WILC Linux Driver Porting Guide for NXP i.MX SABRE Board

Based on the i.MX 6 Series

INTRODUCTION

This porting guide describe the steps to interface WILC devices with the NXP i.MX by adding the WILC Linux driver. NXP i.MX SABRE evaluation board has been used as the platform to interface WILC device.

The Smart Application Blueprint for Rapid Engineering (SABRE) board for smart devices introduces developers to quad-core processing, low-power consumption and bleeding-edge multimedia and graphics applications on the i.MX 6Quad applications processor based on the Arm® Cortex™-A9 core. The Board is powered using 5 V/5 A universal power supply. There are 2 SD card slot available in the board and the development kit is booting from the SD card. The other SD card slot can be used to connect the WILC device for the Wi-Fi connectivity.

Official webpage link for SABRE board with i.MX 6QuadPlus processor is :
https://nxp.com/SABRESDB

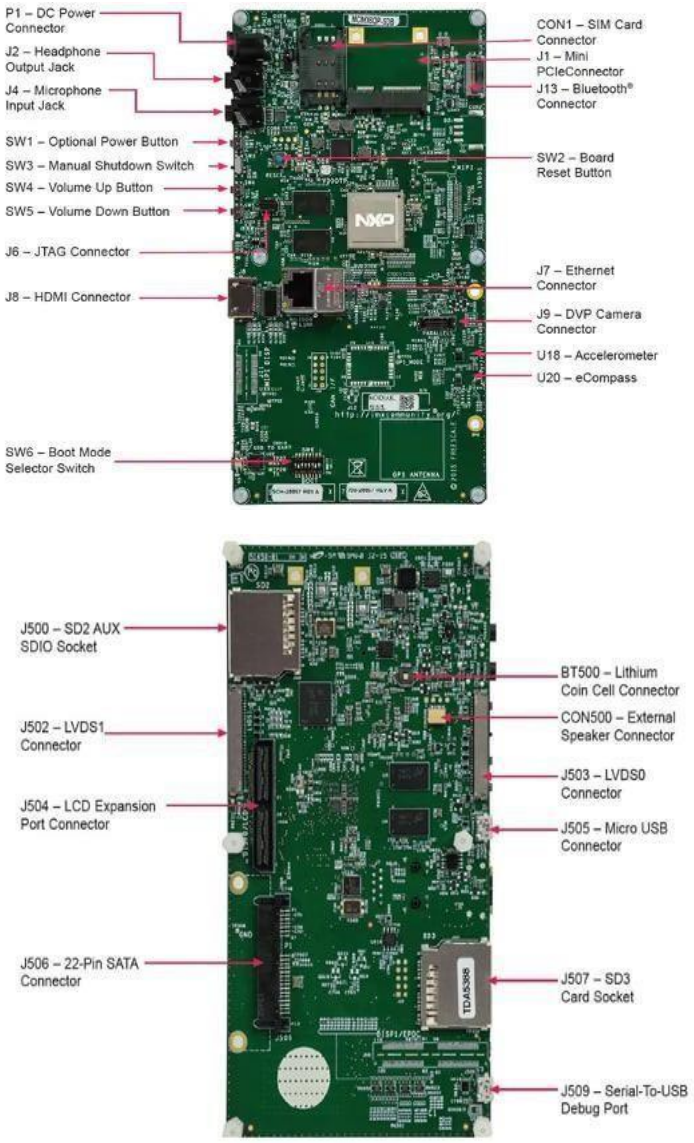
PREREQUISITES

- Hardware prerequisites:
  - ATWILC1000 SD
  - i.MX6 SABRE Board
  - Linux Host machine
  - Access Point
  - USB SD Card adaptor
- Software prerequisites:
  - i.MX6 SABRE board BSP
  - Linux kernel
  - Etcher software to flash SD Card image
  - ATWILC1000 Linux driver source code
  - ATWILC1000 firmware binary

