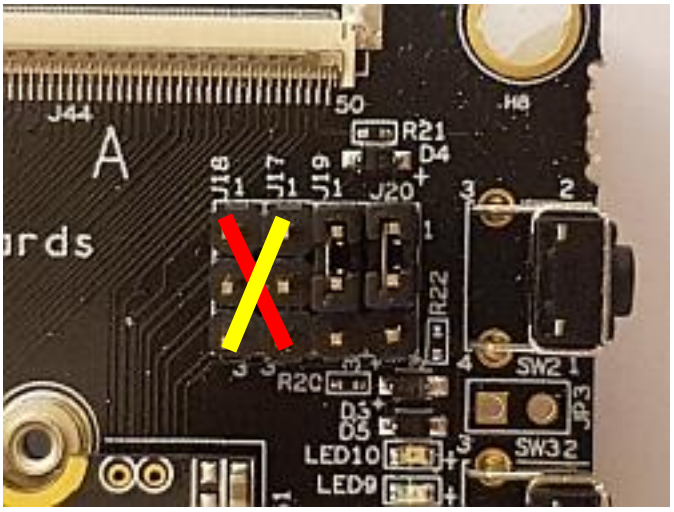


4.8.2 COM Carrier Board V2

COM Carrier Board V2 has an onboard Dual USB to serial IC making the external FTDI cable obsolete. The default position of the jumpers and the settings in both Linux and the u-boot have been made so that the console always ends up on UART-A.

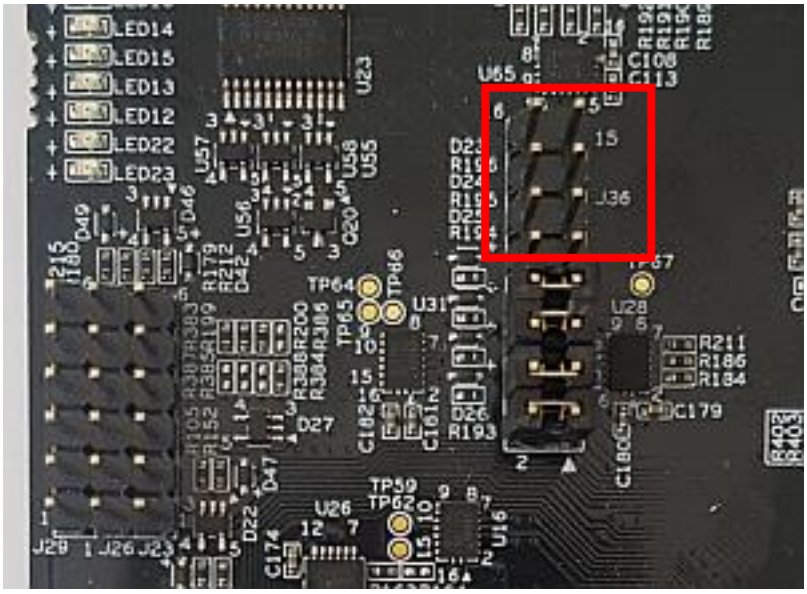
COM board	NXP Name	Linux	On COM Carrier Board V2
iMX6 SoloX COM	UART1	/dev/ttymxc0 (console)	UART-A
	UART2	/dev/ttymxc1	UART-B
	UART5	/dev/ttymxc4	UART-C
iMX6 Quad COM	UART1	/dev/ttymxc0	UART-C
	UART4	/dev/ttymxc3 (console)	UART-A
	UART5	/dev/ttymxc4	UART-B
iMX6 DualLite COM	UART1	/dev/ttymxc0	UART-C
	UART4	/dev/ttymxc3 (console)	UART-A
	UART5	/dev/ttymxc4	UART-B
iMX6 UltraLite COM	UART1	/dev/ttymxc0 (console)	UART-A
	UART2	/dev/ttymxc1	UART-B
	UART3	/dev/ttymxc2	UART-C
iMX7 Dual COM	UART1	/dev/ttymxc0 (console)	UART-A
	UART2	/dev/ttymxc1	UART-B
	UART3	/dev/ttymxc2	UART-C
iMX7 Dual uCOM	UART1	/dev/ttymxc0 (console)	UART-A
	UART2	/dev/ttymxc1	UART-B
	UART3	/dev/ttymxc2	UART-C
iMX7ULP uCOM	LPUART4	/dev/ttyLP0 (console)	UART-A
	LPUART6	/dev/ttyLP2	UART-B
iMX8M Quad COM	UART1	/dev/ttymxc0 (console)	UART-A
	UART2	/dev/ttymxc1	UART-B
	UART3	/dev/ttymxc2	UART-C
iMX8M Mini uCOM	UART1	/dev/ttymxc0	UART-B
	UART2	/dev/ttymxc1 (console)	UART-A
iMX8M Nano uCOM	UART1	/dev/ttymxc0	UART-B
	UART2	/dev/ttymxc1 (console)	UART-A

Two jumper cables are needed for this test. Connect as shown in the image below (J17-1 to J18-3, J17-3 to J18-1). Please note that for boards that only support two UARTs (such as iMX8M Mini) it is not possible to do this test.



Before doing the UART test, four jumpers must be removed. It is the top four positions for J36 as shown in the figure below. When the jumpers are inserted the UART interfaces will be used by the Bluetooth interface on the M.2 connector.

Note: This does not work on revision PE23 of the COM Carrier Board V2 since the jumpers are not available on that revision.



4.8.3 U-boot

No special tests to run in the u-boot. The console gets tested automatically and the other UARTs are not available

4.8.4 Linux

The examples below are for the iMX6 UltraLite COM board, so replace the devices according to the CPU you are testing.

To see which UARTs are available:

```
# ls /dev/ttymxc*
/dev/ttymxc0  /dev/ttymxc1  /dev/ttymxc2
```

The console uses /dev/ttymxc0, so it does not need further testing.

To test /dev/ttymxc1 and /dev/ttymxc2 cross-connect the RX and TX lines on the COM Carrier Board as shown above.

After connecting, set them up with a baud rate of 115200, raw mode and no echoing:

```
# stty -F /dev/ttymxc1 115200 raw -echo
# stty -F /dev/ttymxc2 115200 raw -echo
```

To listen on /dev/ttymxc1 and send on /dev/ttymxc2:

```
# cat /dev/ttymxc1 &
# echo Hello World > /dev/ttymxc2
Hello World
```

The & in the first command above means that it will be executed in the background. Don't forget to stop the background process when you're done with it by bringing it to the foreground and then pressing Ctrl+C:

```
# fg
cat /dev/ttymxc1
^C
```

4.9 PCI

COM board	PCI in u-boot	PCI in Linux
iMX6 SoloX COM	No	Yes
iMX6 Quad COM	No	Yes
iMX6 DualLite COM	No	Yes
iMX6 UltraLite COM	Not Supported by CPU	Not Supported by CPU
iMX7 Dual COM	No	Yes
iMX7 Dual uCOM	No	Yes
iMX7ULP uCOM	Not Supported by CPU	Not Supported by CPU
iMX8M Quad COM	No	Yes
iMX8M Mini uCOM	No	Yes
iMX8M Nano uCOM	Not Supported by CPU	Not Supported by CPU

For a COM Carrier Board this test assumes that an Intel™ Dual Band Wireless-AC 7260 Plus Bluetooth (http://www.intel.com/content/www/us/en/wireless-products/dual-band-wireless-ac-7260-bluetooth.html) PCIe half mini card is inserted before booting into Linux.

