Software Developer Guide
SPB209A
Application Note

Table of Content

1 Prepare host platforms for SPB209A operation

1.1 Using evaluation platform i.MX6

1.1.1 Install the development environment for i.MX6 evaluation boards using Ubuntu

1.1.1.1 Prerequisite

- A host system with minimum 50Gb free disk space running Ubuntu 14.04
- Freescale release layer and Yocto project community layers
- Agreement to NXP/Freescale End User License Agreement using the Freescale Yocto Project Community BSP
- For later stage an iMX6 evaluation board and SPB209A evaluation board

1.1.1.2 Features

The Freescale Yocto Project Release have the following features:

- Linux kernel recipe
- U-Boot recipe
- Graphics recipes
- i.MX package recipes
- Core recipes
- Demo recipes

1.1.1.3 Set-up the Host development environment

This guide will briefly take you through a set-up of the build environment for an iMX6 platform. More details is available in NXP/Freescale documentation, see ref 1 and Yocto Project Page, see ref 2.

1. Install Essential and Graphical Yocto project host packages

```
$ sudo apt-get install gawk wget git-core diffstat unzip texinfo gcc-multilib \build-essential
chrpath socat libsdl1.2-dev
```

2. Install i.MX layers host packages for Ubuntu 14.04 host set-up

```
$ sudo apt-get install libsdl1.2-dev xterm sed cvs subversion coreutils texi2html \docbook-utils
python-pysqlite2 help2man make gcc g++ desktop-file-utils \libgl1-mesa-dev libglu1-mesa-dev
mercurial autoconf automake groff curl lzop asciidoc
```

3. Install i.MX u-boot tools

```
$ sudo apt-get install u-boot-tools
```

4. Set-up the repo utility (support utility to manage projects that contain multiple repositories)

```
$ mkdir ~/bin (this step may not be needed if the bin folder already exists)
$ curl http://commondatastorage.googleapis.com/git-repo-downloads/repo > ~/bin/repo
$ chmod a+x ~/bin/repo
```

Add the following line to the .bashrc file to ensure that the ~/bin folder is in your PATH variable.

export PATH=~/bin:$PATH

5. Set-up Git properly

```
$ git config --global user.name "Your Name"
$ git config --global user.email "Your Email"
$ git config --list
```

6. Download the Freescale Yocto Project Community BSP recipe layers

```
$ mkdir fsl-release-bsp
$ cd fsl-release-bsp
$ repo init -u git://git.freescale.com/imx/fsl-arm-yocto-bsp.git -b imx-4.1.15-1.0.0_ga
$ repo sync
```

1.1.2 Build an Image for i.MX6 evaluation boards using Ubuntu

1.1.2.1 Build configurations

A build configuration is generated into a distro file (local.conf) using a script fsl-set-up-release.sh with appropriate entered configuration defines

1. Generated distro file

```
$ DISTRO=<distro name> MACHINE=<machine name> source fsl-setup-release.sh -b <build dir>
```

<distro name> : Freescale distro configurations

- fsl-imx-x11 - Only X11 graphics (Used for SPB209A Demo)
- fsl-imx-wayland - Wayland weston graphics
- fsl-imx-xwayland - Wayland graphics and X11. X11 applications using EGL are not supported
- fsl-imx-fb - Frame Buffer graphics - no X11 or Wayland

<machine name> : Freescale evaluation board type

- imx6qpsabreauto
- imx6qpsabresd
- imx6ulevk
- imx6dlsabreauto
- imx6dlsabresd (Used for SPB209A Demo)
- imx6qsabreauto
- imx6qsabresd
- imx6slevk

- imx6solosabreauto
- imx6solosabresd
- imx6sxsabresd
- imx6sxsabreauto
- imx7dsabresd

-b <build dir> : specifies build directory name (name will be created by the script)

1.1.2.2 Execute SPB209A Yocto Layer adding wifi/Bluetooth/nfc drivers

If provided i.MX6 image for SPB209A or other agreed image isn't sufficient, obtain the SPB209A Yocto Layer for H&D Wireless Sales by signing a software license agreement.

1. Unzip the package to a directory of choice.
2. Modify the bblayers.conf file, adding the meta-hdw layer, see shadowed line below.

```
LCONF_VERSION = "6"

BBPATH = "${TOPDIR}"
BSPDIR := "${@os.path.abspath(os.path.dirname(d.getVar('FILE', True)) + '/../..')}"

BBFILES ?= ""
BBLAYERS = " \
 ${BSPDIR}/sources/poky/meta \
 ${BSPDIR}/sources/poky/meta-yocto \
 \
 ${BSPDIR}/sources/meta-openembedded/meta-oe \
 ${BSPDIR}/sources/meta-openembedded/meta-multimedia \
 \
 ${BSPDIR}/sources/meta-fsl-arm \
 ${BSPDIR}/sources/meta-fsl-arm-extra \
 ${BSPDIR}/sources/meta-fsl-demos \
 ${BSPDIR}/sources/meta-hdw \
"
##Freescale Yocto Project Release layer
BBLAYERS += " ${BSPDIR}/sources/meta-fsl-bsp-release/imx/meta-bsp "
BBLAYERS += " ${BSPDIR}/sources/meta-fsl-bsp-release/imx/meta-sdk "
BBLAYERS += " ${BSPDIR}/sources/meta-browser "
BBLAYERS += " ${BSPDIR}/sources/meta-openembedded/meta-gnome "
BBLAYERS += " ${BSPDIR}/sources/meta-openembedded/meta-networking "
BBLAYERS += " ${BSPDIR}/sources/meta-openembedded/meta-python "
BBLAYERS += " ${BSPDIR}/sources/meta-openembedded/meta-ruby "
BBLAYERS += " ${BSPDIR}/sources/meta-openembedded/meta-filesystems "
BBLAYERS += " ${BSPDIR}/sources/meta-openembedded/meta-xfce "
BBLAYERS += " ${BSPDIR}/sources/meta-qt5 "
```

3. Select what driver package to use by editing the ../conf/bblayers.conf as follows (The hdw yocto layer supports two driver packages; The driver provided with the linux distribution and the full featured driver (the mlan driver)):

a: To select the Linux default distribution driver:

```
Comment out the line "IMAGE_INSTALL_append = " hdw-firmware-mlan""
```

b: To select the full featured (mlan) driver:

```
Comment out the line "IMAGE_INSTALL_append = " hdw-firmware-mwifiex""
```

4. Note that the ../conf/bblayers.conf also contains a line for including the hdw wifi/bluetooth/nfc demo and a line for including the hostapd package.

