Industrial IoT User Guide

Release v0.3

Table of Contents

1. Introduction

The industrial IOT project based on opensource project LEDE, source branch base is tag v17.01.2. It includes four platforms: LS1021A-IOT, LS1012A-RDB, LS1043A-RDB and LS1046A-RDB. All these platforms can be built from LEDE. IOT features each platform can support are listed as below.

LS1021A-IOT: BLE, Thread, Wi-Fi and NFC.

LS1012A-RDB: BLE, Thread, Wi-Fi, NFC and ZigBee.

LS1043A-RDB: Docker, Wi-Fi

LS1046A-RDB: Docker, Wi-Fi

The document guides how to prepare hardware setup and how to build images from LEDE project. It also demonstrates how to verify the available features.

1.1 LEDE Introduction

LEDE is a highly extensible GNU/Linux distribution for embedded devices (typically wireless routers). LEDE is based on the OpenWrt project. Unlike many other distributions for routers, LEDE is built from the ground up to be a full-featured, easily modifiable operating system for embedded devices. In practice, this means that you can have all the features you need with none of the bloat, powered by a modern Linux kernel.

For more information, please refer to the site: https://lede-project.org/.

1.2 Set up Host Environment

The following is the detailed package list on the Ubuntu host or Debian host:

```
$ sudo apt-get install build-essential libncurses5-dev gawk git subversion libssl-dev gettext unzip zlib1g-dev file python


Then get LEDE code source from tarball:
$ tar –jxvf IoT-gateway-LEDE-source-v0.3.tar.bz2
$ cd IoT-gateway-LEDE-source-v0.3/
$ unzip lede-source-master.zip
$ cd lede-source
$ ./scripts/feeds update -a
$ ./scripts/feeds install -a
```

When first run “make menuconfig” on host PC, it may prints errors that missing some packages. Install them as it prompt.

1.3 Rebuild LEDE Project

If you want to rebuild the LEDE project, you should clean the project as follows:

$make clean

$rm –rf tmp .config dl/*

1.4 LS1021A-IoT Platform

1.4.1 Switch Settings

The following table lists and describes the switch configuration for LS1021AIOT Board.

Feature	Settings (OFF=1,ON=0)	Option	Comments
S2.1	OFF	RCW & Boot Source	0 : QSPI (not supported on revA board) 1 : SDHC (default)
S2.2	OFF	SYSCLK Select	0 : DIFF_SYSCLK 1 : SYSCLK (default)
S2.3	OFF	Reserved	0 : Reserved 1 : Reserved (default)
S2.4	OFF	Reserved	0 : Reserved 1 : Reserved (default)
S2.5	OFF	SGMII2_SATA MUX	0 : SerDes Lane 2 - SATA 1 : SerDes Lane 2 – SGMII2 (default)
S2.[6 :7]	ON:ON	SYSCLK Frequency Select	00 : 100MHz (default) 01 : 99MHz 10 : 96MHz 11: Reserved
S2.8	OFF	SDA_SWD_EN Control	0 : K22 CMSIS-DAP 1 : JTAG HEADER (default)

Table 1-1 Switch Settings of LS1021aIOT

The following table lists the jumper settings:

Jumpers	Default settings on LS1021A-IOT	Description
J11	OFF	VDD_LP Source Select OFF – Battery ON - +1V0_VDDC

J18	OFF	Reserved
J19	OFF	Reserved
J20	OFF	Reserved

Table 1-2 Jumper Settings of LS1021aIOT

1.4.2 Build SDcard Image

Selecting the following options to compile SD image that could be programmed into SD card:

$ cd lede-source
$ make menuconfig
Set configures as following:
Target System (NXP Layerscape) --->
(X) NXP Layerscape
Subtarget (layerscape armv7 boards) --->
(X) layerscape armv7 boards
Target Profile (ls1021aiot) --->
(X) ls1021aiot
Target Images --->
[*] ext4 --->
[ ] squashfs ----
[ ] GZip images
(20) Boot (SD Card) filesystem partition size
(This FAT_fs partition is stored for kernel and DTB)
(256) Root filesystem partition size (in MB)
(This ext4_fs partition is stored for rootfs)
[*] Advanced configuration options (for developers) --->
[*] Toolchain Options --->
C Library implementation (Use glibc) --->
$make –j1 V=s

After building, SD card image for LS1021A-IoT is generated at:

bin/targets/layerscape/armv7/lede-layerscape-armv7-ls1021aiot-ext4-firmware.bin

1.4.3 Deployment

The SD card image for LS1021A-IoT can be programmed into a SD card directly.

$dd if=bin/targets/layerscape/armv7/ lede-layerscape-armv7-ls1021aiot-ext4-firmware.bin of=/dev/mmcblk0
Then plug the SD card into LS1021A-IoT board and start the board.

1.5 LS1012ARDB Platform

1.5.1 Switch Settings

Feature	Settings (ON=1,OFF=0)	Option	Comments
S1.1	ON	SW_RCW_SRC1	0 - Hard coded source 3 (Reserved) 1 - QSPI is the RCW source (default)
S1[2-3]	ON:ON	SW_ENG_USE/SW_ENG_USE2	XOSC Transconductance Multiplier 0 0 - 0.21x 0 1 - 0.55x 1 0 - 0.66x 1 1 - 1.00x (default)
S1.6	ON	BATT_OTG_BST_EN_B	OTG 5 V VBUS enable 0 - Enable the VBUS boost regulator of LTC4155 1 - Disable VBUS Boost regulator of LTC4155 (default)1 : Reserved (default)
S1.7	OFF	CFG_UART_MUX_EN_B	0- LS1012A UART1 is connected to K22 UART0 for CMSIS DAP debug. (default) 1 - LS1012A UART1 0 is translated to RS232 levels and is available on 1x3 header (J24).
S1.8	OFF	CFG_SERDES_MUX_SEL	SERDES Lane A MUX selection 0 - SGMII 1G to PHY (default) 1 - PCIe TX clock to mini PCIe connector
S2[1-2]	ON:ON	CFG_MUX_SDHC2_S0/CFG_MUX_SDHC2_S1	SDHC 2 interface demultiplexer select lines 00 - SDIO WiFi (default) 01 - GPIO (to Arduino) 10 - eMMC Memory 11 - SPI
S2.6	OFF	CFG_RGMII_MUX_EN_B	RGMII interface demultiplexer select 0 -> RGMII enabled (default) 1 -> SAI2 enabled (through Arduino)
S2.7	OFF	CFG_MUX_QSPI_S0/ CFG_MUX_QSPI_S1	QSPI chip-select demultiplexer select CFG_MUX_QSPI_S[1:0] 00 - CS routed to SPI memory bank 1 (default) 01 - CS routed to SPI memory bank 2 10 - CS routed to Emulator 11 - Invalid (Never use this

			option as it causes bus contention)

