Hands-on Linux & NFC

Instructions

Welcome to *Hands-on Linux & NFC Training*! This guide will show you how to start Quickly with NFC and with ARM-based I.MX6ULL systems-on-chip (SoC) running Embedded Linux.

PN7150

VisionSTK-6ULL-TR01

The hands-on Training consists of two parts:

- **Exercise 1.** The first step will be to prepare an SD card image the board can boot from. Flashing a raw system image will be demonstrated for both Linux and Windows-based host PCs. While waiting for the cards to flash, basics of embedded Linux systems design will be discussed. Once the SD cards are ready, we shall boot the VisionSOM6-ULL, log in through the serial console, and establish an Ethernet connection with the PC.

- **Exercise 2.** After a short introduction to NFC technology and NXP product offering, we will run sample applications to read common NFC tags, connect to smartphones, and exchange data with connected tags.

You will need about 2 hours to complete this hands on. Many important steps in designing a Linux system are not covered, such as porting u-boot bootloader and the Linux kernel to a new board. A ready to use image has been prepared for this training, but it is not intended to be used directly in production.

SoMLabs provides both documentation and software enablement for the VisionSOM modules on their product wiki: http://wiki.somlabs.com/index.php?title=VisionSOM-6ULL

Additional software and documentation for the i.MX6ULL SoC is available at *http://nxp.com/imx6ull*

Let's get started!

EXERCISE 1
Preparing an SD card to boot VisionSOM-6ULL

In order to suit a wide range of applications, **SoMLabs** released three versions of the **VisionSOM-6ULL** system-on-module (SoM), each with a different type of boot memory installed:

- *on-board eMMC Flash*
- *on-board NAND Flash*,
- *micro-SD card slot*.

Together with the VisionCB-STD base-board, VisionSOM-6ULL can be used as a stand-alone embedded computing platform.

For this Hands-on Linux Academy, the VisionCB-STD base-boards have been fitted with the micro-SD version of the SoM. This allows us to use a common SD card reader to prepare the boot memory.

Download the SD card image from the link below:

http://co.rru.pt/somlabs/images/

After you have extracted the image, connect the USB card reader to your PC or insert the card to the built-in card reader in your laptop.

Follow instructions in section 1.1 or 1.2, depending on the operating system you are using.

1.1. Preparing the SD card under Linux

In Linux, many devices, including storage media, are represented by files in the /dev directory. Open a console and use the command below to identify which block device in the system corresponds to the SD card:

```
dmesg -w
```

This will show the kernel message buffer in the console.

If using the SD card reader, you should see output similar to:

```
[21870.506727] sdb: sdb1 sdb2
[21870.509486] sd 1:0:0:0: [sdb] Attached SCSI removable disk
```

If using a built-in reader, expect the following log messages:

```
[   52.475132] mmc0: new high speed SDHC card at address 0007
[   52.475411] mmcblk0: mmc0:0007 SD8GB 7.42 GiB
[   52.480792]  mmcblk0: p1 p2
```

/dev/sdb (or /dev/mmcblk0), represents the entire raw SD card.
/dev/sdb1 (or /dev/mmcblk0p1) corresponds to the first primary partition, /dev/sdb2 (or /dev/mmcblk0p2) the second one, and so on.

To terminate a task running in a Linux console, strike Ctrl+C.

Some Linux systems automatically mount (map the filesystem on the block device to a directory) the SD card upon insertion. This might interfere with raw device access in the next step. In order to unmount the filesystem, first list the mount points:

```
mount
...
/dev/sdc1 on /media/user/Kingston type vfat (...)
```

If you notice that any of the partitions on the SD card is mounted, unmount it:

```
umount /dev/mmcblk...
```

You can also use the File Manager window to unmount the device.