   If you don't want to include the demo, comment out the line:

```
IMAGE_INSTALL_append = " hdw-demo"
```

   If you don't want to include the hostapd package, comment out the line:

```
IMAGE_INSTALL_append = " hostapd"
```

5. Apply the meta-hdw layer for the SPB209A wifi/bluetooth/nfc drivers (assuming current directory is ../build):

```
$ cat BBLAYERS += " ${BSPDIR}/sources/meta-hdw " >> ../conf/bblayers.conf
```

6. Add Bluetooth audio capabilities:

```
$ cat DISTRO_FEATURES_append = " bluetooth bluez5 pulseaudio" >>
../build/conf/local.conf
```

Build the image:

```
$ bitbake fsl-image-qt5
```

Now there is a complete image available. Load it into sd flash card:

```
$ sudo dd if=../build/tmp/deploy/images/imx6dlsabresd/fsl-image-qt5-
imx6dlsabresd.sdcard of=/dev/mmcblk0
```

1.1.2.3 Build an image

The following commands builds a complete image defined in the chosen <image type> (fsl-image-qt5 used for SPB209A Demo). This will take time as it builds the complete kernel and tools. Depending on how you run your linux host or if you need debug info you might want to add <parameter>, see table below.

```
$ bitbake <parameter> <image type>
```

-c fetch	Fetches if the downloads state is not marked as done.
-c cleanall	Cleans the entire component build directory. All the changes in the build directory is lost.
-c deploy	Deploys an image or component to the rootfs.
-k	Continues building components even if a build break occurs.
-c compile -f	It is not recommended that the source code under the tmp directory is changed directly,
-g	Lists a dependency tree for an image or component.
-DDD	Turns on debug 3 levels deep. Each D adds another level of debug.

Validate that the i.MX6DL image is available under the image directory.

```
$ cd ~/home/fsl-release-bsp/build/tmp/deploy/images/imx6dlsabresd/
```

1.1.3 Deploy an i.MX6 image to the evaluation hardware

The following i.MX6 software is required to be able to boot the i.MX6 evaluation board and run the Linux operating system:

- Bootloader (U-Boot)
- Linux kernel image (zImage)
- A device tree file (.dtb) for the i.MX6 evaluation board being used
- A Linux root file system (rootfs) for the particular Linux image

The software can be booted from different medias, such as NOR, NAND, SATA ARM Core, QSPI, EMMC, m4fastup or epdc. Default is boot from SD card which will be used in this guide. To config for other boot option please read more about it in ref 1.

1.1.3.1 Flash i.MX6 image to SD card

There is two ways of placing an image on a device, one way is to use the MFGTool and the other way is to Flash a SD card directly using Linux dd command. This user guide will use the latter. The MFGTool procedure is discribed more in ref 1.

1.1.3.1.1 Prepare the SD card using Windows DiskPart

To install an i.MX6 image containing two partitions (u-boot and rootfs) on a SD card a proper SD card prepare is needed. A flow using Windows DiskPart works well, to clean a formatted SD card with minimum size of 8Gbyte and create a new primary partition. There is probably other ways to get to the same result.

```
DISKPART> list disk
DISKPART> select disk X
DISKPART> list part
DISKPART> clean
DISKPART> list part
DISKPART> create partition primary
```

X: SD card disk (be careful with this selection as we will clean it in coming commands)

Eject the SD card from the windows computer and insert it into the Ubuntu host for continued operation to flash an i.MX6 image.

1.1.3.1.2 Flash the SD card

On the Linux Host run the following command to write the image to the SD card

```
$ sudo dd if=<image name>.sdcard of=/dev/sd<partition> bs=1M && sync
```

<image name> : The image built in earlier steps is available under <build directory>/tmp/deploy/images. (file should be .sdcard at the end)

sd<partition> : check where the SD card is connected by using the "lsblk" command. If sda enter sda, if mmcblk0 enter mmcblk0

The below examples is used for SPB209A Demo:

```
$ sudo dd if=../build/tmp/deploy/images/imx6dlsabresd/fsl-image-qt5-imx6dlsabresd.sdcard of=/dev/mmcblk0
```

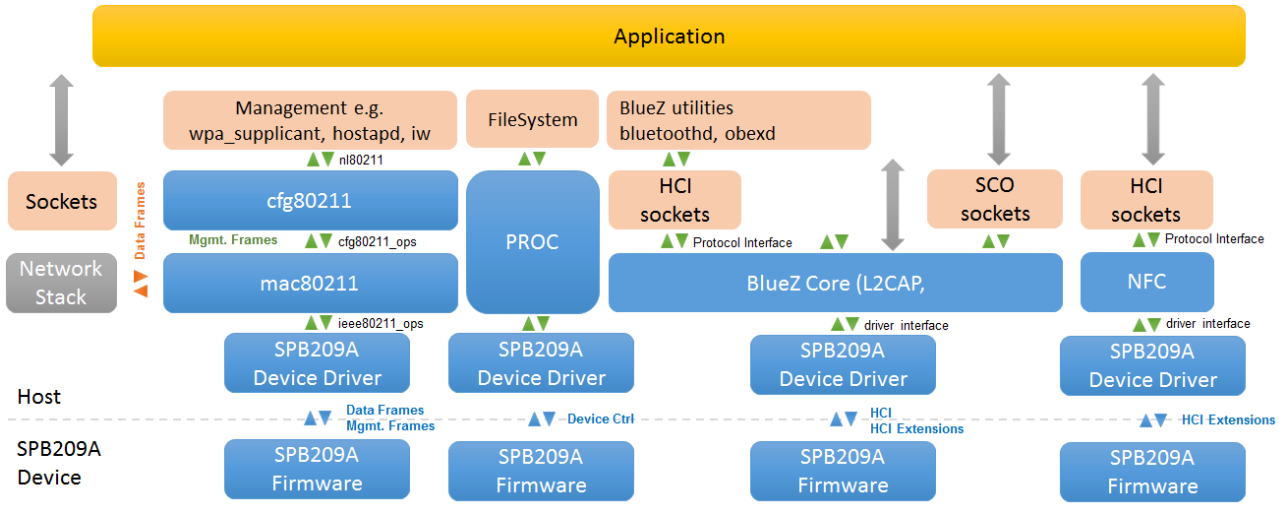
1.1.4 Operate an i.MX6 Evaluation Board

Insert the i.MX6 image flashed SD card into the i.MX6 evaluation board SD slot. Set up the DIP switches on the board to define which device the system shall boot from, in this case an SD slot. Follow instruction available under ref 1.

Connecting appropriate network, debug interfaces, image SD card and plugging in the power. This will start the system.

To evaluate SPB209A follow the SPB209A Qucik Start Guide 1543 SPB209A_Rev._EVB_Quick_Start_Guide.pdf on linux.hd-wireless.se site.

2 Software Control Interface



The Linux system operation is divide into two distinct virtual memory operation regions namely - the kernel space and the user space.

The *kernel space* operate core of the operating system and control access to physical devices on the computer. It also schedule when and how processes interact with these devices.

The *user space* operate applications and functions outside the operating system's kernel and use the predefined system calls to access kernel space functions.

The SPB209A can be controlled from user space by an application using the following software interfaces:

- Interfacing to wpa_supplicant and hostapd controlling the SPB209A WiFi functionality
- Interfacing BlueZ controlling the SPB209A Bluetooth
- Interfacing NFC controlling the SPB209A NFC controller
- Interfacing GPIO using HCI vendor specific commands (PROC) and FW API

2.1 Interfacing wpa_supplicant, hostapd

2.1.1 Wpa_supplicant

The wpa_supplicant is the IEEE 802.1X/WPA component used in the client stations. The WPA supplicant can be configured to control the roaming and IEEE 802.11 authentication/association of the SPB209A device.