Table 1-3 Switch Settings of LS1012aRDB

1.5.2 Build Image

Selecting the following options to compile SD card image and QSPI image that could be programmed into QSPI flash:

```
$ cd lede-source

$ make menuconfig
Set configures as following:
Target System (NXP Layerscape)  --->
        (X) NXP Layerscape
Subtarget (layerscape 64b boards)  --->
        (X) layerscape 64b boards
Target Profile (ls1012ardb-64b)  --->
        (X) ls1012ardb-64b
Target Images  --->
        [*] ext4  --->
        [ ] squashfs  ----
        [ ] GZip images
        (20) Boot (SD Card) filesystem partition size
                (This FAT_fs partition is stored for kernel and DTB)
        (256) Root filesystem partition size (in MB)
                (This ext4_fs partition is stored for rootfs)
[*] Advanced configuration options (for developers)  --->
        [*]   Toolchain Options  --->
                C Library implementation (Use glibc)  --->
Firmware  --->
        <*> rcw-layerscape-ls1012ardb
Global build settings  --->
        Binary stripping method (none)  --->

$ make –j1 V=s
```

After building, two images for LS1012ARDB are generated:

bin/targets/layerscape/64b-glibc/lede-layerscape-64b-ls1012ardb-ext4-firmware.bin. This is a SD card image include kernel and filesystem that need to write to SD card.

bin/targets/layerscape/64b-glibc/ls1012ardb-64b-rcw-uboot.bin. This is a qspi flash image include RCW and u-boot that need to write to qspi flash.

1.5.3 Deployment

LS1012ARDB uses QSPI nor flash boot. It has two qspi nor flash bank. Do the follows to update RCW and u-boot after starting u-boot of bank0:

=> i2c mw 0x24 0x7 0xfc; i2c mw 0x24 0x3 0xf5
=>sf probe 0:0
=>tftp 84000000 tftpboot/ ls1012ardb-64b-rcw-uboot.bin
=>sf erase 0 +0x300000
=>sf write 84000000 0 $filesize
=>reset

lede-layerscape-64b-ls1012ardb-ext4-firmware.bin can be programmed into a SD card directly.

$dd if= bin/targets/layerscape/64b-glibc/lede-layerscape-64b-ls1012ardb-ext4-firmware.bin of=/dev/mmcblk0
Then plug the SD card into LS1012aRDB board and start the board.

1.6 LS1043ARDB Platform

1.6.1 Switch Settings

The following table lists and describes the switch configuration for LS1043ARDB Board to boot from NOR.

Feature	Settings (OFF=0,ON=1)	Option	Comments
SW5[1] + SW4[1-8]	0_0010_0101	RCW_SRC[0:8] select	0_0010_0101: 16-bit NOR

Table 1-4 Switch Settings of LS1043aRDB

1.6.2 Build Image

$ cd lede-source

$ make menuconfig
Set configures as following:
Target System (NXP Layerscape) --->
(X) NXP Layerscape
Subtarget (layerscape 64b boards) --->
(X) layerscape 64b boards
Target Profile (ls1043ardb-64b) --->

```
        (X) ls1043ardb-64b
Target Images  --->
        [*] ext4  --->
        [ ] squashfs  ----
        [ ] GZip images
        (20) Boot (SD Card) filesystem partition size
                (This FAT_fs partition is stored for kernel and DTB)
        (256) Root filesystem partition size (in MB)
                (This ext4_fs partition is stored for rootfs)
[*] Advanced configuration options (for developers)  --->
        [*]   Toolchain Options  --->
                C Library implementation (Use glibc)  --->

$ make –j1 V=s
```

After building, a SD card image for LS1043ARDB is generated:

bin/targets/layerscape/64b-glibc/

1.6.3 Deployment

The SD card image for LS1043aRDB can be programmed into a SD card directly.

$dd if= bin/targets/layerscape/64b-glibc/lede-layerscape-64b-ls1043ardb-ext4-firmware.bin of=/dev/mmcblk0

Then plug the SD card into LS1043aRDB board and start the board.

1.7 LS1046ARDB Platform

1.7.1 Switch Settings

The following table lists and describes the switch configuration for LS1046ARDB Board to boot from QSPI.

Feature	Settings (OFF=0,ON=1)	Option	Comments
SW5[1-8] +SW4[1]	00100000_0	RCW_SRC[0:8] select	0010_0010_0: QSPI (Default)

Table 1-5 Switch Settings of LS1046aRDB

1.7.2 Build Image

$ cd lede-source

$ make menuconfig

Set configures as following:
Target System (NXP Layerscape) --->
(X) NXP Layerscape
Subtarget (layerscape 64b boards) --->
(X) layerscape 64b boards
Target Profile (ls1046ardb-64b) --->
(X) ls1046ardb-64b
Target Images --->
[*] ext4 --->
[ ] squashfs ----
[ ] GZip images
(20) Boot (SD Card) filesystem partition size
(This FAT_fs partition is stored for kernel and DTB)
(256) Root filesystem partition size (in MB)
(This ext4_fs partition is stored for rootfs)

[*] Advanced configuration options (for developers) --->

[*] Toolchain Options --->

C Library implementation (Use glibc) --->

$ make –j1 V=s

After building, flash image for LS1046ARDB is generated:

bin/targets/layerscape/64b-glibc/

1.7.3 Deployment

The SD card image for LS1046aRDB can be programmed into a SD card directly.

$dd if= bin/targets/layerscape/64b-glibc/lede-layerscape-64b-ls1046ardb-ext4-firmware.bin of=/dev/mmcblk0

Then plug the SD card into LS1046aRDB board and start the board

1.8 FRDM-KW41Z

1.8.1 Introduction

The FRDM-KW41Z is a development kit enabled by the Kinetis® W series KW41Z/31Z/21Z (KW41Z) family built on ARM® Cortex®-M0+ processor with integrated 2.4 GHz transceiver supporting Bluetooth® Smart/Bluetooth®Low Energy (BLE) v4.2, Generic FSK, IEEE® 802.15.4 and Thread. The FRDM-KW41Z kit contains two Freedom boards that can be used as a development board or a shield to connect to a host processor.

Explore more out-of-box demos and download software and tools on: www.nxp.com/FRDM-KW41Z/startnow

1.8.2 Firmware Download

1.8.2.1 Firmware Introduction

You can get firmware from IoT-gateway-images-v0.2.tar.bz2. Following shows KW41Z firmwares where to be used.

