

Wireless Communication on iMX Developer's Kits

Embedded Artists AB

Davidshallsgatan 16
SE-211 45 Malmö
Sweden

<http://www.EmbeddedArtists.com>

Copyright 2016 © 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	Communication interfaces	7
3.1	PCIe	7
3.2	SDIO	8
3.3	XBee / UART	8
3.4	USB.....	8
4	Wi-Fi	9
4.1	Introduction	9
4.2	General requirements	9
4.2.1	Wireless support in the kernel	9
4.2.2	PCIe support in the kernel	9
4.2.3	Required tools	9
4.3	Module: Intel 7260 (PCIe).....	9
4.3.1	Kernel configuration	10
4.3.2	Firmware	10
4.4	Module: D-Link DWA-121 (USB).....	11
4.4.1	Kernel configuration	11
4.4.2	Firmware	11
4.5	Connect manually from console	12
4.6	Connect automatically during boot	14
5	Bluetooth	16
5.1	Introduction	16
5.2	General requirements	16
5.2.1	Bluetooth support in the kernel	16
5.2.2	Required tools	16
5.3	Module: Intel 7260 (PCIe).....	16
5.3.1	Firmware	16
5.4	Access / configure Bluetooth devices.....	17
5.5	Additional links.....	19
6	XBee (UART).....	20
6.1	Introduction	20
6.2	Module: XBee 802.15.4.....	20
6.2.1	Flow Control	23
6.2.2	Software Alternative – GitHub	24
6.2.3	Software Alternative – Python XBee 2.2.3.....	24
6.2.4	Software Alternative – Python	24

7 Cellular networks	26
7.1 Introduction	26
7.2 Module: U-Blox TOBY-L210.....	26
7.2.1 AT Commands	27
7.2.2 Network	27

1 Document Revision History

<i>Revision</i>	<i>Date</i>	<i>Description</i>
A	2016-12-06	First release

2 Introduction

This document describes different ways to add wireless functionality to an *iMX Developer's Kit*. The different wireless technologies and a selection of hardware modules implementing the technologies will be presented. There will also be a description of how to connect the hardware modules to the developer's kit as well as how to use the wireless technology from within *Linux*.

There are many different *iMX Developer's Kits* 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.

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 *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 Carrier Board Datasheet*
- *EACOM Board Specification*

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 Communication interfaces

The *iMX Developer's Kits* don't have any onboard module directly supporting a wireless technology. Instead a module can be connected to one of the many interface connectors available on the COM Carrier Board.

The picture below shows the connectors on the COM Carrier board that is typically used when connecting wireless technologies to the *iMX Developer's Kits*. It is of course also possible to use any of the expansion connectors, but this typically requires an adapter board to be used between the expansion connector and the module. Because of this the expansion connectors are not described in this document.