FEATURES

Processor	• i.MX 6QuadPlus 1 GHz processor based on the ARM® Cortex®-A9 core
Development for	• i.MX 6QuadPlus and i.MX 6DualPlus
Memory/Storage	• 1 GB DDR3 SDRAM up to 533 MHz (1066 MTPS) memory • 8 GB eMMC flash
Display	• 2 x LVDS connectors • HDMI connector • LCD expansion connector (parallel, 24-bit) • MIPI DSI connector (two data lanes, 1 GHz each)
User Interface	• Power, reset, volume buttons
Power Management	• NXP MMPF0100F9
Audio	• Audio codec • Microphone and headphone jacks
Expansion Connector	• Camera MIPI CSI port • I²C, SSI, SPI signals
Connectivity	• 2 x Full-size SD/MMC card slots • 22-pin SATA connector • 10/100/1000 Ethernet port • 1 x USB 2.0 OTG port (micro USB) • mPCIe® connector
Debug	• JTAG connector (10-pin) • 1x Serial-to-USB connector (for JTAG)
OS Support	• Linux® and Android™ • Others supported third party (QNX, Windows® Embedded)
Tools Support	• Manufacturing Tool • Processor Expert IOMUX tool
Additional Features	• NXP MMA8451 three-axis accelerometer • NXP MAG3110 three-axis magnetometer • USB plug power supply • NXP 3D magnetometer

SABRE Development Board



SWITCH FUNCTIONS

Item	Description
POWER SW1	Prolonged depress (>5 sec) will force an immediate hardware shutdown
RESET SW2	Momentary depress of button will reset the system and begin a boot sequence.
SHUTDOWN SW3	. Sliding the switch to the ON position connects the 5 V power supply to the SABRE board main power system . Sliding the shutdown switch immediately removes all power to the board.

Before the Linux® OS kernel can boot on an i.MX board, the **Linux® image** needs to be copied to a **boot device** and the **boot switches** need to be set accordingly.

To bring up the board and run Linux®, four elements are needed:

- Boot loader (U-Boot)
- Linux® kernel image (zImage)
- A device tree file (.dtb) for the board being used
- A root file system (rootfs)

The release contains a pre-built SD card image that is built specifically for the i.MX 6Quad SabreSD board. The SD card image is typically named <board name.sdcard> and is a specially constructed disk image including partitions and all necessary files to boot the board, including all four components mentioned above. Before the Linux OS kernel can boot on an i.MX board, the images (U-Boot, Linux kernel, device tree, and rootfs) need to be copied to a boot device and the boot switches need to be set to boot that device.

DIP Switch Configuration

The boot modes of the i.MX boards are controlled by the boot configuration DIP switches on the board.

Below table shows the DIP switch settings for booting the smart device from SD card slot SD3 on the i.MX 6 SABRE-SD boards.

DIP Switch (SW6)

D1	D2	D3	D4	D5	D6	D7	D8
Off	On	Off	Off	Off	Off	On	Off

The default SD card image can be flashed directly by dd command or by using Etcher application. Flashing the sd card image through Etcher will be the simplest way to load everything needed onto the card.

The other way to create a SD card image is to set up the partitions manually and individually load the bootloader, kernel, and rootfs. In this document, we will be dealing with the direct SD card image that comes along with the release.

The. sdcard image contains all four images properly configured for an SD card. When more flexibility is desired, the individual components can be loaded separately.

By default, the release uses the following layout for the images on the SD card. The kernel image and DTB uses the FAT partition without a fixed raw address and rootfs uses the EXT4 partition on the SD Card.