1. ble_shell_frdmkw41z.bin: used by end device for BLE demo.
2. LS1012aRDB_hci_black_box.bin: used by LS1012aRDB for BLE demo.
3. LS1021aIOT_hci_black_box.bin: used by LS1021aIOT for BLE demo.
4. LS1012ARDB_host_controlled_device.bin: used by LS1012aRDB for Thread demo.
5. LS1012ARDB_router_eligible_device.bin: used by end device for Thread demo.
6. LS1021aIOT_host_controlled_device.srec: used by LS10121aIOT for Thread demo.
7. LS1021aIOT_end_device.srec: used by end device for Thread demo.

1.8.2.2 LS1021A-IoT Platform

Using micro-USB interface to connect KW41Z Freedom board to host PC. KW41Z board will be recognized as a new storage device, copy the firmware to this storage device, then the firmware will be downloaded into board.

1.8.2.3 LS1012ARDB Platform

Using Jlink to load the firmware into kw41 on LS1012ARDB.

1. Make sure that you install the latest J-Link driver.
   https://www.segger.com/jlink-software.html
2. Connect Jlink to SWD interface of LS1012ARDB.

Figure 1-1 Connect Jlink to LS1012aRDB

3. Using Jlink commander from start menu that is created after installing jlink driver to download firmware into kw41 on LS1012ARDB. Following the steps below for flashing the image on the microcontroller.

```
Type "connect" to establish a target connection, '?' for help
J-Link>connect
Please specify device / core. <Default>: MKW41Z512XXX4
Type '?' for selection dialog
Device>MKW41Z512XXX4
Please specify target interface:
  J) JTAG (Default)
  S) SWD
TIF>S
Specify target interface speed [kHz]. <Default>: 4000 kHz
Speed>4000
Device "MKW41Z512XXX4" selected.


Found SWD-DP with ID 0x0BC11477
Found SWD-DP with ID 0x0BC11477
AP-IDR: 0x04770031, Type: AHB-AP
AHB-AP ROM: 0xF0002000 (Base addr. of first ROM table)
Found Cortex-M0 r0p1, Little endian.
FPUnit: 2 code (BP) slots and 0 literal slots
CoreSight components:
ROMTbl 0 @ F0002000
ROMTbl 0 [0]: FFFFE000, CID: B105900D, PID: 001BB932 MTB-M0+
ROMTbl 0 [1]: FFFFF000, CID: B105900D, PID: 0008E000 MTBDWT
ROMTbl 0 [2]: F00FD000, CID: B105100D, PID: 000BB4C0 ROM Table
ROMTbl 1 @ E00FF000
ROMTbl 1 [0]: FFF0F000, CID: B105E00D, PID: 000BB008 SCS
ROMTbl 1 [1]: FFF02000, CID: B105E00D, PID: 000BB00A DWT
ROMTbl 1 [2]: FFF03000, CID: B105E00D, PID: 000BB00B FPB
Cortex-M0 identified.
J-Link>loadbin hci_black_box.bin 0
```

Figure 1-2 Use Jlink to flash firmware on KW41z

2. BLE Demo

2.1 Introduction

The Bluetooth specification has a very well defined interface between the Controller and the Host called the HCI (Host Controller Interface). This interface is defined for and can be used with various transport layers including an asynchronous serial transport layer.

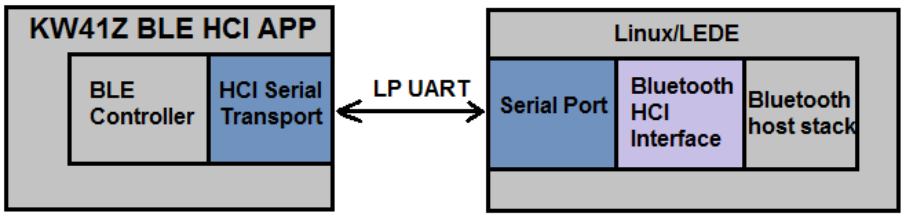

Figure 2-1 BLE HCI Structure

2.2 Hardware Preparation

2.2.1 LS1021A-IoT Platform

There are two FRDM-KW41Z Freedom boards needed, one is used as a shield to connect to LS1021A-IoT board, another is used as an end device to test BLE connection setup.

Download firmware into FRDM-KW41Z Freedom boards and used as follows:

a) FRDM-KW41Z board A which connect to LS1021A-IoT:

1. Download firmware is named **LS1021aIOT_hci_black_box.bin** into this board.
2. Plug it in arduino interface of ls1021A-IoT.
3. Jump J30 and J31 as photo below.

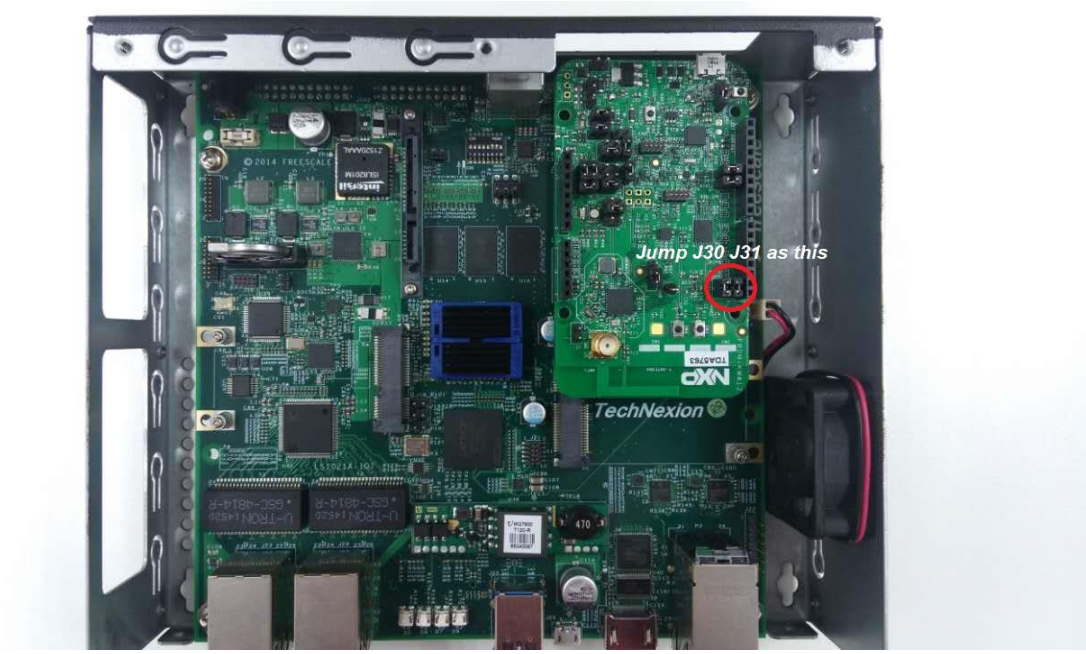

Figure 2-2 Jumper settings on FRDM-KW41Z

b) FRDM-KW41Z board B which used as end device:

1. Download firmware is named **ble_shell_frdmkw41z.bin** into this board.
2. Connect it to host PC by micro-USB interface.
3. Open a serial terminal to connect FRDM-KW41Z board by openSDA.
4. Board start log is as follow:

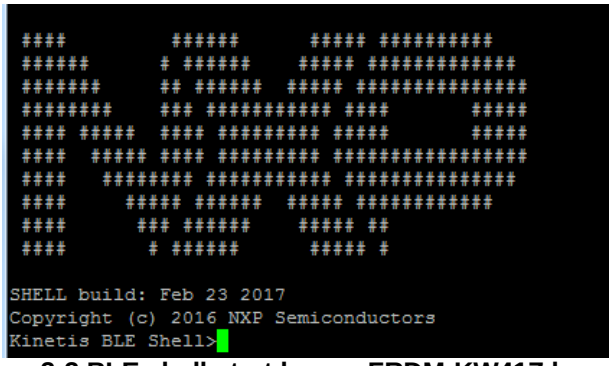

Figure 2-3 BLE shell start log on FRDM-KW41Z board B.

2.2.2 LS1012ARDB Platform

a) LS1012ARDB has a KW41z on the board which is connected by a dual-channel high performance UART bridge. The UART bridge is a SPI slave device. Because LS1012A SPI bus is muxed with LS1012A SDHC1 IOs. So Muxes are used on the board to select between the available options.

Configure the SW2 switch as:

Switch	Option	ON/OFF
SW2 [1]	CFG_MUX_SDHC2_S0	ON
SW2 [2]	CFG_MUX_SDHC2_S1	ON

Table 2-1 BLE Switch Settings of LS1012aRDB

b) Use Jlink to flash the firmware on KW41 of LS1012ARDB board: **LS1012aRDB_hci_black_box.bin**

c) FRDM-KW41Z board B which used as end device is the same as LS1021A-IoT platform.

2.3 LEDE Configuration

$make menuconfig
Set configures as following:
Utilities --->
<*> bluez-utils

2.4 Test BLE connectivity

2.4.1 LS1021A-IoT Platform

After LS1021A-IoT board startup, do the following instruction to connect LS1021A-IoT to KW41Z board B by BLE protocol.

1. Reset KW41Z board A by pressing SW1 on KW41Z board A.
2. Connect the Linux Bluetooth stack to a serial HCI interface.
   $ hciattach /dev/ttyLP0 any 115200 noflow nosleep

   Device setup complete
3. Start the hci0 interface manually.
   $ hciconfig hci0 up
4. Run the hciconfig command with no parameters to check the HCI interface.
   $ hciconfig
   hci0: Type: BR/EDR Bus: UART
   BD Address: 00:04:9F:00:00:15 ACL MTU: 500:20 SCO MTU: 0:0
   UP RUNNING
   RX bytes:462 acl:0 sco:0 events:32 errors:0
   TX bytes:240 acl:0 sco:0 commands:32 errors:0
5. Advertising BLE on KW41Z board B.
   Kinetis BLE Shell> gap advstart
6. Scan for Advertising LE Devices.
   $ hcitool -i hci0 lescan

   LE Scan ...

   00:04:9F:00:00:16 (unknown)
7. Obtaining Remote LE Device (KW41Z board B) Information.
   $ hcitool -i hci0 leinfo 00:04:9F:00:00:16

   Requesting information ...

   Handle: 705 (0x02c1)

   LMP Version: 4.2 (0x8) LMP Subversion: 0x121

   Manufacturer: NXP Semiconductors (formerly Philips Semiconductors) (37)

   Features: 0xff 0x00 0x00 0x00 0x00 0x00 0x00 0x00
8. Connect to a remote LE device (KW41Z board B).
   # gatttool -I

   [ ][LE]> connect 00:04:9F:00:00:16

   Attempting to connect to 00:04:9F:00:00:16

   Connection successful

9. After disconnect from BLE network, if you want to connect it again, you should do from step 5.

2.4.2 LS1012ARDB Platform

After LS1012ARDB board startup, do the following instruction to connect the Linux Bluetooth stack to a serial HCI interface.

$ hciattach /dev/ttySC1 any 9600 noflow nosleep

Device setup complete

Then do the same as LS1021A-IoT platform to test BLE demo.

3. Thread Demo

3.1 Introduction

Kinetis Thread Stack provides Thread networking components over IEEE-802.15.4 MAC 2006 layer running on Kinetis MCUs which are enabled to use IEEE 802.15.4. It provides a CoAP application profile API.

The Kinetis Thread stack implements a serial Tunnel media interface which can be used to exchange FSCI encapsulated IPv6 packets with a host system. To provide connectivity to the host, there are 2 components needed: the TUN/TAP kernel module, which allows the operating system to create virtual interfaces and a program that knows to encapsulate/decapsulate IP packets to/from FSCI/THCI.

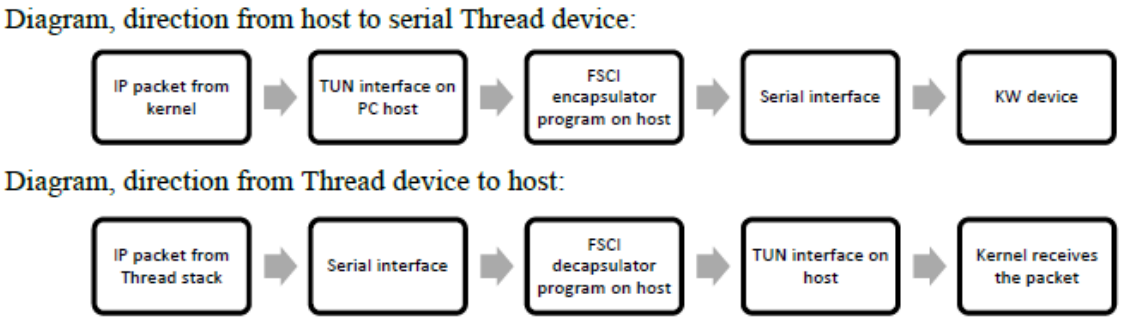

Figure 3-1 Thread Stack Structure

3.2 Hardware Preparation

3.2.1 LS1021A-IoT Platform

Prepare hardware as BLE demo hardware prepare.