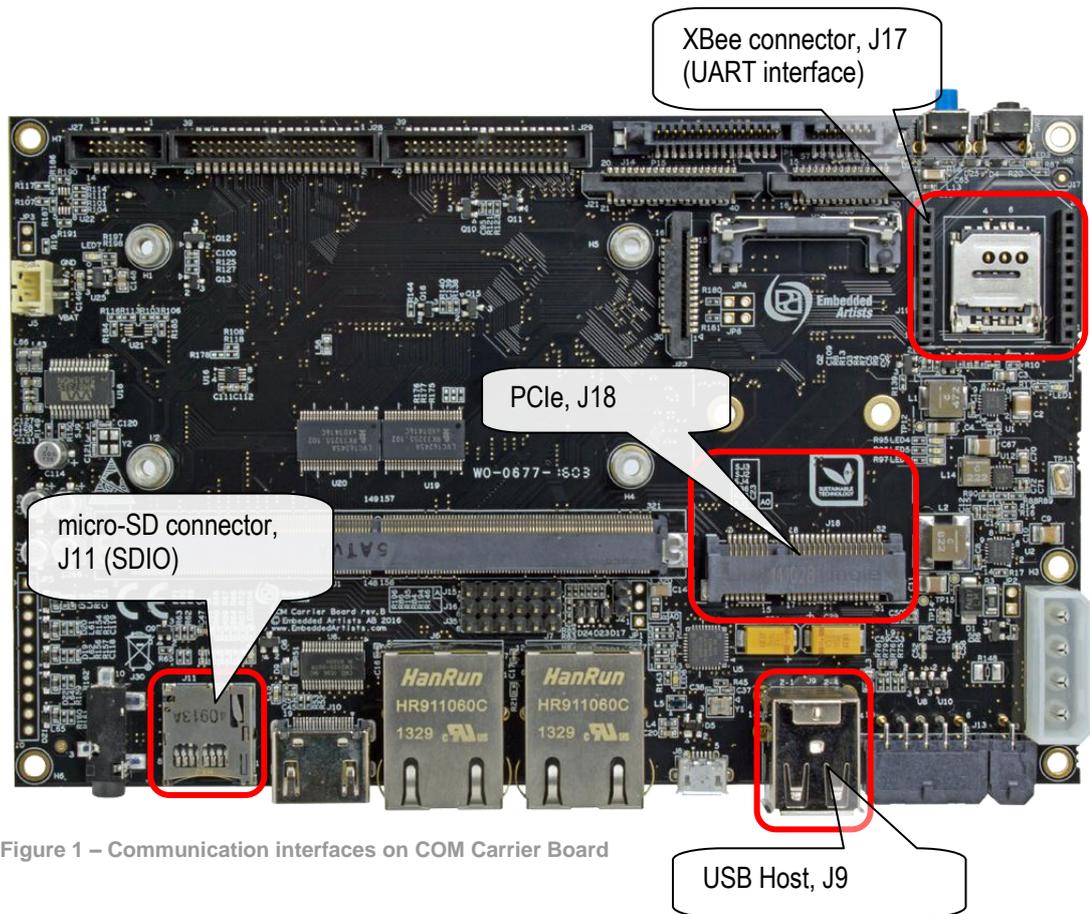


Figure 1 – Communication interfaces on COM Carrier Board

3.1 PCIe

PCI Express (Peripheral Component Interconnect Express), officially abbreviated as PCIe, is a high-speed serial computer expansion bus standard.

The COM Carrier board has one socket (J18) for PCIe mini. The PCIe socket includes one USB port and some wireless boards use the PCIe form factor but only use the USB part of the connector.

https://en.wikipedia.org/wiki/PCI_Express

NOTE: Not all *iMX COM Boards* support PCIe

3.2 SDIO

SDIO (Secure Digital Input Output) is an interface that extends the functionality of devices by using a standard SD card slot (J11) to give devices new capabilities. This could include GPS, Camera, Wi-Fi, FM Radio etc. This interface is not as commonly as, for example, USB.

<https://www.sdcard.org/developers/overview/sdio/index.html>

3.3 XBee / UART

The different i.MX processors have varying number of UARTs and one of them is always designated for the console. One of the free UARTs is connected to the XBee socket (J17) on the COM Carrier board.

Several modules with different wireless technologies exist that use the XBee form factor. There are, for example, modules supporting ZigBee, Wi-Fi and Bluetooth.

3.4 USB

USB, short for Universal Serial Bus, is an industry standard developed in the mid-1990s that defines the cables, connectors and communications protocols used in a bus for connection, communication, and power supply between computers and electronic devices.

The COM Carrier board has a dual USB Host type A connector (J9) with two USB2.0 Host interfaces. It is quite common to use USB on a wireless module. There are, for example, many Wi-Fi modules using the USB interface.

4 Wi-Fi

4.1 Introduction

Wi-Fi is a technology that allows devices to connect to a wireless network (WLAN), mainly using the 2.4 gigahertz UHF and 5 gigahertz SHF ISM radio bands. A WLAN is usually password protected, but may be open, which allows any device within its range to access the resources of the WLAN network.

A common way of adding Wi-Fi communication is to attach via USB but it is possible to use SDIO as well as the XBee port.

4.2 General requirements

4.2.1 Wireless support in the kernel

The Linux kernel must be configured to be able to use Wi-Fi. This configuration is already done for the COM boards.

- "Networking Support" → "RF switch subsystem support"
- "Networking Support" → "Wireless" → "cfg80211 wireless extensions compatibility"

This corresponds to the following configuration flags:

```
CONFIG_RFKILL=y
CONFIG_CFG80211_WEXT=y
```

4.2.2 PCIe support in the kernel

When using the Intel 7260 module PCIe must be supported in the Linux kernel. The PCI bus is enabled on the COM boards supporting it. To do the configuration for other hardware:

- "Bus Support" → PCI
- "Bus Support" → "Message Signaled Interrupts..."
- "Bus Support" → "PCI host controller drivers" → "Freescale i.MX6 PCIe controller"

This corresponds to the following configuration flags:

```
CONFIG_PCI=y
CONFIG_PCI_MSI=y
CONFIG_PCI_IMX6=y
```

4.2.3 Required tools

The root file system must have the following tool(s) installed:

- `wpa_supplicant`
- `wpa_cli`

Both tools are added with `packagegroup-base-extended` and it in turn is included in all images in Yocto except for `core-image-minimal`.

4.3 Module: Intel 7260 (PCIe)

Product	Interface	Comment
Intel^(R) Dual Band Wireless AC 7260 Plus Bluetooth	PCIe	It is commonly used in laptops and is available at a low price



4.3.1 Kernel configuration

Using the Intel 7260 requires some configuration options in the Linux kernel compared to the default for iMX. These have been enabled for the COM boards already and are only listed as reference below.