Start address (sectors)	Size (sectors)	Format	Description
0x400 bytes (2)	0x9FFC00 bytes (20478)	RAW	U-Boot and reserved area
0xa00000 bytes (20480)	500 Mbytes (1024000)	FAT	Kernel zImage and DTBs
0x25800000 bytes (1228800)	Remaining space	Ext3/Ext4	Rootfs

Demo SD Card Package

The ready to load release package is available at the “Embedded Software” subsection under SOFTWARE session on the official website of nxp https://nxp.com/SABRESDB

The **Embedded Software** section lists the various packages for different Linux versions. This document is prepaid based on the testing done with the Linux version 4.9.88 (**L4.9.88_2.0.0_MX6QDLSOLOX**). Extract the file to get the sd card image.

- ✓ Board Support Package will contain the SD card zip file **fsl-image-qt5-validation-imx-xwaylandimx6qpdlsolox.sdcard.bz2**
- ✓ Extracting the above compressed file will generate the already portioned SD card image **fsl-imageqt5-validation-imx-xwayland-imx6qpdlsolox.sdcard**

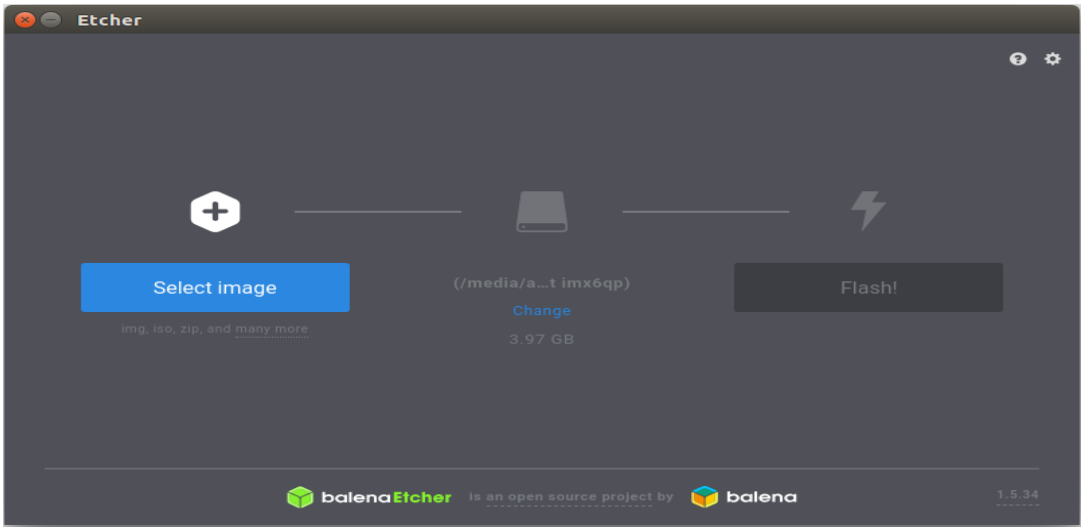
To flash the SD card image, a 4GB SD card is used. Flashing the SD card image into the SD card is carried out using the Etcher application. The SD card can be either inserted into the sd card slot of the Linux host machine or it can be inserted using the USB card reader. Etcher application will detect the device node allocated for the SD card by itself. Any Linux distribution can be used for the following procedure.