For a COM Carrier Board V2 the test assumes that an Embedded Artists 1CX M.2 Module is inserted in the M.2 connector before booting into Linux. Also note that the PCIe support must be enabled by selecting the correct dtb file in the u-boot. For an iMX6 DualLite the commands are:

```
=> setenv fdt_file imx7dlea-com-kit_v2-pcie.dtb
=> saveenv
```

4.9.1 U-boot

Not applicable.

4.9.2 Linux

To see if the board is detected, use the `lspci` command:

```
# lspci
00:00.0 PCI bridge: Device 16c3:abcd (rev 01)
01:00.0 Network controller: Intel Corporation Wireless 7260 (rev bb)
```

If the 1CX M.2 module is used it will look like this:

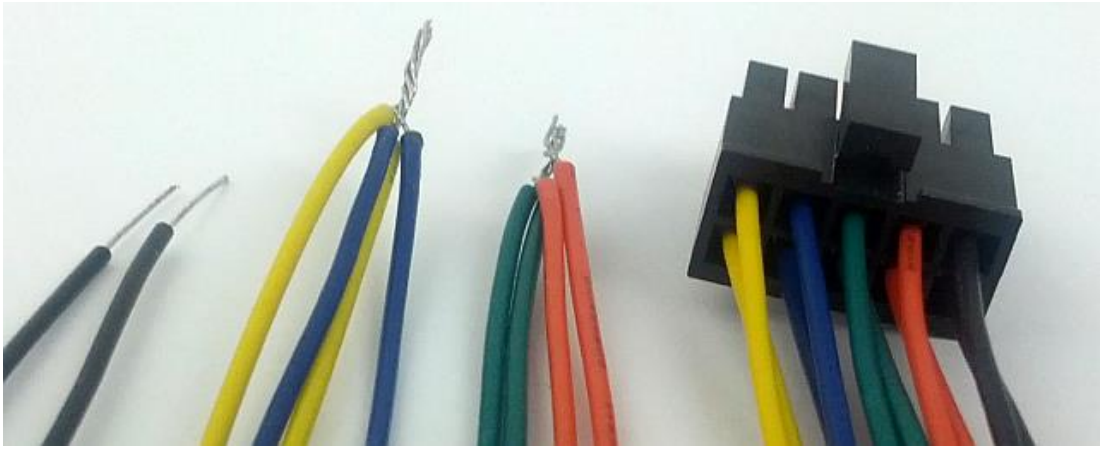
```
# lspci
00:00.0 PCI bridge: Synopsys, Inc. Device abcd (rev 01)
01:00.0 Network controller: Broadcom Limited BCM4356 802.11ac Wireless
Network Adapter (rev 02)
```

4.10 CAN

COM board	CAN in u-boot	CAN in Linux
iMX6 SoloX COM	Not Supported	Yes
iMX6 Quad COM	Not Supported	Yes
iMX6 DualLite COM	Not Supported	Yes
iMX6 UltraLite COM	Not Supported	Yes
iMX7 Dual COM	Not Supported	Yes
iMX7 Dual uCOM	Not Supported	Yes
iMX7ULP uCOM	Not Supported by CPU	Not Supported by CPU
iMX8M Quad COM	Not Supported by CPU	Not Supported by CPU
iMX8M Mini uCOM	Not Supported by CPU	Not Supported by CPU
iMX8M Nano uCOM	Not Supported by CPU	Not Supported by CPU

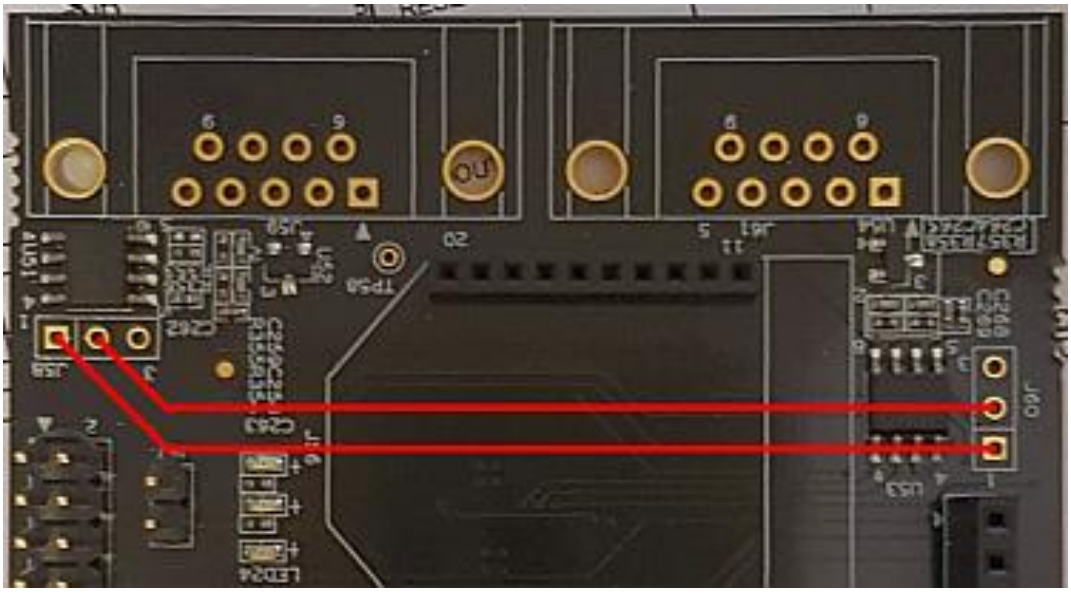
4.10.1 COM Carrier Board

To run these tests the two can busses must be connected together. Use the cable that comes with the Carrier Board. Twist together the yellow and blue wires (CANH). Twist together the orange and green wires (CANL) as shown in the image below.



4.10.2 COM Carrier Board V2

Two CAN interfaces are located on the Expansion Board (see User's Manual). The test below is based in connecting the two busses together which can be done with jumper cables as shown here:



Note that for iMX6 UltraLite CAN bus 2 has been disabled in the device tree file as it conflicts with one of the UARTs. Without modifying the device tree there is no way to run the CAN tests.