The configuration is usually performed in a configuration file, e.g. /etc/wpa_supplicant.conf. It is also possible to directly issue commands to the WPA Supplicant, using a dedicated shell command, wpa_cli. The usage of wpa_cli is out of the scope of this document, but is described in detail in the WPA supplicant documentation http://hostap.epitest.fi/wpa_supplicant/.

Below list show supported WPA Supplicant network options

- Key management (key_mgmt): WPA-PSK, NONE
- Group key encryption (group): CCMP, TKIP
- Pairwise key encryption (pairwise): CCMP, TKIP
- Protocol (proto): WPA, WPA2

Below list show examples of instructions on how to perform the following operations using WPA Supplicant, details is outlined in ref 3.

- Connect to an unencrypted network
- Connect to a WPA protected network that uses TKIP encryption
- Connect to a WPA2 enabled network that uses CCMP encryption
- Connect to a network that uses any WPA/WPA2 protocol and TKIP/CCMP encryption.
- Connect to a network with hidden SSID
- List of supported WPA Supplicant network options

2.1.2 Hostapd

Hostapd is a user space daemon for access point and authentication servers, and can be used for SPB209A access point management, see further details under ref5.

2.1.2.1 Linux Driver operating in AP mode (hostapd)

Currently there is only hw support for the Linux driver with SDIO interface for the SPB209A.

Linux kernel version 3.19 or higher is required. If DFS in AP mode is to be used, **Linux kernel version should be at least 4.0**.

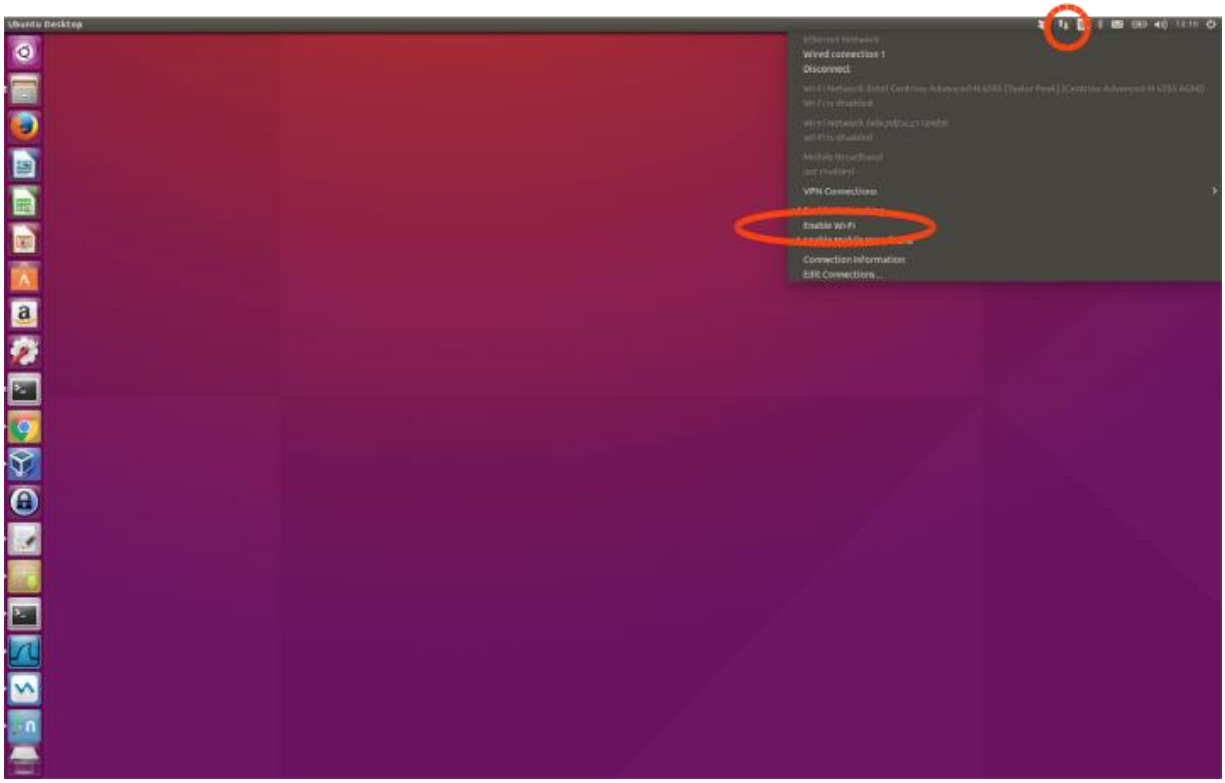
The Linux driver consist of four kernel objects: mwifiex.ko, mwifiex_sdio.ko, btmrvl.ko and btmrvl_sdio.ko.

mwifiex.ko and mwifiex_sdio.ko handles the wifi protocols, while btmrvl.ko and btmrvl_sdio.ko handles Bluetooth, BLE and NFC.

Along with the driver goes a fw binary that is downloaded to the chip by the driver. It must be named sd8887_uapsta.bin and located at /lib/firmware/mrvl/

HOWTO run Linux softAP with hostapd

1. Make sure the network manager is disabled with regards to wifi:



2. Make sure the radio interface is unblocked:

```
3. rfkill unblock all
```

3. Download the attachment:hostapd.conf file to the local disk.

4. Install Linux Wifi host AP package:

```
5. sudo apt-get install hostapd
```

5. Plugin the sdio module.
   Make sure mwifiex driver was successfully started by typing:

```
6. iwconfig mlan0
```

This command should display information about the mlan0 interface

6. mwifiex driver does not support ap mode on native interface, so an additional ap-dedicated interface must be created.
   In order to do so we need to find out the phy<n> enum for the mlan0 interface by typing:

```
7. iw list | grep Wiphy
```

Normally phy0 corresponds to builtin wlan0, and the next higher enum will correspond to mlan0

7. Now create the ap specific interface (uap0) by typing

```
8. sudo iw phy phy<n> interface add uap0 type __ap
```

   Where <n> is the enum found out from iw list command

8. Now configure the AP by editing the hostapd.conf file.

Example: For 11n, 5GHz band, channel 36, DFS enabled: Search for and edit the follwing parameters in hostapd.conf file:

```
interface=uap0

ssid=<desired-ssid>

hw_mode=a

channel=<desired channel>

wmm_enabled=1

ieee80211n=1

ieee80211d=1

ieee80211h=1

country_code=<country_code>


Valid <country_code>'s are:

  US       # US FCC

  CA       # IC Canada

  EU       # ETSI

  ES       # Spain

  FR       # France

  JP       # Japan
```

```
CN      # China
```

9. Start the AP by typing:

```
10.     sudo hostapd <hostapd_config_file_name>
```

AP should now be up an running

10. To run traffic, assign a fixed ip address to the interface:

```
11.     sudo ifconfig uap0 <dersired ip> #e.g. 192.168.10.1
```

12. Associate a station and assign a static ip at the same subnet

12. To remove the uap0 interface, kill the hostapd process and run:

```
13.     sudo iw dev uap0 del
```

2.2 Interfacing BlueZ

The BlueZ Bluetooth stack is an open source Linux Bluetooth stack. It provides support for the core Bluetooth layers and protocols. See further details in ref 4.