Download following firmware into FRDM-KW41Z boards:

a) FRDM-KW41Z board A which connect to LS1021A-IoT:
LS1021aIOT_host_controlled_device.srec
b) FRDM-KW41Z board B which used as end device:
LS1021aIOT_end_device.srec

Board start log is as follow:

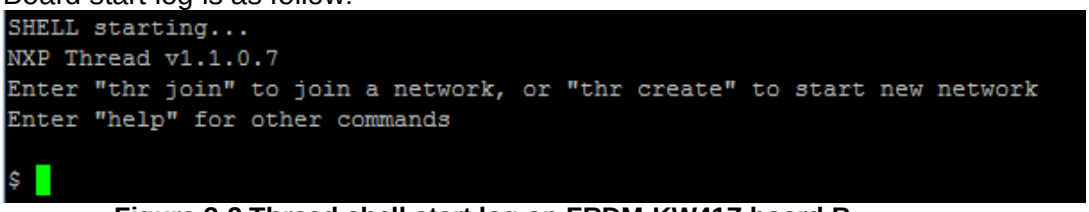

Figure 3-2 Thread shell start log on FRDM-KW41Z board B.

3.2.2 LS1012ARDB Platform

Prepare hardware as BLE demo hardware prepare.

Download following firmware into FRDM-KW41Z boards:

a) Use Jlink to flash the firmware on KW41 of LS1012ARDB board:
LS1012ARDB_host_controlled_device.bin
b) Download following firmware into FRDM-KW41Z board B which used as end:
LS1012ARDB_router_eligible_device.bin

3.3 LEDE Configuration

$make menuconfig

Set configures as following:

Network --->

Routing and Redirection --->

<*> ip-full

3.4 Test Thread connectivity

3.4.1 LS1021A-IoT Platform

After LS1021A-IoT board startup, do the following instruction to connect ls1021A-IoT to KW41Z board B by Thread protocol.

1. Create a TUN/TAP interface and add ip route tables.
$ make_tun.sh
This script automates the creation of the TUN interface, while configuring it with the proper IPv6 addresses and routes.
$ cat make_tun.sh
#!/bin/bash
Create a new TUN interface for Thread interaction.
ip -6 tuntap add mode tun fslthr0
Assign it a global IPv6 address.
ip -6 addr add FD01::2 dev fslthr0

```
# Add route to default address of Serial TUN embedded interface.
ip -6 route add FD01::1 dev fslthr0
# Add route to Unique Local /64 Prefix via fslthr0.
ip -6 route add FD01:0000:0000:3EAD::/64 dev fslthr0
# The interface is ready.
ip link set fslthr0 up
# Enable IPv6 routing on host.
sysctl -w net.ipv6.conf.all.forwarding=1
```

2. Reset KW41Z board A by pressing SW1 on KW41Z board A.
3. Create thread network on LS1021A-IoT.
   $Thread_KW_Tun /dev/ttyLP0 fslthr0 1 15 115200 &
   [170.543616] IPv6: ADDRCONF(NETDEV_CHANGE): fslthr0: link becomes ready
   WARNING: Cannot open /usr/share/hsdk/hsdk.conf => FSCI ACKs are disabled
   [THR] Factory Reset OK!
   [THR] Create Network OK!
   [THR] Border Router Add Prefix OK!
   [THR] Border Router Sync Prefix OK!
   [MESHCOP] Start Commissioner OK!
   [MESHCOP] Add Expected Joiner OK!
   [MESHCOP] Sync Steering Data OK!

4. Join thread network on FRDM-KW41Z board B. Do the following after add expected joiner is ok on LS1021a-IOT board.
   $ thr join
   Joining a Thread network...
   Commissioning successful

5. Test thread connectivity by ping tun/tap interface IP of LS1021A from FRDM-KW41Z board B.
   $ping FD01::2
   $ifconfig
   Interface 0: 6LoWPAN
   Mesh local address (ML64): fdc8:9eea:ea7c::8ed:6664:78bf:826f
   Mesh local address (ML16): fdc8:9eea:ea7c::ff:fe00:400
   Unique local address: fd01::3ead:b0f0:e421:57e9:562a

6. Ping the remote device IP(use the Unique local address) from LS1021a.
   $ping fd01::3ead:b0f0:e421:57e9:562a

3.4.2 LS1012ARDB Platform

After LS1012ARDB board startup, do the same steps as LS1021A-IoT to test thread demo. The only difference is how to use Thread_KW_Tun to create thread network on LS1012ARDB.

$Thread_KW_Tun /dev/ttySC1 fslthr0 1 15 9600 &

4. NFC Demo

4.1 Introduction

The PN7150 NFC module package comes with the following items

1) PN7150 Module
2) Arduino Shield Interface board (OM5578)
3) NFC Sample Tag

Figure 4-1 NFC module package

LS1021A-IoT and LS1012ARDB platforms use OM5578/PN7150ARD module, which connects to I2C-0 bus as a NFC module through Arduino interface. The following diagram depicts the required hardware connection signals between PN7150 NFC module and LS1021A-IoT/LS1012ARDB boards.