4.10.3 U-boot

Not applicable.

4.10.4 Linux

Make sure that both `can0` and `can1` are detected:

```
# ip link show
...
2: can0: <NOARP,ECHO> mtu 16 qdisc noop state DOWN mode DEFAULT
    group default qlen 10
    link/can
3: can1: <NOARP,ECHO> mtu 16 qdisc noop state DOWN mode DEFAULT
    group default qlen 10
    link/can
...
```

If the text <NOARP,ECHO> appears then the interface is down and must be brought up before it can be used:

```
# ip link set can0 up type can bitrate 125000
flexcan 2090000.can can0: writing ctrl=0x0e312005

# ip link set can1 up type can bitrate 125000
flexcan 2094000.can can1: writing ctrl=0x0e312005
```

As the interfaces are now up, the status will have changed:

```
# ip link show
...
2: can0: <NOARP,UP,LOWER_UP,ECHO> mtu 16 qdisc pfifo_fast state
    UNKNOWN mode DEFAULT group default qlen 10
    link/can
```

```
3: can1: <NOARP,UP,LOWER_UP,ECHO> mtu 16 qdisc pfifo_fast state
    UNKNOWN mode DEFAULT group default qlen 10
    link/can
...
```

To test the CAN bus the two interfaces must be connected together as shown above.

The following example listens on `can1` and sends on `can0`. Start by listening on the `can1` interface in the background:

```
# candump can1 &
```

Now send a message on `can0` and see what arrives on `can1`:

```
# cansend can0 5A1#11.2233.44556677.88
  can1  5A1   [8]  11 22 33 44 55 66 77 88
```

The second line above is from the `candump` process and it shows that it has detected an 8 byte message with id 5A1 being received by the `can1` interface.

After completing the tests, terminate the background process with the `fg` command and `Ctrl+C`

```
# fg
candump can1
^C
```

4.11 Audio Out

There is a headphone jack on the COM Carrier Board. The COM Carrier Board V2 has both a headphone and a Line Out jack. Audio out is supported in Linux on all CPUs, but not in the u-boot.

This test requires either headphones or speakers with a 3.5mm audio jack.

4.11.1 U-boot

Not applicable.

4.11.2 Linux

Use headphones in the jacket on the COM Carrier Board.

The `alsa-utils` package comes with a set of sample sound files available on the file system:

```
# ls /usr/share/sounds/alsa/
Front_Center.wav  Noise.wav         Rear_Right.wav
Front_Left.wav    Rear_Center.wav   Side_Left.wav
Front_Right.wav   Rear_Left.wav     Side_Right.wav
```

The sound files can be played using the `aplay` command:

```
# aplay /usr/share/sounds/alsa/Front_Left.wav
```

Note that even if the file is mono, it will be played as stereo.

Another way to test the audio is with the `speaker-test` application:

```
# speaker-test -c2 -l2 -twav
```

The speaker-test application has a lot of options to play with, but in our example the options are `-c2` for stereo, `-l2` to play each sound twice and `-twav` for wav-file testing.

If there is no sound at all, it is most likely because the output is off or muted. To get a long list of all mixer controls:

```
# amixer
Simple mixer control 'Master',0
  Capabilities: pvolume
  Playback channels: Front Left - Front Right
  Limits: Playback 0 - 127
  Mono:
  Front Left: Playback 101 [80%] [-20.00dB]
  Front Right: Playback 101 [80%] [-20.00dB]
...
Simple mixer control 'Output Mixer HiFi',0
  Capabilities: pswitch pswitch-joined
  Playback channels: Mono
  Mono: Playback [off]
...
```

Or a shorter list of the names:

```
# amixer controls
numid=2,iface=MIXER,name='Master Playback ZC Switch'
numid=1,iface=MIXER,name='Master Playback Volume'
numid=4,iface=MIXER,name='Line Capture Switch'
numid=5,iface=MIXER,name='Mic Boost Volume'
numid=6,iface=MIXER,name='Mic Capture Switch'
numid=8,iface=MIXER,name='ADC High Pass Filter Switch'
numid=3,iface=MIXER,name='Capture Volume'
numid=10,iface=MIXER,name='Playback Deemphasis Switch'
numid=11,iface=MIXER,name='Input Mux'
numid=14,iface=MIXER,name='Output Mixer HiFi Playback Switch'
numid=12,iface=MIXER,name='Output Mixer Line Bypass Switch'
numid=13,iface=MIXER,name='Output Mixer Mic Sidetone Switch'
numid=7,iface=MIXER,name='Sidetone Playback Volume'
numid=9,iface=MIXER,name='Store DC Offset Switch'
```

On the core-image-base build that you should be running, the only mixer setting that must be changed is the "Output Mixer HiFi Playback Switch" which is turned off by default. To enable it look for the "numid=xx" in the list above to find that it is "numid=14". Enable with the following commands:

```
# amixer -q cset numid=14 on
```

If for some reason there is still no sound, repeat the `amixer` command for all channels that have the word "Playback" in the name.

4.12 Audio In (COM Carrier Board V2 only)

COM Carrier Board V2 has a microphone jack and line in. The audio output is described in the previous section and should be verified to work before attempting the audio input tests in this section.

4.12.1 U-boot

Not applicable

4.12.2 Linux

As described it the Audio Out section it is possible to see all the audio mixer controls:

```
# amixer controls
numid=2,iface=MIXER,name='Master Playback ZC Switch'
numid=1,iface=MIXER,name='Master Playback Volume'
numid=4,iface=MIXER,name='Line Capture Switch'
numid=5,iface=MIXER,name='Mic Boost Volume'
numid=6,iface=MIXER,name='Mic Capture Switch'
numid=8,iface=MIXER,name='ADC High Pass Filter Switch'
numid=3,iface=MIXER,name='Capture Volume'
numid=10,iface=MIXER,name='Playback Deemphasis Switch'
numid=14,iface=MIXER,name='Input Mux'
numid=13,iface=MIXER,name='Output Mixer HiFi Playback Switch'
numid=11,iface=MIXER,name='Output Mixer Line Bypass Switch'
numid=12,iface=MIXER,name='Output Mixer Mic Sidetone Switch'
numid=7,iface=MIXER,name='Sidetone Playback Volume'
numid=9,iface=MIXER,name='Store DC Offset Switch'
```

Line Bypass

The audio codec on the COM Carrier Board V2 can be put into Line Bypass mode which connects Line In with Line Out. To test this connect an audio source (e.g. phone or PC) to the Line In jack and connect speakers to the Line Out jack.

To start the Line Bypass:

```
# amixer -q cset numid=11 on
```

To stop the Line Bypass:

```
# amixer -q cset numid=11 off
```

Audio Recording

Another way to test audio is to record from the microphone and then play it back.