2.2.1 Supported profiles/features

The currently supported profiles/features are as follows:

```
Profile/protocol Version          Role(s)
-----------------------------------------------------------------

GAP              4.2         (LE) Central, Peripheral,
                                  Observer, Broadcaster
L2CAP            4.2         Server, Client
SDP              4.2         Server, Client
GATT             4.2         Server, Client
SDAP             1.1         Server, Client
RFCOMM           1.1         Server, Client
SPP              1.1         Server, Client

PXP              1.0         Reporter, Monitor
HOGP             1.0         Host
HTP              1.0
TIP              1.0
CSCP             1.0         Collector

SAP              1.1         Server
DUN              1.1         Server, Client

DID              1.3         Server, Client

HFP              1.6         AG, HF
HSP              1.2         AG, HS
```

```
GAVDTP          1.2         Source, Sink
AVDTP           1.3         Source, Sink
A2DP            1.3         Source, Sink
AVCTP           1.3         CT, TG
AVRCP           1.5         CT, TG

GOEP            2.0         Client, Server
FTP             1.2         Client, Server
OPP             1.2         Client, Server
SYNCH           1.1         Client
PBAP            1.1         Client, Server
MAP             1.0         Client, Server

HID             1.1         Host

BNEP            1.0
PAN             1.0         PANU, NAP, GN

HCRP            1.2

MCAP            1.0
HDP             1.0
```

It should be noted that some profiles/roles will depend on external components such as oFono or ConnMan.

2.2.2 BlueZ Utilities

The BlueZ utilities (utils) is a tool-set that helps to manage the Bluetooth devices for test and debug. The tools / commands available is outlined below.

Tool	Function
Ciptool	ciptool is used to set up, maintain, and inspect the CIP configuration of the Bluetooth subsystem in the Linux kernel.
Dund	BlueZ Bluetooth dial-up networking daemon
Hcitool	hcitool is used to configure Bluetooth connections and send some special command to Bluetooth devices.
Hidd	Bluetooth HID daemon
I2ping	Attempts to create a connection to a device using logical link control and adaptation protocol (L2CAP)
Pand	Connection with Personal Area Network
rfcomm	rfcomm is used to set up, maintain, and inspect the RFCOMM configuration of the Bluetooth subsystem in the Linux kernel.
sdptool	sdptool provides the interface for performing SDP queries on Bluetooth devices, and administering a local sdpd.

2.2.3 BlueZ API

A custom Linux application could use the command line tools described in section 2.2.2, however a more integrated solution will be to use the available API. There have been a major architectural change between version 4 and version 5 release of the BlueZ stack.

The earlier version 4 release use C API defined under Ref 9 and version 5 introduces the D-BUS API. The benefit with D-BUS API is that it is much easier to work with than the version 4 C API or wrapping the command line tools.

The Linux Org promoted way to work with BlueZ is to use the D-Bus API. And since it is standard D-Bus interface, you can use any programming language with its d-bus binding library, see API documentation in ref6 and D-BUS specification ref7.

As a complement to the D-BUS API documentation there are some python example test scripts under ref10 that could be useful.

2.2.4 SPB209A Vendor Specific Commands

SPB209A support most of the standard HCI commands and is operated through the BlueZ API. The following vendor specific commands can be used:

OCF	HCI Command	Description
0x0006	SET_SCO_CHUNK_SIZE	Controls length of voice packet over HCI interface
0x0007	WRITE_PCM_SETTINGS	Writes PCM settings
0x0009	UART_BAUD	Sets UART baud rate
0x001D	SET_SCO_DATA_PATH	Sets SCO data path
0x0023	SET_BT_SLEEP_MODE	Sets Bluetooth sleep mode information
0x0028	WRITE_PCM_SYNC_SETTINGS	Writes PCM synchronization settings
0x0029	WRITE_PCM_LINK_SETTINGS	Write PCM link settings
0x0053	WAKE_UP_METHOD	Sets wake-up method information for Bluetooth over UART operation
0x005D	HIU_MODULE_CONFIGURATION	Sets host interface module configuration
0x005D	SET_AFH_CHANNEL_CLASSIFICATION_MODE	Sets AFH channel classification mode
0x0068	SET_ED_OPTIONS	Controls options for enabling PPEC and padding of voice data in case of errors
0x006B	UCD_CONFIG	Sets UCD configuration
0x006F	HOST_PCM_CONFIG	Initializes and configures PCM
0x0070	HOST_PCM_CONTROL_ENABLE	Controls PCN lines
0x0073	SET_WBS_CONNECTION	Sets WBS connection
0x279	SBC_INITIALIZATION	Performs a SBC initialization
0x279	SBC_DEINITIALIZATION	Performs a SBC de-initialization
0x279	SBC_SET_CONFIGURE	Sets the SBC configuration
0x279	SBC_GET_CONFIGURE	Gets the SBC configuration
0x279	SBC_MEDIA_PACKET_HEADER	Sets the SBC packet header
0x279	SBC_CONTROL	Sets the SBC control
0x279	SBC_PROTOCOL_PARAMS	Sets the SBC protocol parameters
0x279	SBC_CONTENT_PROTECTION	Sets the content protection p<rt of SBC packet header
	SBC Event	Event for SBC offloading command
0x008D	BLE_POWER_SAVE_MODE	Sets/gets the BLE power save mode configuration

2.2.4.1 SET_SCO_CHUNK_SIZE

OCF	0x0006		
OGF	0x3F		
CMD Param.	Name	Length	Description
	SCO Package Size	1	SCO packet size used over HCI interface. This can have a value of 60, 120, 180 and 240 bytes. Any other value will return error status of invalid common parameters. It is recommended to keep the parameter to 120 bytes for optimal performance. Default = 120.
Return Param.	Status (1 octet) 0x00 = success 0x01 to 0xFF = error		
Events	Command complete event returned		
Note	This command shall be given before the standard HCI command to read the controller buffer size.		

2.2.4.2 WRITE_PCM_SETTINGS

OCF	0x0007		
OGF	0x3F		
CMD Param.	Name	Length	Description
	PCM_Setting	1	Bit[4]: PCM Clock On 0 = PCM clock is terminated after last data bit has been transmitted 1 = make PCM clock available continuously Bit[3]: Reserved Bit[2]: PCM Sync Source 0 = PCM sync page generated from system clock 1 = PCM sync page generated from frame clock Bit[1]: Master/Slave 0 = PCM I/F slave, external PCM clock synchronization 1 = PCM I/F master, internal PCM clock and synchronization Bit[0]: PCM Direction 0 = port A receive, port B transmit 1 = port A transmit, port B receive
Return Param.	Status (1 octet) 0x00 = success 0x01 to 0xFF = error		
Events	Command complete event returned		
Note			

2.2.4.3 UART_BAUD

OCF	0x0009		
OGF	0x3F		
	Name	Length	Description

CMD Param.	Baud Rate	4	New baud rate to be programmed 9600 19200 38400 57600 115200 230400 460800 500000 921600 1000000 1382400 1500000 1843200 2000000 2100000 2764800 3000000 3250000 2692300 4000000
Return Param.	Status (1 octet) 0x00 = success 0x01 to 0xFF = error		
Events	Command complete event returned at the old baud rate		
Note	Host can switch to the new baud rate after receiving the complete event. The Host shall wait for 5ms or more before sending any new command.		