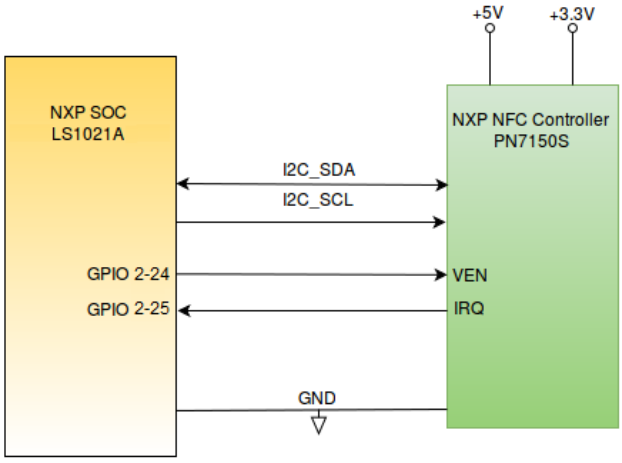

Figure 4-2 Structure of NFC connect to LS1021a

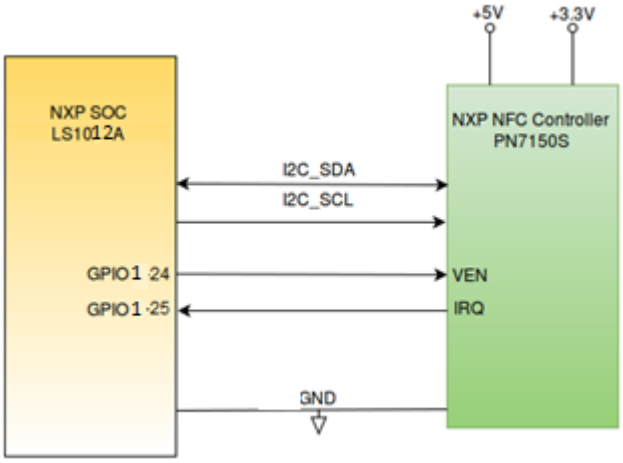

Figure 4-3 Structure of NFC connect to LS1012a

4.2 Hardware Preparation

4.2.1 LS1021A-IoT Platform

Because LS1021A-IoT board does not connect GPIO pins to Arduino interface, we need to connect two GPIO pins to Arduino interface, like the photo below. (Link: J8.1->J502.3, J17.8->J502.5).

Then, plug the PN7150 NFC card into Arduino interface of LS1021A-IoT.

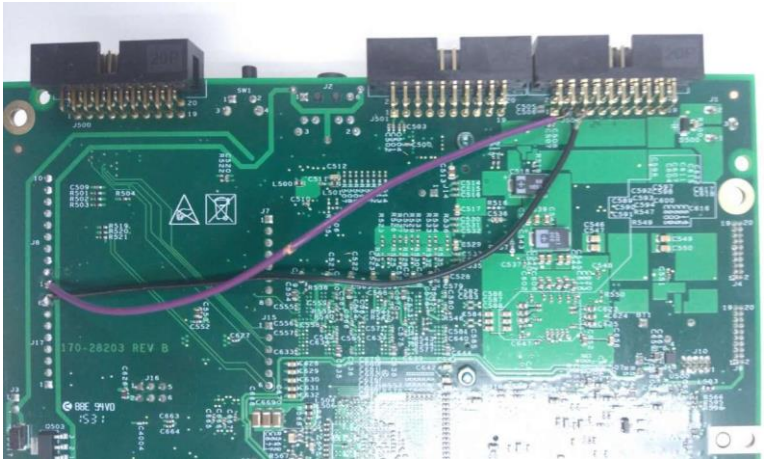

Figure 4-4 NFC hardware rework on LS1021aIOT

4.2.2 LS1012ARDB Platform

Just now LS1012ARDB RevC board support NFC, in order to make the NFC feature work correctly, need to rework the hardware, short circuit J16.2 and J17.14, as below photo shows.

Figure 4-5 NFC hardware rework on LS1012aRDB

In order to support NFC on LS1012aRDB RevD board, you need to configure the SW2 switch as: SW2[1-2]: ON/ON before power on, then set the SW2 switch as: SW2[1-2]: ON/OFF after power on the board.

4.3 LEDE Configuration

4.3.1 LS1021A-IoT Platform

Set configures as following:

$make menuconfig
Kernel modules ---->
Other modules ---->
<> kmod-nxp_pn5xx*
Utilities ---->
<> libnfc-nci*

4.3.2 LS1012ARDB Platform

$make menuconfig
Kernel modules ---->
Other modules ---->
<> kmod-nxp_pn5xx*
Utilities ---->
<> libnfc-nci*

Note: Using NFC module on LS1012aRDB, the RCW will be changed to modify mux_GPIO pins. You need to clean the project and rebuild it to generate images.

4.4 Test NFC feature

Make sure the U-boot has identify the NFC module before startup the Linux Kernel with below command in uboot prompt:

```
=> i2c probe
=> Valid chip addresses: 00 08 09 1E 20 24 25 26 28 40 7C
```

I2C address 0x28 is for OM5578/PN7150 module, if not identify it, please re-install the hardware and repeat the former command.

4.4.1 LS1021A-IoT Platform

After LS1021A-IoT board startup, do the below instruction to start NFC demo.

$ ./data/nfc/nfcDemoApp poll

4.4.2 LS1012ARDB Platform

Make sure the SW2[1:2] = on:on.

Make sure below sequences of power are implemented:

1) Connect USB/serial port "CONSOLE" to PC USB port.
2) Connect the supplied AC power to LS1012ARDB RevC board.

In u-boot, make sure the correct RCW binary is being used, as below red shows:

Reset Configuration Word (RCW):

00000000: 08000008 00000000 00000000 00000000

00000010: 35080000 c000000c 40000000 00001800

00000020: 00000000 00000000 00000000 00014511

00000030: 00000000 18c2a120 00000096 00000000

After LS1012ARDB board startup, do the below instruction to start NFC demo.

$ ./data/nfc/nfcDemoApp poll

4.4.3 Logs

When the application is running, the print log is:

```
root@LEDE:/# nfcDemoApp poll
################################################################################
#########
##                                    NFC demo
      ##
################################################################################
#########
##                                Poll mo[   23.182571] pn54x_dev_open : 10,58
de activated   [   23.186872] pn54x_dev_ioctl, cmd=1074063617, arg=1
               [   23.192908] pn544_enable power on
               ##
################################################################################
#########
                             ... press enter to quit ...

[   23.413046] pn54x_dev_ioctl, cmd=1074063617, arg=0
[   23.417809] pn544_disable power off
[   23.633045] pn54x_dev_ioctl, cmd=1074063617, arg=1
[   23.637807] pn544_enable power on
BrcmHcpX8103
BrcmHcpR8180
BrcmHcpX810103020304
BrcmHcpR8180
BrcmHcpX8101010194fffffe93ffff
BrcmHcpR8180
BrcmHcpX810204
BrcmHcpR818000
Waiting for a Tag/Device...
```

Figure 4-6 NFC Demo log

Touching a NFC sample tag to OM5578/PN7150 NFC reader, the application print log:

```
Waiting for a Tag/Device...

[  208.433090] random: nonblocking pool is initialized
        NFC Tag Found

                Type :          'Type A - Mifare Ul'
                NFCID1 :        '04 67 66 D2 9C 39 81 '
                Record Found :
                                NDEF Content Max size :         '868 bytes'
                                NDEF Actual Content size :      '29 bytes'
                                ReadOnly :                      'FALSE'
                                Type :                          'URI'
                                URI :                           'http://www.nxp.com/d
emoboard/OM5578'


                29 bytes of NDEF data received :
                D1 01 19 55 01 6E 78 70 2E 63 6F 6D 2F 64 65 6D 6F 62 6F 61 72 64 2F
4F 4D 35 35 37 38

        NFC Tag Lost

Waiting for a Tag/Device...
```

Figure 4-6 NFC Demo log after recognizing sample tag

5. Wifi Demo

5.1 Introduction

A WNC DNXA-H1 card is used for the WiFi verification. And the corresponding kernel driver ATH9K also be set in default. This demo can be used on LS1021aIOT LS1012aRDB LS1043aRDB and LS1046aRDB. The LS1021A-IoT board has two mini-PCIe slots so the max two WiFi cards can be inserted. It cannot be inserted in MPCIE SLOT2 if there is only one WIFI card.