- "Device Drivers" → "Generic Driver Options" → "Userspace firmware loading support"
- "Device Drivers" → "Network Device Support" → "Wireless LAN" → "Intel Wireless WiFi Next Gen AGN -"
- "Device Drivers" → "Network Device Support" → "Wireless LAN" → "Intel Wireless WiFi MVM Firmware Support"

This corresponds to the following configuration flags:

```
CONFIG_FW_LOADER=y
CONFIG_IWLWIFI=m
CONFIG_IWLDVM=m
CONFIG_IWLMVM=m
CONFIG_IWLWIFI_OPMODE_MODULAR=y
```

4.3.2 Firmware

The Intel 7260 card needs a firmware file to work. The following prints will appear when booting if the firmware is missing:

```
Intel(R) Wireless WiFi driver for Linux, in-tree:
Copyright(c) 2003- 2014 Intel Corporation
PCI: enabling device 0000:01:00.0 (0140 -> 0142)
iwlwifi 0000:01:00.0: Direct firmware load failed with error -2
iwlwifi 0000:01:00.0: Falling back to user helper
iwlwifi 0000:01:00.0: Direct firmware load failed with error -2
iwlwifi 0000:01:00.0: Falling back to user helper
iwlwifi 0000:01:00.0: Direct firmware load failed with error -2
iwlwifi 0000:01:00.0: Falling back to user helper
iwlwifi 0000:01:00.0: request for firmware file 'iwlwifi-7260-7.ucode' failed.
iwlwifi 0000:01:00.0: no suitable firmware found!
```

The firmware can be downloaded from [Intel](http://intel.com). Make sure to pick the correct version for the Linux kernel. At the time this document was written that was `iwlwifi-7260-ucode-25.30.13.0.tgz`. Transfer the file to the target (for example with a USB memory stick) and then unpack it:

```
# tar -xf iwlwifi-7260-ucode-25.30.13.0.tgz
# cd iwlwifi-7260-ucode-25.30.13.0
# cp iwlwifi-7260-13.ucode /lib/firmware/
```

Reboot to load the new drivers. The error message above will be replaced with:

```
Intel(R) Wireless WiFi driver for Linux, in-tree:
Copyright(c) 2003- 2014 Intel Corporation
PCI: enabling device 0000:01:00.0 (0140 -> 0142)
iwlwifi 0000:01:00.0: loaded firmware version 25.30.13.0 op_mode
iwlvmv
iwlwifi 0000:01:00.0: Detected Intel(R) Dual Band Wireless AC
7260, REV=0x144
```

Go to section 4.5 or section 4.6 to continue with creating a connection to a wireless network.

4.4 Module: D-Link DWA-121 (USB)

Product	Interface	Comment
Wireless N 150 Pico USB Adapter 	USB	This module is using a chipset from Realtek called RTL8188CUS.

The DWA-121 module from D-Link is using a RTL8188CUS/RTL8892CUS compatible chipset from Realtek. It should be possible to use the instructions in this section also for other modules using the same chipset.

4.4.1 Kernel configuration

Using the Realtek RTL8188CUS/RTL8192CUS chipset requires some configuration options in the Linux kernel compared to the default for iMX. These have been enabled for the COM boards already and are only listed as reference below.

- “Device Drivers” → “Network device support” → “Wireless LAN” → “Realtek rtlwifi family of devices”
- “Device Drivers” → “Network device support” → “Wireless LAN” → “Realtek rtlwifi family of devices” → “Realtek RTL8192CU/RTL8188CU USB Wireless Network Adapter”

This corresponds to the following configuration flags:

```
CONFIG_RTL_CARDS=y
CONFIG_RTLWIFI=y
CONFIG_RTLWIFI_USB=y
CONFIG_RTL8192C_COMMON=y
CONFIG_RTL8192CU=y
```

4.4.2 Firmware

The DWA-121 module needs a firmware file to work. If the file is missing the output below can be seen in the console when inserting a module into the USB host port.

```

rtl8192cu: Chip version 0x10
rtl8192cu: MAC address: 54:b8:0a:01:23:87
rtl8192cu: Board Type 0
rtl_usb: rx_max_size 15360, rx_urb_num 8, in_ep 1
rtl8192cu: Loading firmware rtlwifi/rtl8192cufw_TMSC.bin
usb 1-1.3: Direct firmware load failed with error -2
usb 1-1.3: Falling back to user helper
usb 1-1.3: Direct firmware load failed with error -2
usb 1-1.3: Falling back to user helper
rtlwifi: Loading alternative firmware rtlwifi/rtl8192cufw.bin
rtlwifi: Firmware rtlwifi/rtl8192cufw_TMSC.bin not available

```

The firmware can be downloaded from several sources on the Internet, for example:

<http://git.kernel.org/cgit/linux/kernel/git/firmware/linux-firmware.git/tree/rtlwifi>

The firmware file should be put in the directory `/lib/firmware/rtlwifi`. Below are instructions that show how the file is downloaded using `wget` (meaning the target must have an Internet connection).

```

# cd /lib/firmware
# mkdir rtlwifi
# cd rtlwifi
# wget http://git.kernel.org/cgit/linux/kernel/git/firmware/linux-firmware.git/plain/rtlwifi/rtl8192cufw_TMSC.bin

```

When the firmware file has been downloaded you can insert the D-Link USB module into the USB host port and you should see output like below in the console.

```

rtl8192cu: Chip version 0x10
rtl8192cu: MAC address: 54:b8:0a:01:23:87
rtl8192cu: Board Type 0
rtl_usb: rx_max_size 15360, rx_urb_num 8, in_ep 1
rtl8192cu: Loading firmware rtlwifi/rtl8192cufw_TMSC.bin
rtlwifi: wireless switch is on

```

Go to section 4.5 or section 4.6 to continue with creating a connection to a wireless network.

4.5 Connect manually from console

Bring up the `wlan0` interface and then start the `wpa_supplicant` daemon in the background:

```

# ip link set wlan0 up
# wpa_supplicant -B -i wlan0 -c /etc/wpa_supplicant.conf

```

Sometimes the `ip link` command will respond with an "Operation not possible due to RF-kill" error message. To fix run the commands below.

```

# rfkill list
0: hci0: Bluetooth
    Soft blocked: yes
    Hard blocked: no
1: phy0: wlan
    Soft blocked: yes           ← Blocked
    Hard blocked: no

# rfkill unblock wifi

```

```
# rfkill list
0: hci0: Bluetooth
    Soft blocked: yes
    Hard blocked: no
1: phy0: wlan
    Soft blocked: no           ← No longer blocked
    Hard blocked: no
```

It is now time to run the interactive `wpa_cli` tool to scan for, select and then connect to a network.

```
# wpa_cli
```

Scan for networks (press ENTER after the last line below to get back to the prompt):

```
> scan
OK
<3>CTRL-EVENT-SCAN-STARTED
<3>CTRL-EVENT-SCAN-RESULTS
```

To list found networks:

```
> scan_result
bssid / frequency / signal level / flags / ssid
00:1e:00:00:ca:89 2412 -76 [WPA2-PSK-CCMP+TKIP][ESS] EA Guest
00:fa:00:00:49:ff 5500 -86 [WPA2-EAP-CCMP][ESS] BOB
00:c7:00:00:b2:80 2462 -86 [WPA2-EAP-CCMP][ESS] ALICE
00:fa:00:00:49:f1 2412 -76 [ESS] Company_Guest
00:c7:00:00:b3:90 2412 -93 [WPA2-EAP-CCMP][ESS] Company_Inc
```

The list shows that the “EA Guest” network uses pre-shared keys (PSK) as authentication. It is a relatively low security alternative but it is simple to use. The following example will connect to that network using the password “welcome”. For examples using other authentication methods see https://wiki.netbsd.org/tutorials/how_to_use_wpa_supplicant/.

To connect to the “EA Guest” network:

```
> add_network
0
> set_network 0 ssid "EA Guest"
OK
> set_network 0 psk "welcome"
OK
> enable network 0
...
<3>CTRL-EVENT-CONNECTED...
> status
bssid=00:1e:00:00:ca:89
ssid=EA Guest
id=0
mode=station
pairwise_cipher=CCMP
group_cipher=TKIP
key_mgmt=WPA2-PSK
wpa_state=COMPLETED
address=d8:00:93:00:1e:00
...
```

```
> save_config          ← Very important to save before quitting
  OK
> quit
```

Note: If there is more than one network (added before or added automatically) the `add_network` call will return another number, for example 1. Use that in the `set_network` and `enable_network` calls. If there is another enabled network then the `enable_network` call won't make any difference. Use the `select_network X` command to select the wanted network and disable all the others. An alternative is to use the `remove_network X` command on each of the existing networks before starting.

Some useful commands:

```
> list_networks
network id / ssid / bssid / flags
0          EA Guest          any          [CURRENT]

> remove_network 1
  OK

> select_network 0
  OK
```

Now that there is a connection to a network all that is left to do is to get an IP address:

```
# udhcpc -i wlan0
udhcpc (v1.22.1) started
Sending discover...
Sending select for 192.168.0.6...
Lease of 192.168.0.6 obtained, lease time 3600
/etc/udhcpc.d/50default: Adding DNS 192.168.0.1
/etc/udhcpc.d/50default: Adding DNS 8.8.8.8
```

The changes will last until next reboot at which time it must all be repeated.