Enable microphone:

```
# amixer -q cset numid=14 Mic
# amixer -q cset numid=6 on
```

Record 5 seconds in CD quality:

```
# arecord -d 5 -fcd myfile.wav
```

Disable microphone:

```
# amixer -q cset numid=6 off
```

Play recorded audio:

```
# aplay myfile.wav
```

If you don't hear any audio check the settings in the Audio Out section above.

4.13 Display Output

Each CPU supports a different set of display options. The table below shows what is supported and which display is the default.

COM board	Parallel RGB *	LVDS0	LVDS1	HDMI	MIPI DSI **
iMX6 SoloX COM	Yes	Yes (default)	Not supported	Not supported	Not supported
iMX6 Quad COM	Yes	Yes	Yes (default)	Yes	Yes
iMX6 DualLite COM	Yes	Yes	Yes (default)	Yes	Yes
iMX6 UltraLite COM	Yes (default)	Not supported	Not supported	Not supported	Not supported
iMX7 Dual COM	Yes (default)	Not supported	Not supported	Not supported	Yes
iMX7 Dual uCOM	Yes (default)	Not supported	Not supported	Not supported	Yes
iMX7ULP uCOM	Not supported	Not supported	Not supported	Yes ****)	Yes
iMX8M Quad COM	Not supported	Not supported	Not supported	Yes ***)	Yes
iMX8M Mini uCOM	Not supported	Not supported	Not supported	Yes ****)	Yes
iMX8M Nano uCOM	Not supported	Not supported	Not supported	Yes ****)	Yes

*) The Parallel RGB interface is available through the use of a COM Display Adapter board

**) The MIPI DSI interface is not controlled by the `eadisp` command described below

***) The iMX8M does not have the `eadisp` command described below enabled as of May 2019

****) Via MIPI-DSI to HDMI bridge on the uCOM Adapter board

The Display Solutions for COM Boards page describes the different interfaces, how to physically connect a display and it has a list of some of the supported displays. There is also the *Adding Displays to iMX Developer's Kit* document which has more in-depth descriptions of the commands below.

4.13.1 U-boot

The u-boot has support for the `eadisp` command to control which display interfaces should be enabled and how they should be configured.

The u-boot is setup to show the DENX™ logo on the default display when booting but to see it the display has to be configured correctly.

Run the `eadisp` command to see available options (output is for the iMX6 Quad COM board):

```
=> eadisp

Available display configurations:
    0) lvds0  hannstar:18:64998375,1024,768,220,40,21,7,60,...
    1) lvds1  hannstar:18:64998375,1024,768,220,40,21,7,60,...
    2)   rgb  Innolux-AT070TN:24:33336667,800,480,89,164,75,75,...
    3)   rgb  nhd-4.3-480272ef:24:9009009,480,272,2,2,2,2,41,...
    4)   rgb  nhd-5.0-800480tf:24:29232073,800,480,40,40,29,13,...
    5)   rgb  nhd-7.0-800480ef:24:29232073,800,480,40,40,29,13,...
    6)   rgb  umsh-8864:24:9061007,480,272,20,20,20,20,...
    7)   rgb  umsh-8596-30t:24:33264586,800,480,128,120,20,20,...
    8)   rgb  umsh-8596-33t:24:32917475,800,480,200,200,45,45,...
    9)   rgb  rogin-rx050a:24:32917475,800,480,200,200,45,45,...
   10)  hdmi  1280x720M@60:m24:74161969,1280,720,220,110,20,5,...
```

```
   11)  hdmi  1920x1080M@60:m24:148500148,1920,1080,148,88,36,...
   12)  hdmi  640x480M@60:m24:25200342,640,480,48,16,33,10,...
   13)  hdmi  720x480M@60:m24:27027027,720,480,60,16,30,9,62,...

Current Selection:
                enabled prefer  configuration
        rgb:      no      no    Innolux-AT070TN:24:33336667,...
        lvds0:    no      no    hannstar:18:64998375,...
        lvds1:    no      no    hannstar:18:64998375,...
        hdmi:     no      no    1280x720M@60:m24:74161969,...
```

The command is described in detail in the *Adding Displays to iMX Developer's Kit* document. An example enabling the Parallel RGB interface to use the UMSH-8864 display:

```
=> eadisp enable rgb
=> eadisp conf rgb 6

selecting rgb=umsh-8864

Current Selection:
                enabled prefer  configuration
        rgb:      yes     no    umsh-8864:24:9061007,...
        lvds0:    no      no    hannstar:18:64998375,...
        lvds1:    no      no    hannstar:18:64998375,...
        hdmi:     no      no    1280x720M@60:m24:74161969,...
```

To make the change, save the environment variables and reset the board:

```
=> saveenv
=> reset
```

The display should show the DENX™ logo.

4.13.2 Linux

Each display has its own framebuffer and to see the available framebuffers:

```
# ls /dev/fb*
/dev/fb0  /dev/fb1  /dev/fb2  /dev/fb3  /dev/fb4  /dev/fb5
```

The number of framebuffers depends on the CPU and what was enabled by the eadisp command in the u-boot.

The i.MX 6Quad and 6DualLite SoCs have support for virtual displays (called overlays) which will have their own framebuffers so an iMX6 Quad COM board with LVDS0 and RGB enabled will have four frame buffers:

```
# ls /dev/fb*
/dev/fb0  /dev/fb1  /dev/fb2  /dev/fb3
```

The overlays are /dev/fb1 and /dev/fb3.

To see the resolution, bit depth etc for a framebuffer:

```
# fbset -fb /dev/fb0
mode "800x480-49"
        # D: 33.501 MHz, H: 31.515 kHz, V: 49.243 Hz
        geometry 800 480 800 480 32
        timings 29850 89 164 75 75 10 10
        accel false
        rgba 8/16,8/8,8/0,0/0
endmode
```

The information above tells us the following interesting information:

- The resolution is 800x480 pixels
- Each pixel uses 32 bits, with 8 bits each for red, green and blue. No alpha information.