5.2 Hardware Preparation

Insert a WNC DNXA-H1 card as wifi card into mPCIE slot of LS1021A-IoT board like this.

Figure 5-1 Wifi card on LS1021aIOT

5.3 LEDE Configuration

There is only need LEDE configuration as follows for LS1021aIOT board.

$make menuconfig
Set configures as following:
Libraries ---->
    <> libnfnetlink*
    <> libnftnl*
    <> libnl*
Network ---->
    <> hostapd*
    -- hostapd-common*
    <> hostapd-utils*
Kernel modules ---->
    Wireless Drivers ---->
        <> kmod-ath9k*
Base system ---->
    <> dnsmasq*
Luci--->

Collection ---->
<> luci*

Following steps is configuration for LS1012aRDB, LS1043aRDB and LS1046aRDB.

1. Kernel configuration.
   $make kernel_menuconfig

   [*] Networking support ---->
   -- Wireless ---->*
   [*] Wireless extensions
   [*] WEXT_SPY
   [*] WEXT_PRIV
   <> cfg80211 - wireless configuration API*
   [*] enable powersave by default
   <> Generic IEEE 802.11 Networking Stack (mac80211)*
   [*] Minstrel
   [*] Minstrel 802.11n support
   -- Enable LED triggers*

   Device Drivers ---->
   [*] Network device support ---->
   [*] Wireless LAN ---->
   <> Atheros Wireless Cards ---->*
   <> Atheros 802.11n wireless cards support*
   [*] Atheros bluetooth coexistence support
   [*] Atheros ath9k PCI/PCIe bus support
   [*] Atheros ath9k support for PC OEM cards

2. LEDE configuration
   $make menuconfig
   Libraries ---->
   <> libnfnetlink*
   <> libnftnl*
   <> libnl*
   Network ---->
   <> hostapd*
   -- hostapd-common*
   <> hostapd-utils*
   Kernel modules ---->
   Wireless Drivers ---->
   <> kmod-ath9k*
   Base system ---->
   <> dnsmasq*
   Luci--->

Collection ---->
<> luci*

5.4 Test WiFi Demo

When first time startup the board, do the following instructions to set WiFi up.

1). Generate wifi config file.

$wifi config

2). Modify wifi config file to delete disable option.

$ vi /etc/config/wireless

#option disabled '1'

3). Set eth1 which is connected to network as wan.

$vi /etc/config/network

```
config interface 'lan'
        option type 'bridge'
        option ifname 'eth0'
        option proto 'static'
        option ipaddr '192.168.1.1'
        option netmask '255.255.255.0'
        option ip6assign '60'

config interface 'wan'
        option ifname 'eth1'
        option proto 'dhcp'
```

3). start wifi up.

$ wifi up

```
[  317.647622] IPv6: ADDRCONF(NETDEV_UP): wlan0: link is not ready
[  317.655381] device wlan0 entered promiscuous mode
[  317.669190] IPv6: ADDRCONF(NETDEV_CHANGE): wlan0: link becomes ready
[  317.675638] br-lan: port 2(wlan0) entered forwarding state
[  317.681116] br-lan: port 2(wlan0) entered forwarding state
[  317.686689] IPv6: ADDRCONF(NETDEV_CHANGE): br-lan: link becomes ready
[  319.672689] br-lan: port 2(wlan0) entered forwarding state
[  327.782870] device wlan0 left promiscuous mode
[  327.787420] br-lan: port 2(wlan0) entered disabled state
[  328.008825] IPv6: ADDRCONF(NETDEV_UP): wlan0: link is not ready
[  328.017056] device wlan0 entered promiscuous mode
[  328.021790] br-lan: port 2(wlan0) entered forwarding state
[  328.027288] br-lan: port 2(wlan0) entered forwarding state
[  328.042613] IPv6: ADDRCONF(NETDEV_CHANGE): wlan0: link becomes ready
[  330.022688] br-lan: port 2(wlan0) entered forwarding state
```

4). You can join the wifi by your mobile phone, the SSID of WIFI is LEDE.

6. Docker Demo

6.1 Introduction

Docker is a tool designed to make it easier to create, deploy, and run applications by using containers. Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package. By doing so, the developer can rest assured that the application will run on any other Linux machine regardless of any customized settings that machine might have that could differ from the machine used for writing and testing the code.

This chapter shows the steps to run Docker on LEDE filesystem environment and NXP ls1043ardb, ls1046ardb platform with 64B images.

Note: Since the LEDE doesn't include the Docker package, so this demo uses Docker binaries built standalone in 64B and copy them into the LEDE filesystem by the patch:

```
0001-lede-docker-Add-docker-support-for-ls1043ardb-ls1046.patch
```

Get the patch from the IoT release v0.2 tarball <IoT-gateway-LEDE-source-v0.2.tar.bz2>, apply it to the LEDE open source tree and lede-17.01 branch, an entire Docker running environment can be built out by LEDE.

6.2 Hardware Preparation

Make sure the ls1043ardb, ls1046ardb are bootup from SD card based on the following SW setting.

Platform	Boot	Board SW Setting
ls1043ardb	SD card	SW4[1-8] +SW5[1] = 0b'00100000_0
ls1046ardb	SD card	SW5[1-8] +SW4[1] = 0b'00100000_0

6.3 Docker test case

1. Obtain the LEDE project code and check to the lede-17.01 branch

```
git clone https://github.com/lede-project/source.git lede
cd lede
git checkout remotes/origin/lede-17.01 -b lede-17.01
./scripts/feeds update -a
./scripts/feeds install -a
```

2. Apply the LEDE patch for docker case.

```
cd lede
```

```
git am 0001-lede-docker-Add-docker-support-for-ls1043ardb-ls1046.patch
```

3. Use the lede-1701-docker-config as the LEDE .config

```
cd lede
mv lede-1701-docker-config .config
```

4. Check the menuconfig

```
cd lede
make menuconfig
# choose the ls1043ardb 64b or ls1046ardb 64b
# exit and save
```

5. Compile the LEDE and get the final image

```
cd lede
make -j8 V=s
# get the binary image for ls1043ardb or ls1046ardb at
# bin/targets/layerscape/64b-glibc/
# named ‘lede-layerscape-64b-ls1043ardb-ext4-firmware.bin’ or
# ‘lede-layerscape-64b-ls1046ardb-ext4-firmware.bin’
```