2.2.4.4 SET_SCO_DATA_PATH

OCF	0x001D		
OGF	0x3F		
CMD Param.	Name	Length	Description
	Voice Path	1	0x00 = Host 0x01 = PCM
Return Param.	Status (1 octet) 0x00 = success 0x01 to 0xFF = error		
Events	Command complete event returned		
Note			

2.2.4.5 SET_BT_SLEEP_MODE

OCF	0x0023		
OGF	0x3F		
CMD Param.	Name	Length	Description
	Power Mode	1	0x02 = sleep mode 0x03 = full power mode
	Inactivity Timeout	2	Timeout

			Two byte attribute to define an idle time to enter power save. The card will send PS_SLEEP to the host only if there is no communication between host and card for the specified interval in this parameter. Valid value for this parameter is from 0x0000 to 0xFFFF. Value 0x0000 implies no idle timeout and PS_SLEEP will be sent immediately right after the last Tx/Rx is done to/from host.
Return Param.	Status (1 octet) 0x00 = success 0x01 to 0xFF = error		
Events	Command complete event returned		
Note			

2.2.4.6 WRITE_PCM_SYNC_SETTINGS

OCF	0x0028		
OGF	0x3F		
CMD Param.	Name	Length	Description
	PCM Sync Settings 1	1	Default = 0x03
	PCM Sync Settings 2	2	Default = 0x0000
	The PCM Sync Setting 1 is according to the following: ISR (IramSyncRate) only valid if IF = Host: PCM Sync Settings 1 Bit[0]: 0 = bursts controlled by Tx or Rx of voice packets 1 = fixed rate of 8 ksamples/s ISS (IramSyncSource) only valid if IF = Host and ISR = Fixed Rate: PCM Sync Settings 1 Bit[1]: 0 = ISR not aligned to frame tick 1 = ISR aligned to frame tick (this field should be set to 1) THE PCM Sync Settings 2 is according to the following: pcmIfMode in PCM Descriptor PCM Sync Settings 2 Bits[1:0]: 00 = PCM short sync 01 = PCM long sync 10 = I2S audio mode pcmLRCPol in PCM Descriptor PCM Sync Settings 2 Bit[4]: 0 = LRC is same polarity as PCM sync 1 = LRC is inverted pcmMClkEn in PCM Descriptor PCM Sync Settings 2 Bit[8]: 0 = disable generation of PCM main clock 1 = enable pcm2048MClkSel in PCM Descriptor PCM Sync Settings 2 Bit[9]: 0 = default 1 = select 2.048 MHz clock for PCM		

	16k Sync in PCM PCM Sync Settings 2 Bit[10]: 0 = 8k Sync 1 = 16k Sync
Return Param.	Status (1 octet) 0x00 = success 0x01 to 0xFF = error
Events	Command complete event returned
Note	

2.2.4.7 WRITE_PCM_LINK_SETTINGS

OCF	0x0029		
OGF	0x3F		
CMD Param.	Name	Length	Description
	PCM Link Setting	2	Bits[13:10]: Each bit corresponds to 1 of 4 PCM time slots (if 0, the slot is used by the BTU) Bits[9:2]: Defines start of PCM slot relative to start of PCM synchronization (must be greater than the size of the PCM slot) Bits[1:0]: indicates which PCM slots should be used Default: 0x0004 = first SCO link 0x0045 = second SCO link 0x0086 = third SCO link
Return Param.	Status (1 octet) 0x00 = success 0x01 to 0xFF = error		
Events	Command complete event returned		
Note	PCM Link settings command should be given after HCI reset and before setting up the voice link. Also, if multiple voice links are supported, this command should be given before setting up the voice link with parameters appropriate for each voice link.		

2.2.4.8 WAKE_UP_METHOD

OCF	0x0053		
OGF	0x3F		
CMD Param.	Name	Length	Description
	Host Wake Up Method	1	0x00 = host wake up no need 0x01 = host wake up by DTR 0x02 = host wake up by Break
	Host Wake Up GPIO	1	This value is ignored by the chip
	Device Wake Up Method	1	0x00 = host wake up by DSR 0x01 = host wake up by Break
	Device Wake Up GPIO	1	This value is ignored by the chip
Return Param.	Status (1 octet) 0x00 = success 0x01 to 0xFF = error		

Events	Command complete event returned
Note	

2.2.4.9 HIU_MODULE_CONFIGURATION

OCF	0x005B		
OGF	0x3F		
CMD Param.	Name	Length	Description
	WLAN/Bluetooth Module Shutdown/Bring Up	1	0xF1 = bring up Bluetooth module (host interface keeps up) 0xF2 = shut off Bluetooth module (host interface keeps up)
	Host Interface Activity	1	--
	Host Interface Type	1	--
Return Param.	Status (1 octet) 0x00 = success 0x01 to 0xFF = error		
Events	Command complete event returned		
Note			

2.2.4.10 SET_AFH_CHANNEL_CLASSIFICATION_MODE

OCF	0x005D		
OGF	0x3F		
CMD Param.	Name	Length	Description
	Enable Master Channel Classification	1	Enables the use of AFH channel reports from slave 0x00 = disable 0x01 = enable (default)
	AFH Slave Classification Reporting Minimum Interval	2	In 625 µs slots, between 0x0640 to 0xBB80 (1 to 30s) Default = 0x1F40
	AFH Slave Classification Reporting Maximum Interval	2	In 625 µs slots, between 0x0640 to 0xBB80 (1 to 30s) Default = 0x2E80
	Channel Classification Algorithm	1	Method for Channel Classification 0x00 = packet loss ratio (PLR) only (default)
	Master Classification Alpha Weighting	1	0x00 = full weight on slaves (default) 0x01 = full weight on master
	Master Classification Master Bias	1	Weight of master's assessment decision Value = 0x00 to 0x10 Default = 0x0E
	Master Classification	1	If (slave report channel quality > threshold). Then channel is good

	Threshold Numerator		Default = 0x01
	Master Classification Threshold Denominator	1	If (slave report channel quality > threshold). Then channel is good Default = 0x02
	RSSI-based AFH Bias	2	Reserved
Return Param.	Status (1 octet) 0x00 = success 0x01 to 0xFF = error		
Events	Command complete event returned		
Note			

