NRC7292 Evaluation Kit User Guide

(Host Driver Porting)

Ultra-low power & Long-range Wi-Fi

Ver 1.3
May 31, 2020

NEWRACOM, Inc.

NRC7292 Evaluation Kit User Guide (Host Driver Porting)
Ultra-low power & Long-range Wi-Fi

Office

NEWRACOM, Inc.
25361 Commercentre Drive, Lake Forest, CA 92630 USA
http://www.NEWRACOM.com

Contents

List of Tables

List of Figures

1 Overview

This guide introduces the overall SW structure of NRC7292 host driver and gives some tips for applying the driver to other Linux hosts.

1.1 Software structure

As seen in Figure 1.1, NRC7292 host driver uses Linux Kernel features, SPI, GPIO, IRQ, mac80211, and netlink socket. SPI (Slave) is used for I/O interface between the host, the NRC7292 module, and mac80211 for SW MAC, which is incorporated with MAC in the NRC7292 module. Netlink socket is used to communicate with user applications like CLI shell on host. GPIO including IRQ is used as an external interrupt source for flow control while communicating via SPI. The host applications hostapd, needs to be installed on the host to be operated as 11ah AP or wpa_supplicant as 11ah STA.

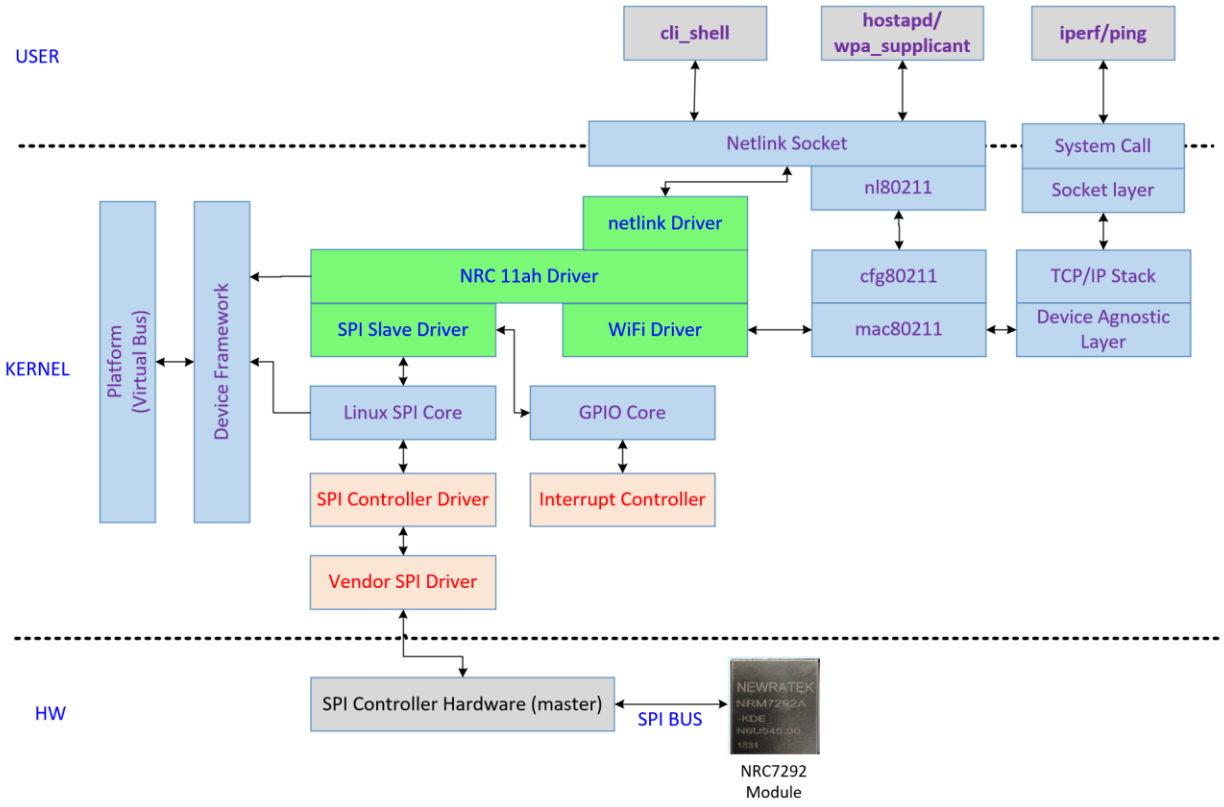

Figure 1.1 SW structure of NRC7292 host driver

1.2 Software components

NRC7292 host driver makes use of Linux Kernel APIs of GPIO, IRQ, SPI, and mac80211 so the host can support these components. To use Linux SPI/GPIO/IRQ framework or subsystem, each controller drivers provided by SoC vendor should be applied on host. If the framework is not supported on host, then the driver will be modified using other APIs (like vendor specific APIs). When it comes to mac80211, mac/cfg80211 modules should be built and loaded before loading NRC7292 host driver.

1.3 Hardware components

To communicate with NRC7292 module, SPI and GPIO (1 PIN) should be designated on Host. Figure 1.2 shows SPI and GPIO block on each Host (Raspberry Pi3) and NRC7292 Module. SPI is the main interface between Host, NRC7292 Module, GPIO, and also used as external Interrupt source for flow control.