4.6 Connect automatically during boot

It is possible to configure the system to automatically connect to a network when booting instead of having to connect manually each time. To do this update the `/etc/wpa_supplicant.conf` file to look like this:

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

network={
    key_mgmt=NONE
    disabled=1
}
network={
    ssid="Ea Guest"
    psk="welcome"
}
```

The first network is disabled to prevent connections to open networks. The second network is the one found with `wpa_cli` in the manual configuration steps above.

Update `/etc/network/interfaces` to contain the following (it will have more lines but the lines below must appear somewhere in the file):

```
# Wireless interfaces
auto wlan0
iface wlan0 inet dhcp
    wireless_mode managed
    wireless_essid any
    wpa-driver wext
    wpa-conf /etc/wpa_supplicant.conf
```

5 Bluetooth

5.1 Introduction

Bluetooth is a wireless technology mainly using the 2.4 to 2.485 gigahertz ISM radio bands for short range communication. It is commonly supported by mobile / handheld devices due to its low-power consumption.

5.2 General requirements

5.2.1 Bluetooth support in the kernel

There are many Bluetooth related configurations for the kernel but all the required ones have already been set in the default configuration for iMX. The Linux kernel provided by Embedded Artists is configured with Bluetooth support.

5.2.2 Required tools

The root file system must have the following tool(s) installed:

- `bluetoothctl`
- `bluetoothd`

Both tools are added with `packagegroup-base-extended` and it in turn is included in all images in Yocto except for `core-image-minimal`.

5.3 Module: Intel 7260 (PCIe)

Product	Interface	Comment
Intel[®] Dual Band Wireless AC 7260 Plus Bluetooth	PCIe	It is commonly used in laptops and is available at a low price

Using the Intel 7260 requires some configuration options in the Linux kernel compared to the default for iMX. These have been enabled for the COM boards already.

- “Device Drivers” → “Generic Driver Options” → “Userspace firmware loading support”

This corresponds to the following configuration flags:

```
CONFIG_FW_LOADER=y
```

5.3.1 Firmware

The Intel 7260 card needs a firmware file to work. The following output will appear when booting if the firmware is missing:

```
Bluetooth: hci0 failed to open Intel firmware file: intel/ibt-hw-37.7.10-fw-1.80.1.2d.d.bseq(-2)
bluetooth hci0: Direct firmware load failed with error -2
bluetooth hci0: Falling back to user helper
Bluetooth: hci0 failed to open default Intel fw file: intel/ibt-hw-37.7.bseq
```

The firmware can be downloaded from [Intel](#). At the time this document was written there was no direct link to the firmware from the Intel page. The file is available in the `linux-firmware` git and can be directly downloaded to the target with the following commands:

```
# mkdir -p /lib/firmware/intel
# cd /lib/firmware/intel
# wget http://git.kernel.org/cgit/linux/kernel/git/firmware/linux-firmware.git/plain/intel/ibt-hw-37.7.10-fw-1.80.1.2d.d.bseq
```

Reboot to load the new drivers. The error message above will be replaced with:

```
Bluetooth: hci0: Intel Bluetooth firmware file: intel/ibt-hw-37.7.10-fw-1.80.1.2d.d.bseq
```

5.4 Access / configure Bluetooth devices

Check if the device is up:

```
# hciconfig -a hci0
hci0:      Type: USB
          BD Address: 00:00:00:00:00:00 ACL MTU: 0:0 SCO MTU: 0:0
          DOWN
          RX bytes:0 acl:0 sco:0 events:0 errors:0
          TX bytes:0 acl:0 sco:0 commands:0 errors:
```

If it is down then bring it up with:

```
# hciconfig hci0 up
```

Then test the status again:

```
# hciconfig -a hci0
hci0:      Type: BR/EDR  Bus: USB
          BD Address: D8:FC:93:E4:1E:A6 ACL MTU: 1021:5 SCO MTU:
96:6
          UP RUNNING PSCAN
          RX bytes:1729 acl:0 sco:0 events:157 errors:0
          TX bytes:21346 acl:0 sco:0 commands:156 errors:0
          Features: 0xff 0xfe 0x0f 0xfe 0xdb 0xff 0x7b 0x87
          Packet type: DM1 DM3 DM5 DH1 DH3 DH5 HV1 HV2 HV3
          Link policy: RSWITCH HOLD SNIFF
          Link mode: SLAVE ACCEPT
          Name: 'BlueZ 5.25'
          Class: 0x200000
          Service Classes: Audio
          Device Class: Miscellaneous,
          HCI Version: 4.0 (0x6) Revision: 0xe00
          LMP Version: 4.0 (0x6) Subversion: 0xe00
          Manufacturer: Intel Corp. (2)
```

Start the Bluetooth daemon:

```
# /usr/lib/bluez5/bluetooth/bluetoothd &
```

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
```

5.5 Additional links

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

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

6 XBee (UART)

6.1 Introduction

XBee is a registered trademark of Digi International and it is really a brand name of a form factor more than a wireless technology. A separate chapter is dedicated to XBee since there is an XBee compatible connector on the COM Carrier board and the use of these modules requires special instructions. It is the UART interface that is used to communicate with XBee modules.