2.2.4.11 SET_ED_OPTIONS

OCF	0x0068		
OGF	0x3F		
CMD Param.	Name	Length	Description
	Options	1	Bit[3]: Disable padding of voice data when Erroneous Data Reporting is enabled (default = 0) Bit[2]: Do not force Burst Mode when Erroneous Data Reporting is enabled (default = 0) Bit[1]: Enable PPEC when Erroneous Data Reporting is enabled (default = 0) Bit[0]: Enable PPEC when Erroneous Data Reporting is disabled (default = 0)
Return Param.	Status (1 octet) 0x00 = success 0x01 to 0xFF = error		
Events	Command complete event returned		
Note	When Erroneous Data Reporting feature is enabled, it is recommended to use: - Disable PPEC (Bit[1] = 0) - Use burst mode (Bit[2] = 0) - Use transparent voice data rather than CVSD as the air encoding		

2.2.4.12 UCD_CONFIG

OCF	0x006B		
OGF	0x3F		
CMD Param.	Name	Length	Description
	Mode	1	0x00 = disable 0x01 = enable (forward packet to host after wakeup) 0x03 = enable (drop packet after wakeup) Others = reserved
	CID	2	0x0002 (reserved)
	PSM	2	PSM
	Payload Length	1	Maximum 16
	Payload Pattern	16	Payload pattern

Return Param.	Status (1 octet) 0x00 = success 0x01 to 0xFF = error
Events	Command complete event returned
Note	

2.2.4.13 HOST_PCM_CONFIG

This command initialize and configure PCM and shall be sent in any of the following situations:

- To initialize PCM after starting voice call on a particular SCO connection
- To switch call from SCO connection 1 to SCO connection 2
- To route SCO connection 1 voice data to SCO connection 2
- To de-initialize PCM once the voice call is over on a particular SCO connection

OCF	0x006F		
OGF	0x3F		
CMD Param.	Name	Length	Description
	Action	1	0x00 = PCM will be initialized 0x01 = PCM will be de-initialized
	Operation Mode	1	0x00=normal mode This mode is used when only 1 voice call needs to be active at a time. Depending on the Action parameter, either the PCM will be initialized or de-initialized. 0x01 = internal loopback This mode is used when there are 2 HF connections and audio data on first connection needs to be routed to second connection. This can be used only when both SCO links are of the same type (that is. NB to NB, or WB to WB only) 0x02 = remote loopback on same link 0x03 = local loopback on same link 0x04 to 0xFF = reserved
	SCO Handle 1	2	Synchronous connection handle for which PCM configuration is to be done. Range from 8 to 10.
	SCO Handle 2	2	Synchronous connection handle for which PCM configuration is to be done. Parameter valid only when Operation Mode = 0x01.
Return Param.	Status (1 octet) 0x00 = success 0x01 to 0xFF = error		
Events	Command complete event returned		
Note			

2.2.4.14 HOST_PCM_CONTROL_ENABLE

OCF	0x0070		
OGF	0x3F		
	Name	Length	Description

CMD Param.	Action	1	0x00 = PCM lines are managed by controller (default) 0x01 = PCM lines managed by host software (enable new use cases)
Return Param.	Status (1 octet) 0x00 = success 0x01 to 0xFF = error		
Events	Command complete event returned		
Note			

2.2.4.15 SET_WBS_CONNECTION

OCF	0x0073		
OGF	0x3F		
CMD Param.	Name	Length	Description
	NextScoConnectionWBS	1	0x00 = disable 0x01 = enable
Return Param.	Status (1 octet) 0x00 = success 0x01 to 0xFF = error		
Events	Command complete event returned		
Note			

2.2.4.16 SBC_INITIALIZATION

OCF	0x279		
OGF	0x3F		
CMD Param.	Name	Length	Description
	Offloading Category (static)	1	0xE0 = SBC
	SBC Offloading Type (static)	1	0x00 = partial
	SBC Offload Command (static)	1	0x01 = initialization
	Source ID	1	0x00 = I2S 0x01 = host 0x02 = FMRX 0x03 = PCM 0x02 to 0x03 = reserved 0x04 = LPBK
	Sink ID	1	0x00 = SBC SRC 0x01 = SBC SNK 0x02 = SBC
	Clock Mode	1	0x00 = clock slave 0x01 = clock master
	Bus Format	1	0x00 = I2S 0x01 = I2S MSB justified 0x02 = I2S LSB justified
	SPDIF	1	0x00 = disable

			0x01 enable
Return Param.	Status (1 byte) - Offload category (1 byte) - Offload type (1 byte) - Offload command (1 byte) - Source ID (1 byte) - Sink ID (1 byte)		
Events	Command complete event returned		
Note			

2.2.4.17 SBC_DEINITIALIZATION

OCF	0x279		
OGF	0x3F		
CMD Param.	Name	Length	Description
	Offloading Category (static)	1	0xE0 = SBC
	SBC Offloading Type (static)	1	0x00 = partial
	SBC Offload Command (static)	1	0x02 = de-initialization
	Source ID	1	0x00 = I2S
	Sink ID		0x00 = SBC SRC
Return Param.	Status (1 byte) - Offload category (1 byte) - Offload type (1 byte) - Offload command (1 byte) - Source ID (1 byte) - Sink ID (1 byte)		
Events	Command complete event returned		
Note			

2.2.4.18 SBC_SET_CONFIGURE

OCF	0x279		
OGF	0x3F		
CMD Param.	Name	Length	Description
	Offloading Category (static)	1	0xE0 = SBC
	SBC Offloading Type (static)	1	0x00 = partial
	SBC Offload Command (static)	1	0x03 =set configuration
	Source ID	1	0x00 = I2S
	Sink ID	1	0x00 = SBC SRC

	Frequency | Channel Mode	1	Bit[7]: 16K Bit[6]: 32K Bit[5]: 44K Bit[4]: 48K Bit[3]: Mono Bit[2]: Dual Bit[1]: Stereo Bit[0]: Joint Default: 0x12
	Block | Subband | Allocation	1	Bit[7]: 4-Blocks Bit[6]: 8-Blocks Bit[5]: 12-Blocks Bit[4]: 16-Blocks Bit[3]: 4-Subbands Bit[2]: 8-Subbands Bit[1]: SNR Bit[0]: Loudness Default: 0x25
	Minimum Bitpool	1	Default: 0x013
	Maximum Bitpool	1	Default: 0x34
Return Param.	Status (1 byte) - Offload category (1 byte) - Offload type (1 byte) - Offload command (1 byte) - Source ID (1 byte) - Sink ID (1 byte)		
Events	Command complete event returned		
Note			

2.2.4.19 SBC_GET_CONFIGURE

OCF	0x279		
OGF	0x3F		
CMD Param.	Name	Length	Description
	Offloading Category (static)	1	0xE0 = SBC
	SBC Offloading Type (static)	1	0x00 = partial
	SBC Offload Command (static)	1	0x04 = get configuration
	Source ID	1	0x00 = I2S
	Sink ID		0x00 = SBC SRC

Return Param.	Status (1 byte) - Offload category (1 byte) - Offload type (1 byte) - Offload command (1 byte) - Source ID (1 byte) - Sink ID (1 byte) - Frequency/channel mode (1 byte) - Block/subband/allocation (1 byte) - Minimum bitpool (1 byte) - Maximum bitpool (1 byte
Events	Command complete event returned
Note	