Flashing the SD card image using Etcher

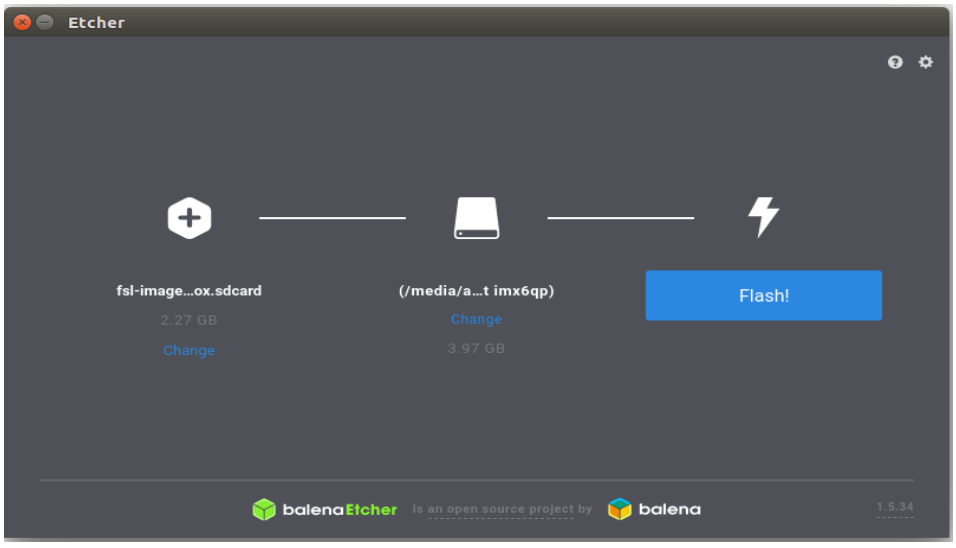
The SD card image (with the extension. sdcard) contains U-Boot, the Linux image and device trees, and the rootfs for a 4 GB SD card. The image can be flashed into the SD card using the Etcher application.

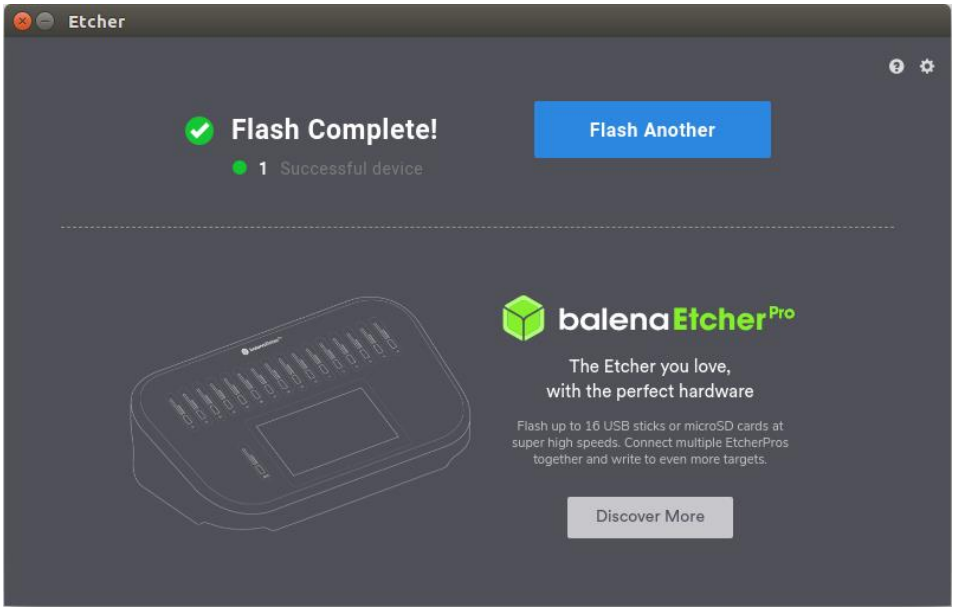
Download and install the Etcher application in the Linux host machine. The Etcher application is quite easy to use. Flashing using the Etcher requires 2 simple steps.

1) Select the SD card image to flash



2) Select Flash to download the image into SD card





After the flashing, the SD card will get partitioned and the kernel image and DTB will be present in the FAT partition and rootfs will be present in the EXT4 partition. To see the partitions in the SD Card, remove and re-insert the SD Card. Etcher will automatically unmount the SD Card after flashing.

Now the SD Card which contain all the necessary elements to boot can be inserted into the SD3 slot of board and is ready to boot up. This way, the default SD Card package is ready to use. To add the WILC support the following steps needs to be done.

Adding WILC Driver Support for Specific Kernel Version

This section will explain how to build the kernel image and supporting modules to include the WILC driver support for the specific kernel version. Built the zImage and dtb file as per the below instruction and replace the existing zImage and dtb(imx6qp-sabresd.dtb) file in the FAT partition of the SD Card with the latest build zImage and dtb file.