Several different wireless technologies may be supported by modules using the XBee form factor.

6.2 Module: XBee 802.15.4

Product	Interface	Comment
XBee 802.15.4 module from Digi International 	UART	

This description is for the XBee 802.15.4 module from Digi International. However the instructions and commands may be applicable to other modules as well.

The XBee module is connected to different pins depending on COM Board as shown in the table below. Note that the number in parentheses is the number to use when using `/sys/class/gpio/export` as described further down.

	SoloX	Quad	UltraLite
UART	/dev/ttyMC1	/dev/ttyMC4	/dev/ttyMC1
RF_CD	1_06 (6)	6_02 (162)	1_19 (19)
RF_DTR	1_07 (7)	6_03 (163)	1_18 (18)
RF_ON	N/A	3_23 (87)	4_16 (112)
RF_RST	1_16 (16)	2_19 (51)	4_23 (119)

Note that the RF_RST signal is available on Carrier boards rev B and later.

The XBee module has two main operating modes: Application Transparent (AT) and Application Programming Interface (API). AT is the default mode and in this mode the XBee module works as a point-to-multipoint serial port, meaning that everything sent from one module is received by all other modules on the same "network". The network is identified with a PAN ID and by default that is 3332. Only modules with the same PAN ID (and within range) will receive data.

The module is easily configured either directly with a terminal program (connected at 9600 baud, 8bits, 1 stop bit and no parity), or using the XTCU software provided by Digi International.

All configuration options are described in the product documentation (http://ftp1.digi.com/support/documentation/90000982_S.pdf), but here are some of the more important ones:

AT Command	Description	Default Value
ATNI	Node Identifier	
ATID	PAN ID	3332
ATAP	API Enable Command	0 – API Disabled
ATD6	DIO6 Configuration	0 – Disabled
ATD7	DIO7 Configuration	1 – CTS Flow Control
ATDH	Destination Address High	0
ATDL	Destination Address Low	0
ATSH	Serial Number High	13A200
ATSL	Serial Number Low	Module specific
ATMY	16-bit Source Address	0
ATBD	Interface Data Rate	3 – 9600 baud

Apart from configuration of the module there is one very important thing that must be done before it can be used and that is resetting. The RF_RST input must be drawn low, held low for at least 200ns, and then drawn high. If this is not done the module might be unresponsive.

To handle the reset sequence in Linux:

```
$ echo 163 > /sys/class/gpio/export
$ echo out > /sys/class/gpio/gpio163/direction
$ echo 0 > /sys/class/gpio/gpio163/value
$ echo 1 > /sys/class/gpio/gpio163/value
```

The number 163 is derived from the formula $num = (port - 1) * 32 + pin$. So GPIO6_03 will give $num = (6 - 1) * 32 + 3 = 163$.

An easy way to use the serial port:

```
$ stty -F /dev/ttymx4 9600 raw -echo
$ cat /dev/ttymx4 &
$ echo bob > /dev/ttymx4
```

All other modules in the network with the same PAN ID (default 3332) will receive the string "bob".

Another way is to use a terminal program like `microcom`:

```
$ microcom -s 9600 /dev/ttymx4
```

To change the PAN ID in `microcom` first enter command mode by entering `+++` and wait for an OK to appear. In command mode it is possible to enter AT commands, for example, `ATID` to view or set configuration options. The options are set in RAM and will be valid until next time the module is reset.

To see the current value of an option:

```
ATID
alice

ATIDBob
OK

ATID
Bob
```

To change network ID from the default one to 'EAEA':

```
ATIDEAEA
OK
```

Command mode can be exited either by being idle for a couple of seconds (how many is configurable) or by entering the `ATCN` command.

Changing the PAN ID from the default will not prevent anyone from joining the network. An alternative is to pair the module with another one so that traffic only flows between the paired modules.

Pairing is done by setting module A's destination address (DH + DL) to B's MAC address and the other way around so if the modules look like this:

	SH	SL
A	101010	20202020
B	303030	40404040

Then the destinations should be configured like this:

	DH	DL
A	303030	40404040
B	101010	20202020

The commands to do this on module **A** are:

```
ATDH303030
OK

ATDL40404040
OK
```

After making the changes only A and B communicates. To restore the defaults set the destinations back to 0 (both DH and DL) on both A and B.

By default, the Xbee modules are operating in Transparent Mode. In this mode the modules act as a point-to-multipoint serial line replacement. That which is sent into one module is transmitted and received on all other modules.