2.2.4.20 SBC_MEDIA_PACKET_HEADER

OCF	0x279		
OGF	0x3F		
CMD Param.	Name	Length	Description
	Offloading Category (static)	1	0xE0 = SBC
	SBC Offloading Type (static)	1	0x00 = partial
	SBC Offload Command (static)	1	0x05 = header
	Source ID	1	0x00 = I2S
	Sink ID		0x00 = SBC SRC
	ACL Handle	2	Default = 0x0001
	Media Header Length	1	Default 0x14
	Not_SBC_Per_Packet	1	Default = 0x0B
	BIGENDIAN	20	Header Default = 0x012010000C00410080010007000039600000001
Return Param.	Status (1 byte) - Offload category (1 byte) - Offload type (1 byte) - Offload command (1 byte) - Source ID (1 byte) - Sink ID (1 byte)		
Events	Command complete event returned		
Note			

2.2.4.21 SBC_CONTROL

OCF	0x279
OGF	0x3F

CMD Param.	Name	Length	Description
	Offloading Category (static)	1	0xE0 = SBC
	SBC Offloading Type (static)	1	0x00 = partial
	SBC Offload Command (static)	1	0x06 = control
	Source ID	1	0x00 = I2S
	Sink ID		0x00 = SBC SRC
	Control Command	1	0x00 = stream start 0x01 = stream stop
Return Param.	Status (1 byte) - Offload category (1 byte) - Offload type (1 byte) - Offload command (1 byte) - Source ID (1 byte) - Sink ID (1 byte)		
Events	Command complete event returned		
Note			

2.2.4.22 SBC_PROTOCOL_PARAMS

OCF	0x279		
OGF	0x3F		
CMD Param.	Name	Length	Description
	Offloading Category (static)	1	0xE0 = SBC
	SBC Offloading Type (static)	1	0x00 = partial
	SBC Offload Command (static)	1	0x07 = protocol parameters
	Source ID	1	0x00 = I2S
	Sink ID		0x00 = SBC SRC
	L2CAP MTU	2	Default = 072
Return Param.	Status (1 byte) - Offload category (1 byte) - Offload type (1 byte) - Offload command (1 byte) - Source ID (1 byte) - Sink ID (1 byte)		
Events	Command complete event returned		
Note			

2.2.4.23 SBC_CONTENT_PROTECTION

OCF	0x279		
OGF	0x3F		
CMD Param.	Name	Length	Description
	Offloading Category (static)	1	0xE0 = SBC
	SBC Offloading Type (static)	1	0x00 = partial
	SBC Offload Command (static)	1	0x08 = content protection header
	Source ID	1	0x00 = I2S
	Sink ID		0x00 = SBC SRC
	ACL Handle	2	Default = 0x0001
	CP Header Length	1	Length of content protection header Range = 0 to 8
	CP Header	0 to 8	0 to 8 bytes of header
Return Param.	Status (1 byte) - Offload category (1 byte) - Offload type (1 byte) - Offload command (1 byte) - Source ID (1 byte) - Sink ID (1 byte)		
Events	Command complete event returned		
Note			

2.2.4.24 SBC Event

Event Name	BT Offload Event		
EventCode	0xFF		
Event ID	0x79		
Param.	Name	Length	Description
	BT Offload Category	1	0xE0 = SBC offload
	BT Offload Type	1	Default = 0x0001

2.2.4.25 BLE_POWER_SAVE_MODE

OCF	0x008B		
OGF	0x3F		
CMD Param.	Name	Length	Description
	Power Mode	1	0x00 = full power mode 0x01 = deep sleep mode
Return Param.	Status (1 octet) 0x00 = success 0x01 to 0xFF = error		
Events	Command complete event returned		

Note	

2.3 Interfacing NFC controlling the SPB209A NFC function

2.3.1 General

SPB209A support most of the standard Near Field Communication (NFC) Controller Interface (NCI) Specification, see ref12 with some extensions, exception and additions. To support simpler software integration of the NFC function a NFC stack/driver is offered as a binary. Source for the NFC stack/driver and example code can be offered under software license agreement.

2.3.2 NFC initialization

The NFC interface is exported and advertised as “mnfcchar” under /dev/ (soft UART) at Bluetooth driver start up. The custom application can then operate the NFC function via the NFC stack/driver with a more high-level API than the full NCI command specification.

2.3.3 NFC stack/driver API

Function	Parameter	Description
mrvl_nci_ status_t	mrvl_nci_init(uint16_t timeout_ms, mrvl_nci_transport_config_t *config);	Initialize the mrvl_nci. Application has to call this function at the beginning of the program. It will configure the transport layer, the OS abstraction layer. Once everything is configured it will start NCI communication with NFCC and put it in IDLE mode. @param timeout_ms general timeout that will be used on all NCI calls (ms) @param config transport configuration (tty, modes, rates, ...)
	mrvl_nci_deinit(void)	Deitinitialize the mrvl_nci. Application has to call this fucntion to end mrvl_nci. This will close transport layers, threads, ...
mrvl_nci_ status_t	mrvl_nci_reset(void);	Reset the NFCC. By calling this function, mrvl_nci will reset the NFCC and put it again in IDLE. Then all configuration has to be restored by caller.
	mrvl_nci_strerror(mrvl_nci_status_t status);	Convert a mrvl_nci status to a readable string. @param status status to be converted.
	mrvl_nci_routing_reset(void);	Reset routing rules. This will reset the pending routing rules. It will not update NFCC routing rules. This function has to be called when the routing rules has to be changed. So this is called to apply the routing rules configuration called, then function to add rules are called and finally #mrvl_nci_routing_commit is