Once none of the partitions are mounted, write the image to the card:

```
dd if=/path/to/somlabs-sdcard-2gb-r1.img of=/dev/sdX bs=4M oflag=dsync
```

The card image (`if` – ***I**nput **F**ile*) will be written to the card (`of` - ***O**utput **F**ile*) in 4M blocks (`bs` - ***B**lock **S**ize*) synchronously (without buffering).

Please make sure you provide the correct arguments to dd. An error may lead to data loss, even making it impossible to boot your PC again!

Use caution when issuing dd, and double check that you specified the correct target block device.

dd, by default, does not show progress. To see how much data has been transferred so far, you can pipe the input through the command pv:

pv file.bin | dd of=/dev/sd... bs=4M oflag=dsync

25.5MiB 0:00:02 [5.03MiB/s] [====>   ] 21% ETA 0:00:27

1.2. Preparing an SD card on a Windows PC

The by far easiest way to flash the SD card on Windows is to use the free Win32DiskImager software, available for download at:
https://sourceforge.net/projects/win32diskimager/

After inserting the card in the reader, enter the following information into Win32DiskImager\s main window:

- (1) path to the **.img* file with the system image,
- (2) target device drive letter assigned by the system
- (3) press *Write*.

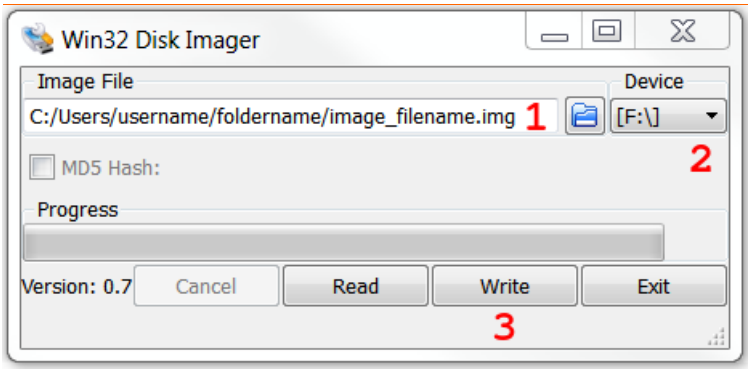


Figure 1. *Win32DiskImager main window*

1.3. First boot

Insert the microSD card into the SD slot on the module. Then, connect the microUSB cable to the connector marked in Figure 2.