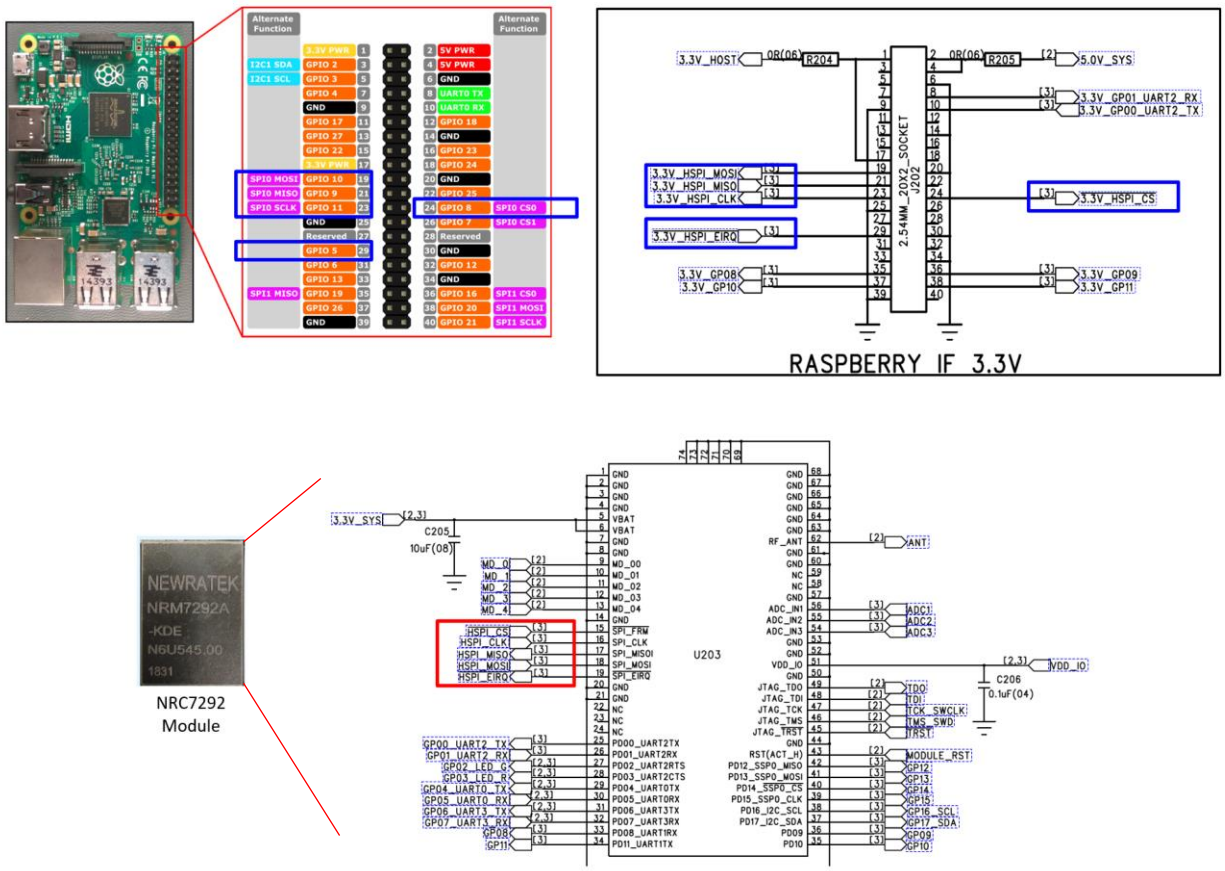

Figure 1.2 HW components on RP3 host and NRC7292 Module

2 How to build NRC7292 host driver

This chapters shows how to build NRC7292 host driver. There are 2 ways to build the driver: 1) direct-built on the host and 2) cross-compile on other hosts.

2.1 Direct compile

Before building NRC7292 host driver on host, Linux kernel sources or headers should be prepared on the host. Our drivers can cover Linux kernel version from 4.0.x to 4.14.x.; in order to build properly, changing the kernel path in Makefile is needed.

2.2 Cross compile

It is a similar procedure as direct compile. But the only difference is the preparing the cross the compiler on your PC.

```
pi@raspberrypi:~/Source/NRC_MACSW/host/linux/driver/nrc $ make -f Makefile.halow.cspi
/home/pi/Source/NRC_MACSW/host/linux/driver/nrc
make[1]: Entering directory '/usr/src/linux-raspberrypi-kernel_1.20180313-1'

  WARNING: Symbol version dump ./Module.symvers
           is missing; modules will have no dependencies and modversions.

  CC [M]  /home/pi/Source/NRC_MACSW/host/linux/driver/nrc/nrc-mac80211.o
  CC [M]  /home/pi/Source/NRC_MACSW/host/linux/driver/nrc/nrc-trx.o
  CC [M]  /home/pi/Source/NRC_MACSW/host/linux/driver/nrc/nrc-init.o
  CC [M]  /home/pi/Source/NRC_MACSW/host/linux/driver/nrc/nrc-debug.o