For a display with 16 bits per pixel and 1024x768 it will look like this instead:

```
# fbset -fb /dev/fb1
mode "1024x768-60"
        # D: 65.003 MHz, H: 48.365 kHz, V: 60.006 Hz
        geometry 1024 768 1024 768 16
        timings 15384 220 40 21 7 60 10
        accel false
        rgba 5/11,6/5,5/0,0/0
endmode
```

Another way to see display information is to look at the files on sysfs:

```
# ls /sys/class/graphics/fb4
bits_per_pixel         fsl_disp_property      state
blank                  mode                   stride
console                modes                  subsystem
cursor                 name                   uevent
dev                    pan                    virtual_size
device                 power
fsl_disp_dev_property  rotate
```

The files can be investigated further.

```
# cat /sys/class/graphics/fb4/fsl_disp_dev_property
lcd

# cat /sys/class/graphics/fb4/fsl_disp_property
1-layer-fb
```

The displays may have power saving options that turns them off after a while. To turn the display back on, write a 0 to the blank control:

```
# echo 0 > /sys/class/graphics/fb0/blank
```

To turn it off, write a 1 instead:

```
# echo 1 > /sys/class/graphics/fb0/blank
```

A quick test is to send data directly to the framebuffer. Assuming the fbset command revealed a 800x480 display with 32 bit addressing, the following command fills the display with random data:

```
# dd if=/dev/urandom of=/dev/fb0 bs=3200 count=480
```

The display can be cleared again by writing zeroes to it:

```
# dd if=/dev/zero of=/dev/fb0 bs=3200 count=480
```

A more practical test is to display an image. Use a USB memory stick and mount it as described in section 4.3.2 . To display an image use the following command:

```
# fbi -T 2 -d /dev/fb0 /mnt/usbstick/*.png
```

4.14 Touch

The interface between the COM Display Adapter board and the COM Carrier Board has an I2C channel for touch controllers either on the COM Display Adapter board (i.e. the AR1021) or on the attached display itself. The LVDS interface on the COM Carrier Board includes an I2C channel as well. HDMI and MIPI-DSI displays can have touch controllers but they will have to be connected using an additional interface (e.g. USB) and that is not handled in this document as it is display specific.

The Display Solutions for COM Boards page describes the different interfaces, how to physically connect a display and it has a list of some of the supported displays. There is also the *Adding Displays to iMX Developer's Kit* document which has more in-depth descriptions of the commands below.

4.14.1 U-boot

The u-boot does not support touch events by itself but it is used to configure the touch interface(s) that Linux will use. The `eatouch` command to control which display interfaces should be enabled and how they should be configured.

Run the `eatouch` command to see available options (output is for the iMX6 Quad COM board):

```
=> eatouch

Available Touch Controllers:
    1) ar1021
    2) ilitek
    3) sitronix
    4) egalax
    5) ft5x06

Current Setup:
                        rgb conn.       lvds0 conn.     lvds1 conn.
            ar1021      Disabled        Disabled        Disabled
            ilitek      Disabled        Disabled        Disabled
          sitronix      Disabled        Disabled        Disabled
            egalax      Disabled        Disabled        Disabled
            ft5x06      Disabled        Disabled        Disabled
```

The `eatouch` command is described in detail in the *Adding Displays to iMX Developer's Kit* document. To enable the AR1021 touch controller (used by a display with resistive touch panel connected via the parallel RGB interface):

```
=> eatouch enable rgb 1

Current Setup:
                        rgb conn.       lvds0 conn.     lvds1 conn.
            ar1021   Enabled 0x4d       Disabled        Disabled
            ilitek      Disabled        Disabled        Disabled
          sitronix      Disabled        Disabled        Disabled
            egalax      Disabled        Disabled        Disabled
            ft5x06      Disabled        Disabled        Disabled
```

To make the change, save the environment variables and reset the board:

```
=> saveenv
=> reset
```

4.14.2 Linux

To see which devices are available:

```
# evtest

No device specified, trying to scan all of /dev/input/event*
Available devices:
/dev/input/event0:      20cc000.snvs-pwrkey
/dev/input/event1:      ar1021 I2C Touchscreen
/dev/input/event2:      EETI eGalax Touch Screen
```

In this case there are two touch controllers enabled: AR1021 and eGalax.

Use the `evtest` program again to test if a touch controller works:

```
# evtest /dev/input/event1
Input driver version is 1.0.1
Input device ID: bus 0x18 vendor 0x0 product 0x0 version 0x0
Input device name: "ar1021 I2C Touchscreen"
Supported events:
  Event type 0 (EV_SYN)
  Event type 1 (EV_KEY)
    Event code 330 (BTN_TOUCH)
  Event type 3 (EV_ABS)
    Event code 0 (ABS_X)
      Value   2494
      Min        0
      Max     4095
    Event code 1 (ABS_Y)
      Value    499
      Min        0
      Max     4095
Properties:
Testing ... (interrupt to exit)
Event: time 1446835189.051908, type 3 (EV_ABS), code 0 (ABS_X), value 2791
Event: time 1446835189.051908, type 3 (EV_ABS), code 1 (ABS_Y), value 1918
Event: time 1446835189.051908, -------------- SYN_REPORT ------------
Event: time 1446835189.063241, type 1 (EV_KEY), code 330 (BTN_TOUCH), value 1
Event: time 1446835189.063241, -------------- SYN_REPORT ------------
Event: time 1446835189.069177, type 3 (EV_ABS), code 1 (ABS_Y), value 1919
Event: time 1446835189.069177, -------------- SYN_REPORT ------------
Event: time 1446835189.087452, type 3 (EV_ABS), code 0 (ABS_X), value 2792
Event: time 1446835189.087452, type 3 (EV_ABS), code 1 (ABS_Y), value 1917
…
```

The program will continue to listen for and display touch events until stopped with `Ctrl+C`.

4.15 QSPI

COM boards	QSPI in u-boot	QSPI in Linux
iMX6 SoloX COM	Yes	Yes
iMX6 Quad COM	Not Supported by CPU	Not Supported by CPU
iMX6 DualLite COM	Not Supported by CPU	Not Supported by CPU
iMX6 UltraLite COM	Not Supported by CPU	Not Supported by CPU
iMX7 Dual COM	Yes	Yes
iMX7 Dual uCOM	Yes	Yes
iMX7ULP uCOM	Yes	No driver available since QSPI should only be used by Cortex-M4
iMX8M Quad COM	Not Supported by COM board	Not Supported by COM board
iMX8M Mini uCOM	Not Supported by COM board	Not Supported by COM board
iMX8M Nano uCOM	Not Supported by COM board	Not Supported by COM board

The iMX6 SoloX COM board has two 8Mbyte QSPI Flash memories (originally the board has two 32Mbyte memories).

The iMX7 Dual COM board has one 32MByte QSPI Flash memory.

The iMX7 Dual uCOM board does not have a QSPI memory on the uCOM board, but has one 32MByte QSPI Flash memory on the uCOM Adapter Board.