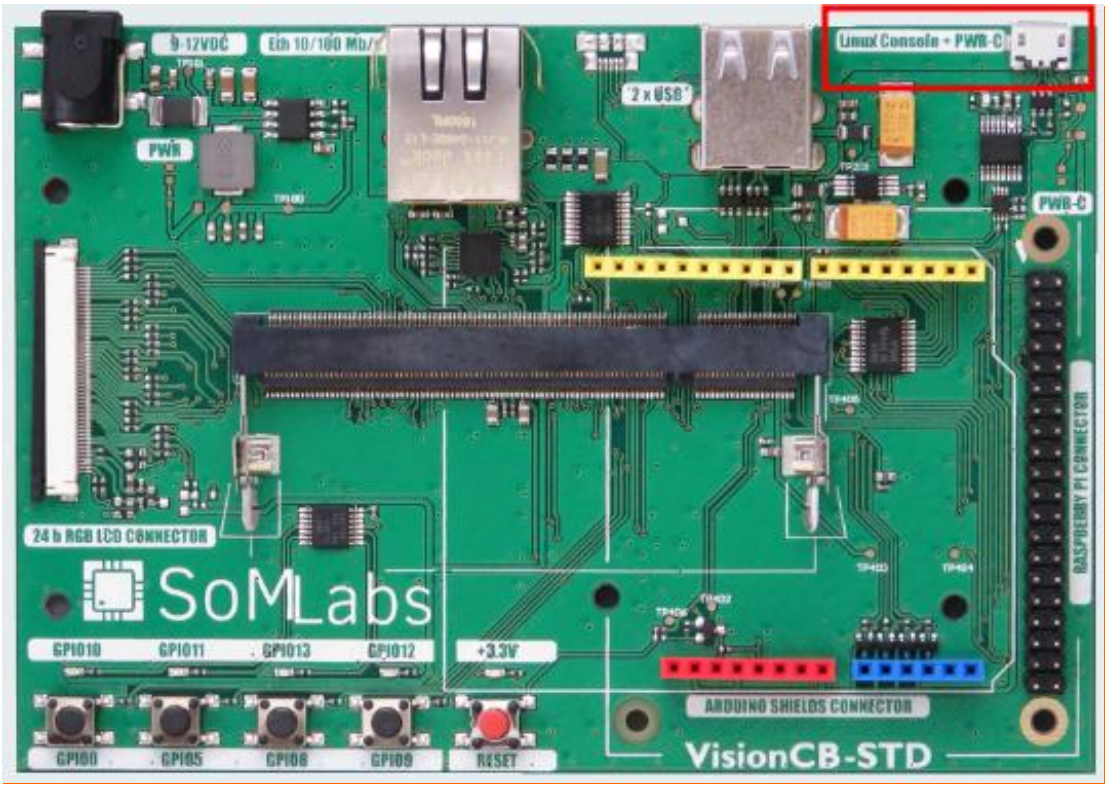


Figure 2. Serial console and power connector location on the VisionCB-STD base board

The micro-USB connector marked in Figure 2 both supplies power to the board, and connects to an FTDI-based USB↔serial converter. To access the console, open the serial port using a terminal emulator program of your choice (e.g. *minicom, picocom, screen* in Linux, or *Putty, HyperTerminal* in Windows). Configure the serial line for 115200 baud, 8N1. In Linux, we recommend using picocom:

```
picocom -b 115200 /dev/ttyUSB0
```

After opening the serial port, log in as **root**. You will not be asked to enter a password.

```
Debian GNU/Linux 9 localhost.localdomain ttymxc0

localhost login: root
root@localhost:~#
```

If you do not have sufficient permissions to run picocom, try adding 'sudo ':

```
sudo picocom -b 115200 /dev/ttyUSB0
```

This will run picocom as the root user, who has permissions to access all files on the system.

1.4. Establishing an Ethernet connection (with DHCP)

Use the included Ethernet cable to connect the board to your PC. Make sure your IPv4 settings are set to *Automatic (DHCP)*. Ethernet connecitvity is needed to access the webserver in Exercise 3.

***Do not type in** the settings below. DHCP has already been set up. This is just for your reference.*

A DHCP server (dnsmasq) has been set up in the system to allow for seamless connectivity to a PC. dnsmasq can be installed just like any other Debian package using the apt package management system:

```
sudo apt-get install dnsmasq
```

/etc/network/interfaces stores connection settings for each interface. By default, Debian expects there to be a DHCP server already in the network:

```
iface eth0 inet dhcp
```

The above line has been deleted and replaced with a static IPv4 configuration:

```
auto eth0
iface eth0 inet static
address 192.168.0.1
netmask 255.255.255.0
```

dnsmasq settings are stored in */etc/dnsmasq.conf*:

```
interface=eth0
dhcp-range=192.168.0.2,192.168.0.254,255.255.255.0,12h
dhcp-option=3
dhcp-option=6
```

The DHCP server binds to interface eth0, and allocates addresses from 192.168.0.2 to 192.168.0.254. The netmask has 24 bits (255.255.255.0), and addresses are leased for 12 hours. Options 3 and 6 specify the DNS server and gateway addresses, they have been set to empty values.

EXERCISE 2

NFC (**N**ear-**F**ield **C**ommunications) is a contactless interface enabling power and bi-directional data transfer, based on inductive coupling between two antennae.