Building the Kernel and device tree file

- Export the toolchain path into the newly opened terminal.
- Configure the kernel with the defconfig file is located at arch/arm/configs

```
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-  imx_v6_v7_defconfig
```

- Replace the existing driver with the latest driver at driver/staging/wilc1000 folder
  - Download the latest ATWILC driver from the repository https://github.com/linux4wilc/driver
  - In the kernel, replace the existing wilc1000 driver directory in `drivers/staging/wilc1000` with the downloaded ATWILC driver.
- In driver/staging/Makefile file, change the CONFIG_WILC1000 to CONFIG_WILC
- Open menuconfig using the command
  make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- menuconfig
  and Modularize('M') the WILC_SDIO under Device Drivers/Staging drivers/ and save the settings.
- Build the kernel zImage:
  make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- zImage
  kernel image will be present at arch/arm/boot directory
- To built the dtb file:
  make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- imx6qp-sabresd.dtb
  dtb file will be present at arch/arm/boot/dts directoy

NOTE 1: Procedure mentioned in this document is based on the test done on the kernel taken from the kernel.org website.

Building the Modules

Initialization of the WILC module requires the module file (wilc-sdio.ko).

Build the modules, using the following command:

```
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- modules
```

Preparing the SD/MMC card to boot

Copy the WILC module file(wilc-sdio.ko)

- Inset the SD Card into the linux host machine.
- Copy the module file wilc-sdio.ko in the EXT4 partition(root) of the SD Card.

Replacing the zImage and dtb

- Replace the existing zImage and dtb file in the FAT partition of the SD Card with the latest build zImage and dtb file.

NOTE 2: After replacing the existing zImage and dtb file with the latest build kernel image and dtb, the u-boot configuration needs to be re-configured to select the appropriate block.

On i.MX 6 SABRE boards, you can boot the system through the rootfs in the SD card by the HannStar LVDS. The kernel MMC module now uses a fixed mmcblk index for the uSDHC slot.

U-Boot Configuration

After the replacement of the existing zImage and dtb file, changes need to be made to the default U-Boot environment to select the appropriate mmcblock. For this changes to get reflected in the U-Boot, the U-Boot image need to be re-built by re-configuring the MMCROOT variable. The existing U-Boot image then needs to be replaced with the newly build U-Boot image.

1) Clone the U-Boot for imx

git clone https://source.codeaurora.org/external/imx/uboot-imx -b imx_v2017.03_4.9.51_imx8_beta2

2) Edit the U-Boot configuration file for imx6 SABRE board

Open the mx6sabresd.h file located at include/configs directory

The default configuration will be:

#define CONFIG_MMCROOT "/dev/mmcblk2p2" /* SDHC3 */

Change the the mmcblk to 1 instead of 2. The final change will look like

#define CONFIG_MMCROOT "/dev/mmcblk1p2" /* SDHC3 */

3) export CROSS_COMPILE=arm-linux-gnueabi-

4) export ARCH=arm

5) Configure the U-Boot for our board

make mx6qpsabresd_defconfig

6) make

This will create a u-boot image u-boot-dtb.imx. The existing U-Boot image in the SD Card need to be replaced with this image

7) Flashing the U-Boot to SD Card.

To flash the latest build U-Boot image to SD Card, we use dd command.

The Linux kernel running on the Linux host machine assigns a device node to the SD/MMC card. To identify the device node assigned to the SD/MMC card, carry out the following command:

```
$ cat /proc/partitions
major minor #blocks   name
8     0     78125000  sda
8     1     75095811  sda1
8     2     1         sda2
8     5     3028221   sda5
8     32    488386584 sdc
8     33    488386552 sdc1
8     16    3921920   sdb
8     18    3905535   sdb1
```

In this example, the device node assigned is /dev/sdb (a block is 512 Bytes). In this case, we used USB card reader to insert the SD Card in to the Linux host machine.