  CC [M]  /home/pi/Source/NRC_MACSW/host/linux/driver/nrc/hif.o
  CC [M]  /home/pi/Source/NRC_MACSW/host/linux/driver/nrc/wim.o
  CC [M]  /home/pi/Source/NRC_MACSW/host/linux/driver/nrc/nrc-hif-debug.o
  CC [M]  /home/pi/Source/NRC_MACSW/host/linux/driver/nrc/nrc-hif-uart.o
  CC [M]  /home/pi/Source/NRC_MACSW/host/linux/driver/nrc/nrc-hif-ssp.o
  CC [M]  /home/pi/Source/NRC_MACSW/host/linux/driver/nrc/nrc-fw.o
  CC [M]  /home/pi/Source/NRC_MACSW/host/linux/driver/nrc/nrc-netlink.o
  CC [M]  /home/pi/Source/NRC_MACSW/host/linux/driver/nrc/nrc-ssp.o
  CC [M]  /home/pi/Source/NRC_MACSW/host/linux/driver/nrc/nrc-hif-cspi.o
  CC [M]  /home/pi/Source/NRC_MACSW/host/linux/driver/nrc/mac80211-ext.o
  CC [M]  /home/pi/Source/NRC_MACSW/host/linux/driver/nrc/nrc-stats.o
  CC [M]  /home/pi/Source/NRC_MACSW/host/linux/driver/nrc/nrc-pm.o
  CC [M]  /home/pi/Source/NRC_MACSW/host/linux/driver/nrc/nrc-dump.o
  CC [M]  /home/pi/Source/NRC_MACSW/host/linux/driver/nrc/nrc-hif-sdio.o
  LD [M]  /home/pi/Source/NRC_MACSW/host/linux/driver/nrc/nrc.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/pi/Source/NRC_MACSW/host/linux/driver/nrc/nrc.mod.o
  LD [M]  /home/pi/Source/NRC_MACSW/host/linux/driver/nrc/nrc.ko
make[1]: Leaving directory '/usr/src/linux-raspberrypi-kernel_1.20180313-1'
```