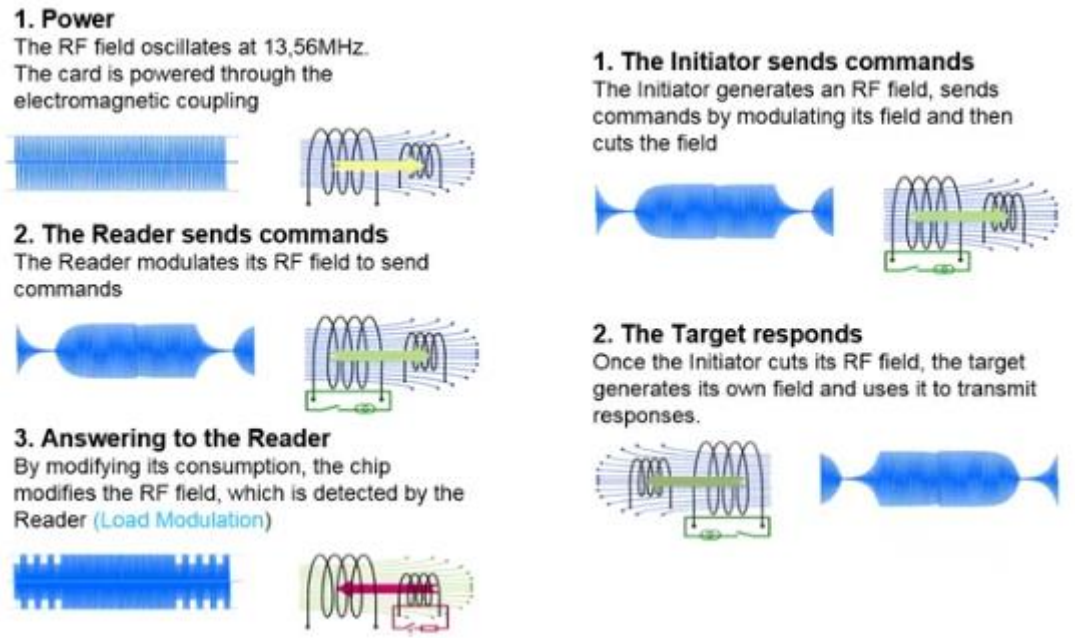


Figure 3.1. *Accessing a passive tag (left) and P2P mode*

Two devices are always involved in NFC communication: an initiator and a target. The initiator must always be active, i.e. able to power the antenna. The target can be either active or passive. A passive target is powered by energy harvested by the antenna, and uses the antenna to modulate the existing field. Virtually all tags are passive targets.

NDEF (**N**FC **D**ata **E**xchange **F**ormat) is used to carry messages via the NFC interface. It is supported by most consumer devices, including NFC-enabled Android or iOS smartphones. An NDEF message consists of one or more records. Standard record types are defined for many common payloads:

- Text
- Phone number
- E-mail address
- Geographical location
- WiFi network credentials
- MIME types
- URIs

Application-specific record types can be defined by specifying new URI schemes or MIME types.

A few example applications implementing different NFC use cases have been created for this training:

- reading and writing tags,
- exchanging NDEF messages with a smartphone (P2P mode)
- interacting with connected tags (NTAG I²C Plus)
- sharing a dynamic NDEF message inside a connected tag
- connecting to an Android smartphone through a connected tag

3.1. Hardware set-up

The PN7150 board does not have I²C pull-up resistors. Use the *SoMLabs 4ch PullUpper* board to connect the I²C bus and control signals to PN7150, as shown below.

Be careful not to cross the 5 V and 3.3 V supply connections!

IMAGE TO BE CREATED

3.2. Reading and writing tags

Let us start with two simple applications for reading and writing single-record NDEF messages from/to MIFARE Ultralight tags.

To read the contents of a tag, build and run *tag_read*:

```
root@localhost:~# cd ~/libnfc-nci-demos/tag_read
root@localhost:~/libnfc-nci-demos/tag_read# make
root@localhost:~/libnfc-nci-demos/tag_read# ./tag_read
```

Once you place the included NFC Sample Card close to the antenna, its contents will show up in the console.