NOTE

- ✓ Check the exact device node assigned by executing the command before and after inserting the SD card. Thus, make sure that the device node is correct for the SD/MMC card. Otherwise, it may damage your operating system or data on your computer.
- ✓ The above result is for the SD card inserted to a USB stick adaptor.

Command to flash the U-Boot image

dd if=<boot_image> of=/dev/sd<x> bs=1k seek=<offset> conv=fsync

-- U-Boot Image: u-boot-dtb.imx
-- Offset is 1 - for i.MX 6 or i.MX 7
-- sd<x> is: Device node for the SD card

NOTE: Failure to update the U-Boot environment will result in kernel panic error message.

Loading Firmware

- The default rootfs doesn't contain the wilc firmware. In order to download the firmware into the wilc device during the wilc initialization, firmware files need to be placed into specific directory in rootfs. Rootfs is present in the EXT4 partition of SD Card.
- Create a directory named "mchp" under lib/firmware directory in rootfs.
- Download the latest firmware from https://github.com/linux4wilc/firmware and copy the 'wilc1000_wifi_firmware.bin' file inside mchp directory

Setting up the System

1) Insert the SD Card
   Insert the SD card into the socket SD3
2) Connect the micro-B end of the USB cable into the debug port J509 to receive the debug log messages. Connect the other end of the cable to a PC acting as a host terminal. If needed, the Serial-to-USB drivers can be found at
   https://www.ftdichip.com/FTDrivers.htm

   On Linux host machine, run the following command to determine the Serial Device:
   `$ ls /dev/ttyUSB*`

   Terminal window configuration: 115.2 kbaud, 8 data bits, 1 stop bit, no parity
3) Connect power supply
   Connect the 5 V power supply cable to the 5 V DC power jack P1. When power is connected to the smart device, it will automatically begin the boot sequence.
4) Login details
   Once the Linux is booted you can login with:
   Username: root
   Password: no password

Initialization of WILC Module

- Move to root

  cd /

  ls command will list the available files in rootfs. Previously placed wilc-sdio.ko file will be present here.

- Insert the WILC1000 Device into the SD2 card slot of the Board.

  NOTE: The testing carried out for this document uses the WILC1000 module.

- Load the module using the command:
  insmod wilc-sdio.ko
  WILC driver will be loaded successfully.

Configuring WILC Module as Station and Connecting to AP

By default, the wilc wpa supplicant file won't be available in the rootfs. In order to initialize the supplicant, the supplicant file need to be created and content needs to be added.

- Create the file inside the /etc using the vi command vi /etc/wilc_wpa_supplicant.conf
- Add the below content into the file ctrl_interface=/var/run/wpa_supplicant
  update_config=1
  ap_scan=1

- Kill the existing supplicant by the below command
  Killall wpa_supplicant

  By default, an instance of wpa_supplicant process will be already running in background. First we need to kill this process before we initialize the supplicant. Keeping this process as alive and initializing another instance of wpa_supplicant will create trouble in initiating DHCP.

- Initializing the supplicant
  Wpa_supplicant -iwlan0 -Dnl80211 -c /etc/wilc_wpa_supplicant.conf -B

- Connecting to a AP using following command
  1) wpa_cli -p /var/run/wpa_supplicant ap_scan 1
  2) wpa_cli -p /var/run/wpa_supplicant add network
  3) wpa_cli -p /var/run/wpa_supplicant set_network 0 ssid '"DEMO_LINUX"'

4) wpa_cli -p /var/run/wpa_supplicant set_network 0 key_mgmt WPA-PSK
5) wpa_cli -p /var/run/wpa_supplicant set_network 0 psk '"12345678"'
6) wpa_cli -p /var/run/wpa_supplicant select_network 0
7) udhcpc -iwlan0