4.15.1 U-boot

The u-boot has the `sf` command to handle SPI flash:

```
=> sf
sf - SPI flash sub-system

Usage:
sf probe [[bus:]cs] [hz] [mode] - init flash device on given SPI
                                  bus and chip select
sf read addr offset len         - read `len' bytes starting at
                                  `offset' to memory at `addr'
sf write addr offset len        - write `len' bytes from memory
                                  at `addr' to flash at `offset'
sf erase offset [+]len          - erase `len' bytes from `offset'
                                  `+len' round up `len' to block
                                  size
sf update addr offset len       - erase and write `len' bytes from
                                  memory at `addr' to flash at
                                 `offset'
```

The two memories can be seen with the `probe` command:

```
=> sf probe 0:0
SF: Detected N25Q256 with page size 256 Bytes, erase size 4 KiB,
total 32 MiB
```

```
=> sf probe 1:0
SF: Detected N25Q256 with page size 256 Bytes, erase size 4 KiB,
total 32 MiB
```

4.15.2 Linux

To see the QSPI flash (output from iMX6 SoloX COM board):

```
# cat /proc/mtd
dev:    size   erasesize  name
mtd0: 02000000 00010000 "21e4000.qspi"
mtd1: 02000000 00010000 "21e4000.qspi"
```

The table shows mtd0 and mtd1 but the corresponding block devices are mtdblock0 and mtdblock1. To test the flash start by creating a test file with 16Kbyte random data:

```
# dd if=/dev/urandom of=write.dat bs=1024 count=16
```

Write the random data to the block device:

```
# time dd if=write.dat of=/dev/mtdblock0
32+0 records in
32+0 records out

real    0m0.074s
user    0m0.000s
sys     0m0.030s
```

Read back the data:

```
# time dd if=/dev/mtdblock0 of=read.dat bs=1024 count=16
16+0 records in
16+0 records out

real    0m0.006s
user    0m0.000s
sys     0m0.000s
```

Compare the two files to make sure that nothing was lost:

```
# diff read.dat write.dat
```

If the files are identical the diff command will not output anything. If there is a difference then it will look like this:

```
# diff read.dat write.dat
Files read.dat and write.dat differ
```

4.16 Analog Output - PWM

COM boards	Channels	Example PWM	pwmchipX
iMX6 SoloX COM	8	pwm1	pwmchip0
iMX6 Quad COM	4	pwm1	pwmchip0
iMX6 DualLite COM	4	pwm1	pwmchip0
iMX6 UltraLite COM	8	pwm1	pwmchip0
iMX7 Dual COM	4	pwm3	pwmchip1
iMX7 Dual uCOM	4	pwm3	pwmchip1
iMX7ULP uCOM	Not available		
iMX8M Quad COM	4	pwm1	pwmchip0
iMX8M Mini uCOM	4	pwm1	pwmchip0
iMX8M Nano uCOM	4	pwm1	pwmchip0

To use the PWM channels they must first be configured and enabled in the device tree. The scope of how to do that is beyond this document.

The table above list one example pwm that is enabled on all boards (backlight for LVDS). All COM boards have this on pin P138/284 of the MXM3 connector and the signal can be observed on resistor R120 or R259 (explanation below).

4.16.1 U-boot

Not applicable.

4.16.2 Linux

To see which PWM channels have been enabled in the device tree:

```
# ls /sys/class/pwm/
pwmchip0  pwmchip1
```

In this case (iMX6 DualLite) two channels are enabled. On an iMX6 SoloX board with all channels enabled it would look like this:

```
# ls /sys/class/pwm/
pwmchip0  pwmchip1  pwmchip2  pwmchip3
pwmchip4  pwmchip5  pwmchip6  pwmchip7
```

Each PWM channel has its own pwmchipX folder with a couple of interesting files:

```
# ls /sys/class/pwm/pwmchip0/
device     export     npwm       power
subsystem  uevent     unexport
```

The export/unexport files work in the same way as for gpio. To use a pwm channel it must first be exported:

```
# echo 0 > /sys/class/pwm/pwmchip1/export
```

It will then appear as a pwm0 node:

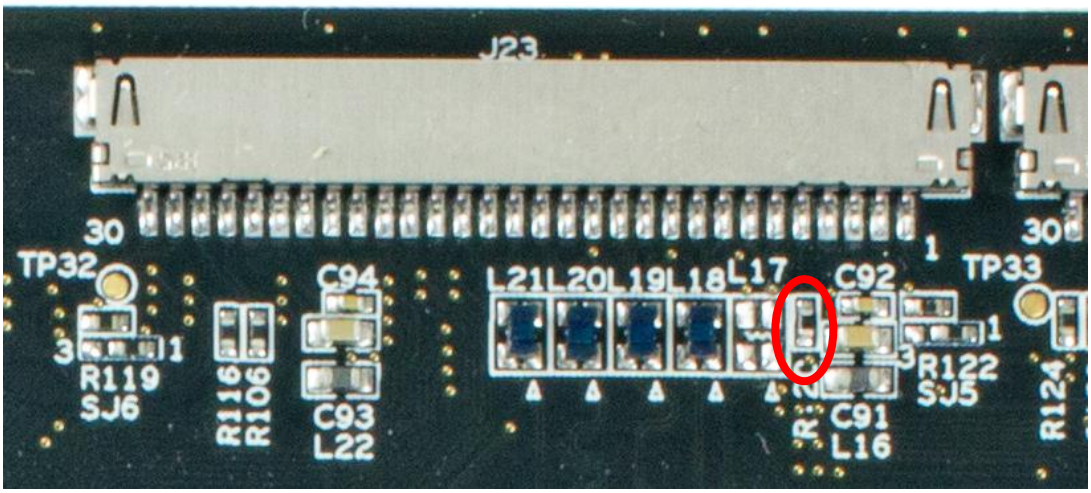
```
# ls /sys/class/pwm/pwmchip1/pwm0
capture     duty_cycle  enable      period
polarity    power       uevent
```

To set the PWM frequency to 100 kHz (100kHz equals 10000 ns period time) and set the duty cycle to 70% (0.7 * period time equals 7000):

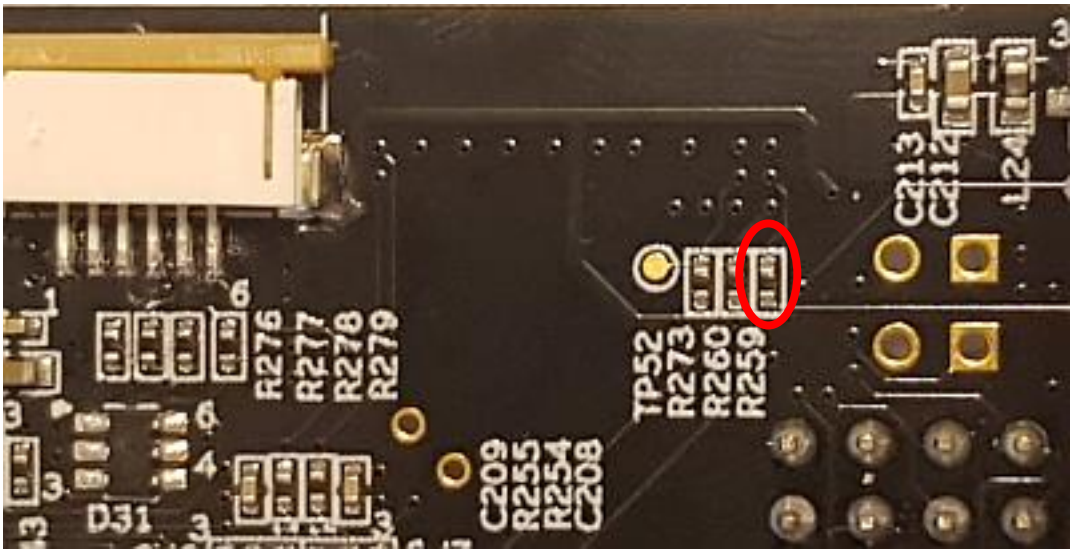
```
# cd /sys/class/pwm/pwmchip1/pwm0
# echo 10000 > period
# echo 7000 > duty_cycle
# echo 1 > enable
```