You can write a new NDEF record to the tag using *tag_write*:

```
root@localhost:~# cd ~/libnfc-nci-demos/tag_write
root@localhost:~/libnfc-nci-demos/tag_write# make
root@localhost:~/libnfc-nci-demos/tag_write# ./tag_write " Hello!"
```

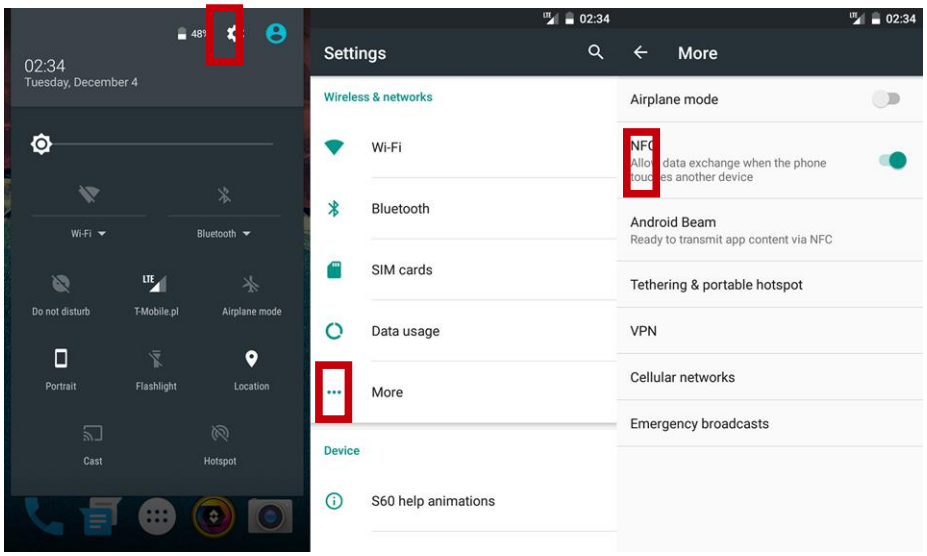
Use the *-u* switch to write an URI instead of a text record, e.g.:

```
./tag_write -u " http://nxp.com/"
./tag_write -u " mailto:witek.ewert@nxp.com"
```

3.3. Exchanging NDEF messages with a smartphone (P2P mode)

P2P mode allows two active devices to exchange NDEF messages. Instead of one device reading the other's memory, P2P mode uses SNEP (Simple NDEF Exchange Protocol) to send and receive messages. During a session, the device which has data to send is the Server, while the one ready to receive it is the Client.

Enable NFC in your Android smartphone, select: *Settings → More → NFC*



3.3.4. SNEP Server

Run the following commands in the Linux console to build this example:

```
root@localhost:~# cd ~/libnfc-nci-demos/snep_send
root@localhost:~/libnfc-nci-demos/snep_send# make
```

You can now send text and URI records through the SNEP connection, just like writing a tag with *tag_write*:

```
./snep_send "This is a text record."
./snep_send -u "http://somlabs.com/"
```

This example also allows you to push arbitrary MIME data through SNEP:

```
./snep_send -m " image/png"  ../common/nxp.png
```

Once you tap your smartphone to the antenna, Android will determine the correct action depending on the record type or MIME media type.

3.3.5. SNEP Client

snep-receive implements a SNEP client that prints the received NDEF record to standard output, just like *tag_read*:

```
root@localhost:~# cd ~/libnfc-nci-demos/snep_receive
root@localhost:~/libnfc-nci-demos/snep_receive# make
root@localhost:~/libnfc-nci-demos/snep_receive# ./snep_receive
```

Open any web page in the browser on your phone. With the page on the display, tap the phone to the NFC antenna. Confirm 'beaming' of the currently displayed content by tapping anywhere on the screen. The address of the web page should appear in the console.