mrvl_nci_status_t	mrvl_nci_routing_add_proto(nci_rf_protocol_t protocol, uint8_t nfcee_id, mrvl_nci_power_mode_t mode);	Add a route to an NFCEE for a specific protocol. To be taken in account, a #mrvl_nci_routing_commit call has to be done. @param protocol Protocol to be routed @param nfcee_id ID of the NFCEE @param mode Power mode where this rule will be valid (mask of NCI_RF_POWER_STATE)
mrvl_nci_status_t	mrvl_nci_routing_add_aid(const uint8_t *aid, uint8_t size, uint8_t nfcee_id, mrvl_nci_power_mode_t mode);	Add a route to an NFCEE for a specific AID. To be taken in account, a #mrvl_nci_routing_commit call has to be done. @param adi AID to be routed @param size AID size @param nfcee_id ID of the NFCEE @param mode Power mode where this rule will be valid (mask of NCI_RF_POWER_STATE)
mrvl_nci_status_t	mrvl_nci_routing_commit(void);	Commit the new routing rules. This will create the global routing rules and configure the NFCC with it.
mrvl_nci_status_t	mrvl_nci_nfcee_for_protocol(nci_nfcee_protocol_t protocol, uint8_t *nfcee_id);	Get the NFCEE ID for a given protocol. @param protocol Protocol to find @param nfcee_id Pointer where the nfcee_id will be stored
mrvl_nci_status_t	mrvl_nci_nfcee_enable(uint8_t nfcee_id, bool enable);	Enable/Disable an NFCEE. @param nfcee_id ID of the NFCEE @param enable Enable if true, disable if false
mrvl_nci_status_t	mrvl_nci_nfcee_set_action_handler(uint8_t nfcee_id, mrvl_nci_nfcee_action_cb_t handler);	Set an action handler for a given NFCEE. @param nfcee_id ID of NFCEE @param handler Callback to be called on NFCEE action on this NFCEE
mrvl_nci_status_t	mrvl_nci_nfcee_ndef_configure(const uint8_t *ndef, uint16_t size, bool read_only);	NDEF NFCEE configuration NDEF NFCEE is a proprietary NFCEE implemented by Marvell to expose a type 4 tag in NFC-A or NFC-B. This tag emulation is done entirely by the NFCC. This function sets the NDEF memory content and if the NFCC has to act as a read only tag or read write. @param ndef NDEF message @param size NDEF size @param read_only R/O if true, R/W if false
mrvl_nci_status_t	mrvl_nci_nfcee_ndef_enable(bool enable);	Enable/Disable NDEF NFCEE To activate NDEF NFCEE it has to be enabled. @param enable Enable if true, disable if false
mrvl_nci_status_t	mrvl_nci_rf_enable_field_info(bool enable, mrvl_nci_rf_field_cb_t *cb);	Enable/Disable RF_FIELD_INFO notification If enable, NFCC will send a notification of external field state (ON or OFF) @param enable Enable if true, disable if false
mrvl_nci_status_t	mrvl_nci_rf_configure_discovery(mrvl_nci_rf_config_t *rf_configs, uint8_t nb_rf_config);	Configure RF discovery. RF discovery configuration is needed by the NFCC to know which RF technologies are allowed, in which mode and with cycling information.

		@param rf_configs RF configurations table @param nb_rf_config Number of entries in #rf_configs
mrvl_nci_status_t	mrvl_nci_rf_start(void);	Start RF. After init, the NFCC is in IDLE state. Calling this function will configure the RF discovery (using the configuration done by #mrvl_nci_rf_configure_discovery) and also all RF configurations needed to acheive the targeted behavior. After this call the NFCC will be in RF_IDLE state.
mrvl_nci_status_t	mrvl_nci_rf_stop(void);	Stop RF. This function can be called when the NFCC is in RF_IDLE state. It will stop RF activities and put back the NFCC in IDLE state.
	mrvl_nci_set_error_handler(mrvl_nci_error_handler_cb_t handler);	Set global error handler @param handler Error handler
	mrvl_nci_set_ndef_handler(mrvl_nci_ndef_cb_t handler);	Set NDEF handler @param handler NDEF read handler
	mrvl_nci_set_p2p_event_handler(mrvl_nci_p2p_event_cb_t handler);	Set P2P event handler @param handler P2P event handler
	mrvl_nci_allow_p2p(bool allow);	Set P2P state @param allow True to allow P2P, false to disable it
	mrvl_nci_set_preferred_protocol(nci_rf_protocol_t protocol);	Set preferred protcol when multiple targets are discovered @param protocol preferred protocol
mrvl_nci_status_t	mrvl_nci_snep_put(uint8_t *ndef, uint32_t ndef_size);	Put an NDEF message over SNEP @param ndef NDEF raw data (can be static, will be realloc and copied) @param ndef_size NDEF size
mrvl_nci_status_t	mrvl_nci_hce_register_service(mrvl_nci_hce_service_t *service);	Register an HCE service @param service HCE service
rvl_nci_status_t	mrvl_nci_hce_unregister_service(mrvl_nci_hce_service_t *service);	Unegister an HCE service @param service HCE service
rvl_nci_status_t	mrvl_nci_hce_send_packet(uint8_t *packet, uint16_t length);	Send RF packet to remote peer @param packet Data to send (must be alloc'ed, will be freed by MRVL NCI) @param length Data length
mrvl_nci_status_t	mrvl_nci_debug_enable(bool enable);	Enable NFCC traces. This can only be used on debug firmware. @param enable Enable if true, disable if false

2.4 Interfacing GPIO using HCI vendor specific commands (PROC) and FW API

The SPB209A GPIO pins have their pre-defined functions as described in the SPB209A Hardware Design Guide. Currently the FW does not provide any API that allows to use GPIOs as arbitrary in- or output pins. Use of the GPIOs is clearly defined by the FW as follows:

GPIO[17:14]	JTAG interface
GPIO[13]	BT wake up host (configured via vendor specific BT HCI command if used)
GPIO[12]	Not used
GPIO[11:8]	UART interface
GPIO[7:3]	PCM or I2S audio interface (configured via vendor specific BT HCI command if used)
GPIO[2]	LED output (can be configured via WLAN FW API)
GPIO[1]	WLAN wake-up host (configured via WLAN FW API)
GPIO[0]	oscillator enable/disable signal (default), or BT wake up host (configured via vendor specific BT HCI command if used)

The set-up is done either with HCI vendor specific commands using PROC files system or via WLAN FW API. There is only a limited set-up than can be done and this is defined below.

2.4.1 Command interface setting SPB209A GPIO

2.4.1.1 JTAG Interface

Not available for application level. Used for SPB209A test.

2.4.1.2 BT wake up host

The GPIO[13] pin is available as default for BT wake up host signaling, but it's also possible to set-up the GPIO[0] for this signaling as follows:

```
$ echo "gpio_gap=[n]" > /proc/mbt/hci0/config
$ echo "hscfgcmd=01" > /proc/mbt/hci0/config
```

[n]: bit 16:8 – gpio_pin, bit 8:0 – gpio_gap

gpio_pin: Is the pin number of GPIO used. It could be any valid GPIO pin.

2.4.1.3 UART Interface

Interface will be operational with a specific firmware supporting BT/NFC via UART. Contact technical support for details.

2.4.1.4 PCM or I2C audio interface

See "User Guide Using PCM interface in Bluetooth" on Ref3 Linux site

2.4.1.5 LED Output

To be defined

2.4.1.6 WLAN wake-up host

To be defined

2.4.1.7 Oscillator enable/disable signal

Default is function to select external High Frequency Oscillator however this is not available on SPB209A outline so it can be configured as an optional GPIO for BT wake-up host, see BT wake up host

3 References

Ref1 Freescale Yocto Project User's guide Doc No IMXLXYOCTOUG www.nxp.com
Ref2 http://www.yoctoproject.org
Ref3 http://linux.hd-wireless.se/bin/view/Linux2/WPASupplicant
Ref4 http://www.bluez.org/
Ref5 http://linuxwireless.org/en/users/Documentation/hostapd/
Ref6 http://git.kernel.org/cgit/bluetooth/bluez.git/tree/doc
Ref7 https://dbus.freedesktop.org/doc/dbus-specification.html#introduction
Ref8 https://people.csail.mit.edu/albert/bluez-intro/c33.html
Ref9 http://git.kernel.org/cgit/bluetooth/bluez.git/tree/doc?id=4.101
Ref10 http://git.kernel.org/cgit/bluetooth/bluez.git/tree/test
Ref11 ETSI TS 102 622, v7.5.0, 2009-06
Ref12 NFC Forum-TS-NCI-1.0