The API mode was mentioned in the beginning of this section, but there are currently no examples for it. The reason for this is that the API mode is a bit more complex and it requires frame handling and multiple processes.

6.2.1 Flow Control

The examples above use no flow control which will be sufficient for low data rates but when the traffic increases flow control will prevent data loss.

Note that the RTS/CTS flow control mode is not working on Carrier boards rev A. The issue has been fixed on rev B boards.

To enable RTS/CTS flow control and send the message "bob" to all listening devices:

```
$ stty -F /dev/ttymx4 9600 raw -echo crtscts
$ cat /dev/ttymx4 &
$ echo bob > /dev/ttymx4
```

If RTS/CTS is unsupported then the stty command will respond with an error like this:

```
$ stty -F /dev/ttymx4 9600 raw -echo crtscts
stty: /dev/ttymx4: cannot perform all requested operations
```

To check if RTS/CTS has been enabled or not:

```
$ stty -a -F /dev/ttymx4
speed 9600 baud;stty: /dev/ttymx4
...
-parenb -parodd cs8 hupcl -cstopb cread clocal crtscts
-ignbrk -brkint -ignpar -parmrk -inpck -istrip -inlcr -igncr
...
```

If there is a minus sign before the `crtscts` string then RTS/CTS is disabled.

RTS/CTS on the iMX6 Ultra Lite

On the UltraLite board a separate dtb file must be loaded to utilize the RTS/CTS flow control. The reason is that the RTS/CTS signals share pinning with the CAN2 interface so only one can be selected at a time. The default dtb file enables CAN2 and disables UART1 RTS/CTS.

Run the following commands in the u-boot to enable RTS/CTS:

```
=> setenv fdt_file imx6ulea-com-kit-rf.dtb
=> saveenv
```

To disable RTS/CTS and restore the CAN2 interface:

```
=> setenv fdt_file imx6ulea-com-kit.dtb
=> saveenv
```

6.2.2 Software Alternative – GitHub

The official code repository from Digi International is their Ansi C library on github: https://github.com/digidotcom/xbee_ansi_library. It is possible to compile it using the toolchain (INSERT LINK TO User's Manual) but out of all examples only the xbee_terminal application worked and it uses the AT commands.

Note that the module still needs to be reset before executing the application.

This is roughly the commands needed to compile the examples in Ubuntu:

```
# source /opt/poky/1.7/environment-setup-cortexa9hf-vfp-neon-poky-
linux-gnueabi
# git clone https://github.com/digidotcom/xbee_ansi_library.git
# cd xbee_ansi_library/samples/posix
# pico Makefile
    replace the "COMPILE = gcc ..." with "COMPILE = $(CC) ..."
# make all
```

6.2.3 Software Alternative – Python XBee 2.2.3

There is a python library (<https://pypi.python.org/pypi/XBee>) publicly available but it has some specific requirements to be installed and that is not available on the imx6 boards at this time.

6.2.4 Software Alternative – Python

It is possible to write your own python script to reset and control the XBee module. To do this the target file system must have at least the python-pyserial module installed. This procedure is described in the Working with Yocto document (INSERT LINK). Add the package like this:

```
IMAGE_INSTALL_append = " \
    python-pyserial \"
```

To enable gpio 6_03 and do a reset of the XBee module:

```
#!/usr/bin/python
import serial
import struct
from time import sleep
import sys

v = ['1', '6', '3']
with open('/sys/class/gpio/export', 'w') as f:
    f.write(struct.pack('c'*len(v), *v))

v = ['0', 'u', 't']
with open('/sys/class/gpio/gpio163/direction', 'w') as f:
    f.write(struct.pack('c'*len(v), *v))
```

```
with open('/sys/class/gpio/gpio163/value', 'r+') as f:
    f.write(struct.pack('c', '0'))

sleep(0.001)

with open('/sys/class/gpio/gpio163/value', 'r+') as f:
    f.write(struct.pack('c', '1'))

sleep(0.5)
```

To enter command mode and set node identifier:

```
#!/usr/bin/python
import serial
import struct
from time import sleep
import sys

ser = serial.Serial(port=comport, baudrate=9600, bytesize=8,
                    parity='N', stopbits=1)

ser.write('+++')
resp = ser.readline()
if resp.strip() == 'OK':
    ser.write('ATNIBob\r')
    resp = ser.readline()
    if resp.strip() == 'OK':
        print "Successfully set node id to 'Bob'"
```

7 Cellular networks

7.1 Introduction

Using cellular technology on embedded systems is getting more and more popular mostly because of its good coverage and high capacity.