This signal can then be observed either with an oscilloscope or by measuring with a multimeter that will typically measure an average voltage. In this case the average should be ca 2.3V which is 70% of 3.3V.

The table at the start of this section shows one example pwm that is enabled on all boards. All COM boards have this on pin P138/284 of the MXM3 connector and the signal can be observed on resistor R120 on the COM Carrier Board



On the COM Carrier Board V2 it is instead resistor R259 which can be found on the back side:



4.17 Analog Input - ADC

COM boards	External Signals of the ADC	Precision	Reference Voltage and Max Input Voltage**
iMX6 SoloX COM	8 signals divided into 2 channels each with 4 inputs	12 bit	3.3V
iMX6 Quad COM	Not Supported by CPU		
iMX6 DualLite COM	Not Supported by CPU		
iMX6 UltraLite COM	10 * signals divided into 1 channel with 10 inputs	12 bit	3.3V
iMX7 Dual COM	8 signals divided into 2 channels each with 4 inputs	12 bit	1.8V
iMX7 Dual uCOM	4 signals divided into 1 channel with 4 inputs	12 bit	1.8V
iMX7ULP uCOM	ADC not available in u-boot / Linux. Only to be used by Cortex-M4		
iMX8M Quad COM	Not Supported by CPU		
iMX8M Mini uCOM	Not Supported by CPU		
iMX8M Nano uCOM	Not Supported by CPU		

*) To use the ADC channels on iMX6 UltraLite they must first be configured and enabled in the device tree. The scope of how to do that is beyond this document.

**) Note that input voltages above max will damage the CPU.

4.17.1 U-boot

Not applicable.

4.17.2 Linux

Each ADC channel will be represented by a iio:device node in the file system:

```
# ls /sys/bus/iio/devices/
iio:device0  iio:device1

# ls /sys/bus/iio/devices/iio\:device0/
buffer                          name
dev                             of_node
in_conversion_mode              power
in_voltage0_raw                 sampling_frequency_available
in_voltage1_raw                 scan_elements
in_voltage2_raw                 subsystem
in_voltage3_raw                 trigger
in_voltage_sampling_frequency   uevent
in_voltage_scale

# ls /sys/bus/iio/devices/iio\:device1/
buffer                          name
dev                             of_node
in_conversion_mode              power
```

```
in_voltage0_raw                 sampling_frequency_available
in_voltage1_raw                 scan_elements
in_voltage2_raw                 subsystem
in_voltage3_raw                 trigger
in_voltage_sampling_frequency   uevent
in_voltage_scale
```

In this case (iMX6 SoloX) the 8 signals are divided into two channels (iio:device0 and iio:device1) of 4 inputs each (in_voltage0_raw, in_voltage1_raw, in_voltage2_raw and in_voltage3_raw)

To read the raw value (range is 0 - 4095 for 12 bit precision):

```
# cat /sys/bus/iio/devices/iio\:device0/in_voltage2_raw
302
```

To see the scale:

```
# cat /sys/bus/iio/devices/iio\:device0/in_voltage_scale
0.805664062
```

To get the actual value multiply the raw value and the scale:

302 * 0.805664062 = 243.3 mV

4.18 M.2 Key B Connector (COM Carrier Board V2 only)

The M.2 Key B connector on the COM Carrier Board V2 has three interfaces: USB, SATA and SIM card. The SATA interface testing is explained in section 4.5

This test uses a Sierra Wireless AirPrime EM7305 module (https://source.sierrawireless.com/devices/em-series/em7305/) and a SIM card. It should be possible to use any USB modem with the correct M.2 key and form factor but other modems may require specific Linux drivers, use different USB serial ports and maybe different commands.

4.18.1 U-boot

Not applicable.

4.18.2 Linux

Start by checking that the module is detected. There should be a Sierra Wireless device like this:

```
# lsusb
Bus 001 Device 004: ID 1199:68c0 Sierra Wireless, Inc.
Bus 001 Device 003: ID 0424:2740 Standard Microsystems Corp.
Bus 001 Device 002: ID 0424:2744 Standard Microsystems Corp.
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

Verify that the serial devices have been created:

```
# ls -l /dev/serial/by-id/
lrwxrwxrwx    1 root     root            13 Feb  8 10:05 usb-
Sierra_Wireless__Incorporated_EM7305-if00-port0 -> ../../ttyUSB0
lrwxrwxrwx    1 root     root            13 Feb  8 10:05 usb-
Sierra_Wireless__Incorporated_EM7305-if02-port0 -> ../../ttyUSB1
lrwxrwxrwx    1 root     root            13 Feb  8 10:05 usb-
Sierra_Wireless__Incorporated_EM7305-if03-port0 -> ../../ttyUSB2
```

If there are no Sierra usb devices then check that the kernel has been configured with support for the device. In this case the configuration option is CONFIG_USB_SERIAL_QUALCOMM:

```
# cp /proc/config.gz .
# gunzip config.gz
# grep CONFIG_USB_SERIAL_QUALCOMM config
CONFIG_USB_SERIAL_QUALCOMM=m
```

Start miniterm with the exit code parameter 27 to use the ESC key to terminate the program instead of the default Ctrl+]:

```
# miniterm.py --exit-char 27 /dev/ttyUSB2 115200
--- Miniterm on /dev/ttyUSB2  115200,8,N,1 ---
--- Quit: Ctrl+[ | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H
---
```

Type AT and then press Enter to see if the Sierra module responds at all. If there is no OK response then exit (ESC key) and try one of the other usb port.