6. Program the binary image into a SD card under a host Linux machine

```
sudo dd if=./lede-layerscape-64b-ls1043ardb-ext4-firmware.bin of=/dev/sdx
# check the SD card name "/dev/sdx" in your machine and replace the # “sdx” in above command
```

7. Insert the programmed SD card into the ls1043ardb or ls1046ardb board and boot up

8. Set the LEDE system network (an example on ls1043ardb board)

```
# set the Ethernet interface IP addr
root@LEDE:/# ifconfig eth2 10.192.208.230
# set the default gateway
```

```
root@LEDE:/# route add default gw 10.192.208.254
# set the DNS server
root@LEDE:/# vi /etc/resolv.conf
# add the namespace in resolv.conf like:
    nameserver 10.192.130.201
    nameserver 10.228.49.200
    nameserver 10.201.141.100
# save and exit
```

9. Run docker and aarch64/hello-world docker image

```
root@LEDE:/# mkdir docker
root@LEDE:/# docker daemon --graph=/docker&
root@LEDE:/# docker pull aarch64/hello-world
root@LEDE:/# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
aarch64/hello-world   latest              70a49d30fa8f        3 months ago        2.088 kB
root@LEDE:/# docker run --net=host aarch64/hello-world
```

7. ZigBee Scenario

ZigBee demonstration scenario bases on LS1012ARDB, JN516x-EK004 Evaluation Kit, refer to another user manual “Zigbee Solution for Industrial IoT User Guide Release v0.2”.

8. OTA Implementation

8.1 Introduction

OTA refers to a method of updating u-boot, kernel, filesystem and even the full firmware to devices through network. If the updated firmware is not working, the device can rollback the firmware to latest version automatically. Because there is no hardware method to rollback device automatically, device could not rollback once the U-boot is not working when updating u-boot and full firmware.

OTA implementation use some shell scripts and configure file to update and rollback:

version.json: This is a JSON file which saves the board name and version of each firmware. Below is an example of version.json.

```
{
  "updatePart":"kernel",  /* Name of firmware image which has been updated. */
  "updateVersion":"1.0.0", /* Version of firmware image which has been updated. */
  "all":"0.1.0",     /* version of the full firmware image which has been used now */
  "u-boot":"0.1.0", /* version of the u-boot image which has been used now */
  "kernel":"0.1.0", /* version of the kernel image which has been used now */
  "filesystem":"0.1.0", /* version of the filesystem image which has been used now */
  "boardname":"ls1021aiot" /* used to get the corresponding firmware from server*/
}
```

update.json: This file is stored in server, it saves the name and version of firmware image which will be updated.

Below is an example of update.json:

```
{
  "updateStatus":"yes", /* set yes or no to tell devices is it need to update. */
  "updatePart":"kernel", /* name of update firmware. */
  "updateVersion":"0.3.0", /* version of update firmware */
}
```

update_ota: This script can get a JSON file named update.json from server, then parse the file and get the new firmware version to confirm whether to download it from server or not, finally write the firmware into SD card to instead the old one. After that, save the "updatePart" and "updateVersion" into version.json, and mark the update status on 4080 block of SD card to let u-boot know it.

startupVersioncheck.sh: This script will check if the firmware has been updated, then update the version of the update part in version.json, and clean the flag of update status on 4080 block of SD card. This script will run automatically each time the system restart.

rollback.sh: This script is running on the ramdisk filesystem after the filesystem is updating fail. It gets old firmware version from version.json and then update it from server.

8.2 LEDE Configuration

Set configures as following:

```
$make menuconfig
        Network  --->
                File Transfer  --->
                        <*> wget
Libraries  --->
        <*> libustream-openssl
```

8.3 OTA Test case

1. Updating u-boot

Write update.json on server like this:

```
{
    "updateStatus":"yes",
    "updatePart":"u-boot",
    "updateVersion":"0.4.0",
}
```

Upload the u-boot image on server path: *0.4.0/<boardname>/u-boot.bin*

Run update_ota command on device board.

2. Updating kernel
   Set the "updatePart" to "kernel" in update.json.
   Upload the kernel image on server path: *0.4.0/<boardname>/uImage*
   Run update_ota command on device board.
3. Updating filesystem
   Set the "updatePart" to "filesystem" in update.json.
   Upload the filesystem image on server path: *0.4.0/<boardname>/rootfs.ext4*
   Run update_ota command on device board.

4. Updating full firmware
   Set the "updatePart" to "all" in update.json.
   Upload the full firmware image on server path: *0.4.0/<boardname>/firmware_sdcard.bin*
   Run update_ota command on device board.

5. Rollback test
   Kernel and filesystem can use a wrong image to upload on server and test update on device.

9. 4G-LTE Modem

9.1 Introduction

A HuaWei E3372 USB Modem is used for the 4G network verification.

9.2 Hardware Preparation

Insert a USB Modem into USB slot of LS1021A-IoT board.

9.3 LEDE Configuration

Set configures as following:

```
$make menuconfig
    Utilities  --->
        <*> usb-modeswitch
```

Kernel configuration.

```
$make kernel_menuconfig
Device Drivers  --->
```

[*] Network device support --->
<> USB Network Adapters --->*
<> Multi-purpose USB Networking Framework--->*
<> CDC Ethernet support*
<> CDC EEM support*
<> CDC NCM support*

9.4 Test 4G USB modem link to the internet

Do the following instructions to set 4G Modem up.

1. Set eth3 connected to network.

```
root@LEDE:/# vi /etc/config/network
# add the wan in network like:
config interface 'wan'
        option ifname 'eth3'
        option proto 'dhcp'

# save and exit
```

2. Test 4G modem link to the internet.

```
root@LEDE:/# ping www.nxp.com
PING www.nxp.com (210.192.117.231): 56 data bytes
64 bytes from 210.192.117.231: seq=0 ttl=52 time=60.223 ms
64 bytes from 210.192.117.231: seq=1 ttl=52 time=95.076 ms
64 bytes from 210.192.117.231: seq=2 ttl=52 time=89.827 ms
64 bytes from 210.192.117.231: seq=3 ttl=52 time=84.694 ms
64 bytes from 210.192.117.231: seq=4 ttl=52 time=68.566 ms
64 bytes from 210.192.117.231: seq=5 ttl=52 time=89.809 ms
```

10. Known Issues

Item	Description