Figure 2.1 Compile log

3 Source-code tree

Table 3.1 shows major files for NRC7292 host driver and description of them. If all the SW and HW components mentioned in Chap.1 is ready, then there is almost no need to modify source codes.

Table 3.1 Host driver files

	Category	Description
nrc-init.*	Module Initialization Linux Platform driver	Define module and module parameters Register platform device and driver
hif.*	HAL (HW Adaptation layer)	Wrapper functions for I/O
nrc-hif-cspi.*	HSPI driver	Functions for HSPI
nrc-mac80211.*	mac80211	Register driver to mac80211 Define mac80211 parameters
nrc-trx.*	Data Path	Define functions for data path
nrc-fw.*	Firmware	Define functions for FW download from host to target
nrc-netlink.*	Netlink socket	Define functions for netlink communication
nrc-pm.*	Power Management	Define functions for Wi-Fi Power Management
nrc-debug.*	Debug	Define functions for debugging
mac80211-ext.*	Utility function	Utility function about ieee80211
nrc-dump.*	Debug	Save Core Dump file when F/W Asserted
nrc-hif-sdio.*	SDIO driver	Functions for SDIO (No needed)
nrc-hif-ssp.*, nrc-ssp.*	SPI driver	Functions for SPI (Not needed)
nrc-hif-uart.*	UART driver	Functions for UART (Not needed)
nrc-pm.*	Power Save	Function for Power Save
nrc-recovery.*	Recovery	Function for Recovery that is work as watchdog function for firmware
nrc-stats.*	Statistics	Function for Statistics to check a SNR/RSSI
wim.*	WIM	Function for handling the information message between host and firmware

nrc-vendor.h	Vender IE	Definition constant/type for Vender IE
fastboot-cm0.h	Boot	Second boot loader for CM0
nrc-build-config.h	Build	Define a definition/constant for building and configuration for host driver
nrc-wim-types.h	WIM	Define a data type for the WIM

3.1 Module Parameters

As seen in Figure 1.1, NRC7292 host driver support module parameters.

```
filename:       /home/pi/nrc/nrc.ko
description:    Newracom 802.11 driver
license:        Dual BSD/GPL
author:         Newracom, Inc.(http://www.newracom.com)
srcversion:     7B3044A136969CE04029B28
depends:        mac80211,cfg80211
name:           nrc
vermagic:       4.19.75-v7+ SMP mod_unload modversions ARMv7 p2v8
parm:           fw_name:Firmware file name (charp)
parm:           hifport:HIF port device name (charp)
parm:           hifspeed:HIF port speed (int)
parm:           spi_bus_num:SPI controller bus number (int)
parm:           spi_cs_num:SPI chip select number (int)
parm:           spi_gpio_irq:SPI gpio irq (int)
parm:           spi_gdma_irq:SPI gdma irq (int)
parm:           alternate_mode:SPI mode alternated (bool)
parm:           loopback:HIF loopback (bool)
parm:           lb_count:HIF loopback Buffer count (int)
parm:           disable_cqm:Disable CQM (0: disable, 1:enable) (int)
parm:           listen_interval:Listen Interval (int)
parm:           bss_max_idle:BSS Max Idle (int)
parm:           enable_short_bi:Enable Short BI (bool)
parm:           enable_monitor:Enable Monitor (bool)
parm:           bss_max_idle_offset:BSS Max Idle Offset (int)
parm:           macaddr:MAC Address (charp)
parm:           power_save:power save (int)
parm:           wlantest:wlantest (bool)
```

Figure 3.1 Module Parameters

According to HW configuration, some SPI and GPIO parameters like chip_select, bus_number, max_speed_hz, etc. should be changed when driver is inserted to kernel. For SPI, user can set using spi_bus_num, spi_cs_num, spi_gpio_irq and hifspeed in module parameters.