```
AT
OK
```

If AT worked the type AT+CPINR and press Enter. If the SIM card is inserted and the interface is working then it should report a number of stats:

```
AT+CPINR
+CPINR: SIM PIN,3
+CPINR: SIM PUK,10
+CPINR: SIM PIN2,3
+CPINR: SIM PUK2,10
+CPINR: PH-FSIM PIN,255
...
+CPINR: PH-SP PUK,1
+CPINR: PH-CORP PUK,1
OK
```

If the SIM card is not inserted or the interface is not working then the command will return this response:

```
AT+CPINR
+CME ERROR: SIM not inserted
```

Exit the miniterm program with the ESC key

4.19 M.2 Key E Connector (COM Carrier Board V2 only)

The M.2 Key E connector on the COM Carrier Board V2 has three interfaces: SDIO, PCIe and UART with the goal of supporting wireless M.2 modules. The very basic use case of detecting PCIe is covered in section 4.9 .

COM boards	PCIe	SDIO	Bluetooth UART
iMX6 SoloX COM	imx6sxea-com-kit_v2-pcie.dtb	imx6sxea-com-kit_v2.dtb	/dev/ttymxc1
iMX6 Quad COM	imx6qea-com-kit_v2-pcie.dtb	imx6qea-com-kit_v2.dtb	/dev/ttymxc4
iMX6 DualLite COM	imx6dlea-com-kit_v2-pcie.dtb	imx6dlea-com-kit_v2.dtb	/dev/ttymxc4
iMX6 UltraLite COM	Not Supported by CPU	imx6ulea-com-kit_v2.dtb	/dev/ttymxc1
iMX7 Dual COM	imx7dea-com-kit_v2-pcie.dtb	imx7dea-com-kit_v2.dtb	/dev/ttymxc1
iMX7 Dual uCOM	imx7dlea-ucom-kit_v2-pcie.dtb	imx7dea-ucom-kit_v2.dtb	/dev/ttymxc1
iMX7ULP uCOM without 1LV	Not supported by CPU	imx7ulpea-ucom-kit_v2.dtb	/dev/ttyLP2
iMX7ULP uCOM with 1LV	Not supported by CPU	imx7ulpea-ucom-ptp-1lv.dtb	/dev/ttyLP2
iMX8M Quad COM	fsl-imx8mq-ea-com-kit_v2-pcie.dtb	fsl-imx8mq-ea-com-kit_v2.dts	/dev/ttymxc1
iMX8M Mini uCOM without 1MW	fsl-imx8mm-ea-ucom-kit_v2-pcie.dtb	fsl-imx8mm-ea-ucom-kit_v2.dtb	/dev/ttymxc0
iMX8M Mini uCOM with 1MW	Not supported	fsl-imx8mm-ea-ucom-kit_v2-1mw.dtb	/dev/ttymxc0
iMX8M Nano uCOM without 1MW	Not supported	fsl-imx8mn-ea-ucom-kit_v2.dtb	/dev/ttymxc0
iMX8M Nano uCOM with 1MW	Not supported	fsl-imx8mn-ea-ucom-kit_v2-1mw.dtb	/dev/ttymxc0

Note that the "fsl-" prefix has been removed from the file names starting with Linux 5.4.24.

Testing the PCIe interface requires a 1CX M.2 module.

Testing the SDIO interface requires either a 1DX, 1LV or 1MW M.2 module.

Similar testing can be done with the 1XA, 1YM and 1ZM modules but requires different setup. See the *Getting Started with M.2 modules and i.MX 6/7/8* document for details.

4.19.1 U-boot

There is no support for using the M.2 modules in the u-boot. However, the u-boot is used to configure if the SDIO or PCI interface (on the M.2 connector) shall be active. This is done by selecting which device tree file to use.

When using a module with PCIe interface (for example the 1CX M.2 module) run the following commands in the u-boot, substituting the filename based on the COM board being used:

```
=> setenv fdt_file imx6qea-com-kit_v2-pcie.dtb
=> saveenv
```

When using a module with SDIO interface (for example the 1DX M.2 module) run the following commands in the u-boot, substituting the filename based on the COM board being used:

```
=> setenv fdt_file imx6qea-com-kit_v2.dtb
=> saveenv
```

4.19.2 Linux - Wifi

Start by checking that the module has been detected (exact text depends on M.2 module connected):

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

Configure the SSID and password of your wifi network by editing /etc/wpa_supplicant.conf so that it looks like this (but with your wifi settings instead):

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

network={
        ssid="EA Guest"
        psk="My password"
}
```

Now connect to that network:

```
# ifconfig wlan0 up
IPv6: ADDRCONF(NETDEV_UP): wlan0: link is not ready

#wpa_supplicant -i wlan0 -D nl80211 -c /etc/wpa_supplicant.conf -B
Successfully initialized wpa_supplicant
rfkill: Cannot open RFKILL control device
rfkill: Cannot get wiphy information
```

After a couple of seconds the module should be connected to the network and then the following will be printed in the terminal:

```
IPv6: ADDRCONF(NETDEV_CHANGE): wlan0: link becomes ready
```

If that doesn't happen then check the settings in /etc/wpa_supplicant.conf, reboot and test again.

Request an IP number from the network:

```
# udhcpc -i wlan0
udhcpc: started, v1.27.2
udhcpc: sending discover
udhcpc: sending discover
udhcpc: sending select for 192.168.1.147
udhcpc: lease of 192.168.1.147 obtained, lease time 86400
/etc/udhcpc.d/50default: Adding DNS 192.168.1.1
```

Use any tool that would work on a wired network to test the wifi connection, e.g. ping and iperf3. See section 4.2

4.19.3 Linux - Bluetooth

Start by checking that the module has been detected:

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

Start by initializing the UART (pick the UART in the table above according to you COM board):

```
# hciattach /dev/ttymxc1 bcm43xx 3000000 flow -t 20
bcm43xx_init
Set Controller UART speed to 3000000 bit/s
Flash firmware /etc/firmware/BCM43012C0.1LV.hcd
Set Controller UART speed to 3000000 bit/s
Setting TTY to N_HCI line discipline
Device setup complete
```

The output from the hciattach command will show the module name (in the case above a 1LV M.2 module was used). If the M.2 module is not inserted or it cannot be communicated with then the hciattach command will timeout after 20 seconds.

Get some basic information:

```
# hciconfig -a

hci0:   Type: Primary  Bus: UART
        BD Address: 44:91:60:9A:7B:3D ACL MTU: 1021:8 SCO MTU:64:1
        DOWN
        RX bytes:708 acl:0 sco:0 events:38 errors:0
        TX bytes:446 acl:0 sco:0 commands:38 errors:0
        Features: 0xbf 0xfe 0xcf 0xfe 0xdb 0xff 0x7b 0x87
        Packet type: DM1 DM3 DM5 DH1 DH3 DH5 HV1 HV2 HV3
        Link policy: RSWITCH SNIFF
        Link mode: SLAVE ACCEPT
```

Open and initialize the hci device:

```
# hciconfig hci0 up
```

Enable page and inquiry scan:

```
# hciconfig hci0 piscan
```

Scan for other Bluetooth devices in range:

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