7.2 Module: U-Blox TOBY-L210

Product	Interface	Comment
 <p>U-Blox TOBY-L210</p>	<p>USB</p> <p>The module has mini PCIe form factor but only the USB pins are in use.</p>	<p>LTE/HSPA+/GSM module for Europe/Asia</p>

Before starting:

- Make sure that you have a suitable external antenna for the network that will be used
- Make sure that the SIM card is valid, has no PIN code and has not expired (i.e. no limit on data usage and/or SMS)

Using the TOBY-L210 requires some configuration options in the Linux kernel compared to the default for iMX. These have been enabled for the COM boards already.

- Device Drivers → USB support → Support for Host-side USB → USB Modem (CDC ACM) support
- Device Drivers → Network Device Support → USB Network Adapters → Multi-purpose USB Networking Framework
- Device Drivers → Network Device Support → USB Network Adapters → CDC Ethernet support
- Device Drivers → Network Device Support → USB Network Adapters → Host for RNDIS and ActiveSync devices

This corresponds to the following configuration flags:

```
CONFIG_USB_ACM=m
CONFIG_USB_USBNET=m
CONFIG_USB_NET_CDCETHER=m
CONFIG_USB_NET_RNDIS_HOST=m
```

The module can be found with the `lsusb` command line tool:

```
# lsusb
Bus 001 Device 004: ID 1546:1146 U-Blox AG
...
```

Some information about the modem is printed during boot. Use the `dmesg` tool to find it:

```
# dmesg | grep acm
cdc_acm 1-1.1:1.2: This device cannot do calls on its own. It is
not a modem.
cdc_acm 1-1.1:1.2: ttyACM0: USB ACM device
usbcore: registered new interface driver cdc_acm
cdc_acm: USB Abstract Control Model driver for USB modems and ISDN
adapters
```

7.2.1 AT Commands

The module can be directly accessed with a terminal program (`minicom` is used as it is available in the default Linux build for all COM boards).

```
# microcom -s 115200 /dev/ttyACM0
AT
OK

AT+CMEE=2
OK

AT+CPIN?
+CPIN: READY
OK
```

All AT commands are described in the document *AT Command Examples for u-blox cellular modules*, found in the resource list for the module: <https://www.u-blox.com/en/product-resources/2451>.

As an example, this is what is needed to send an SMS to phone number 012-123456:

```
AT+CMGF=1
OK

AT+CMGS="012123456"    ← Press enter after phone number
> Hello from iMX6      ← Press Ctrl-Z to end message and send it

+CMGS: 43
```

Exit `microcom` with `Ctrl-X`

7.2.2 Network

The TOBY-L210 will be automatically detected during boot if the Linux kernel has RNDIS support (see above). To check the boot log:

```
# dmesg | grep rndis
rndis_host 1-1.1:1.0 usb0: register 'rndis_host' at usb-ci_hdrc.1-
1.1, RNDIS device, b2:f6:11:7b:09:17
usbcore: registered new interface driver rndis_host
```

Make sure that there is an `usb0` interface. It will not be active yet:

```
# ifconfig -a
usb0      Link encap:Ethernet  HWaddr 7E:61:55:F8:5E:35
          BROADCAST MULTICAST  MTU:1500  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:1000
RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
```

Check if the module is in router or bridged mode:

```
# microcom -s 115200 /dev/ttyACM0
AT+UBMCONF?
+UBMCONF: 1      ← 1 means router mode, 2 means bridged mode
OK
```

If the module is in bridged mode, use the instructions in the [Networking Modes AppNote](#) to switch to router mode.

Bring up the interface with the following command:

```
# udhcpc -i usb0
udhcpc (v1.22.1) started
Sending discover...
Sending select for 192.168.1.100...
Lease of 192.168.1.100 obtained, lease time 43200
/etc/udhcpc.d/50default: Adding DNS 192.168.1.1
```

The interface is now fully operational.

```
# route
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
default 192.168.1.1 0.0.0.0 UG 0 0 0 usb0
192.168.1.0 * 255.255.255.0 U 0 0 0 usb0

# ifconfig
...
usb0 Link encap:Ethernet HWaddr 0E:27:00:00:00:C6
inet addr:192.168.1.100 Bcast:192.168.1.255
Mask:255.255.255.0
inet6 addr: fe80::c967:bff:fa6a:4ba6/64 Scope:Link
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:12 errors:0 dropped:0 overruns:0 frame:0
TX packets:26 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:1096 (1.0 KiB) TX bytes:6742 (6.5 KiB)

# ping -I usb0 8.8.8.8
PING 8.8.8.8 (8.8.8.8): 56 data bytes
64 bytes from 8.8.8.8: seq=0 ttl=49 time=72.994 ms
64 bytes from 8.8.8.8: seq=1 ttl=49 time=80.355 ms
64 bytes from 8.8.8.8: seq=2 ttl=49 time=84.304 ms
```

However, the connection will be lost when rebooting. To make the interface available after a reboot, edit `/etc/network/interfaces` and add the following lines:

```
auto usb0
iface usb0 inet dhcp
```