4 Host SPI

NRC7292 contains SPI slave engine for a host interface. The SPI slave engine consists of two separate domain, device and host side, as shown in Figure 4.1. This separation is mainly for power save. In a deep-sleep mode, most parts of NRC7292 including device side are turned off, but the host side keeps awake to monitor wake-up trigger from the external host.

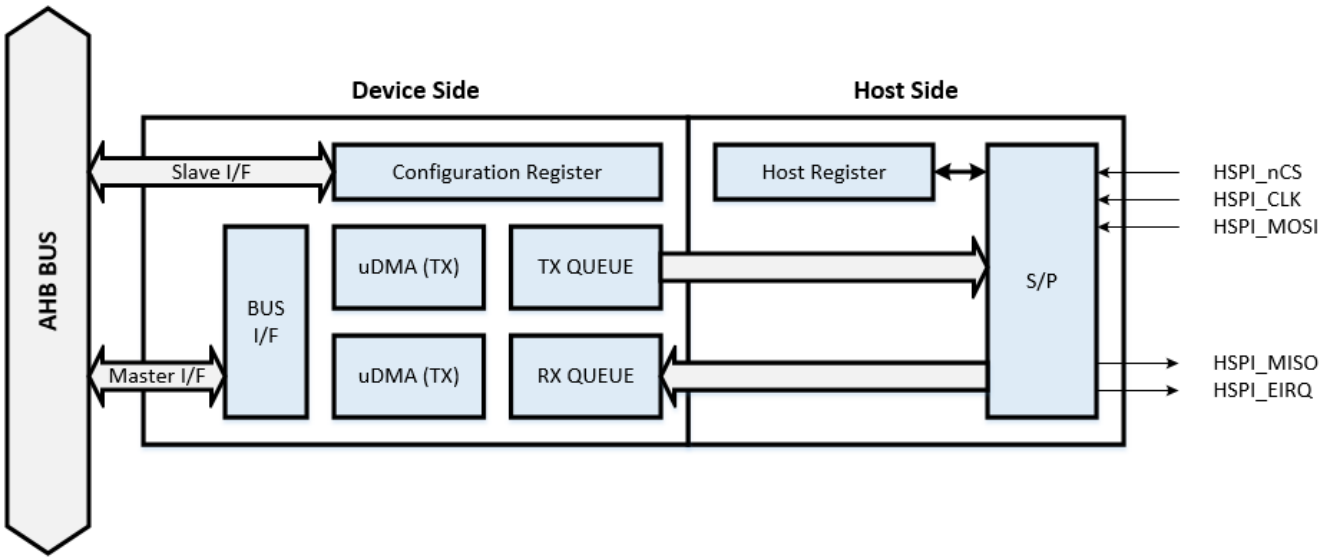

Figure 4.1 Block diagram of SPI slave engine of NRC7292

Table 4.1 HSPI pin description

Pin Name	I/O	Description
HSPI_EIRQ	Output	Interrupt to external host
HSPI_MOSI	Input	Master out Slave in
HSPI_MISO	Output	Master in Slave out
HSPI_nCS	Input	Chip select (active low)
HSPI_CLK	Input	Clock

A total of 5 dedicated pins are assigned for HSPI interface as presented in **Error! Reference source not found.**.

To use HSPI interface, the BOOT mode must be configured to HOST BOOT mode. The HOST BOOT mode can be selected by tying MODE_01 and MODE_02 pins to ground and MODE_00 pin to DVDDE_MODE. The DVDDE_MODE is the power supply for MODE_xx pins and the DVDDE_HSPI is for HSPI IOs such as HSPI_EIRQ and HSPI_MISO.

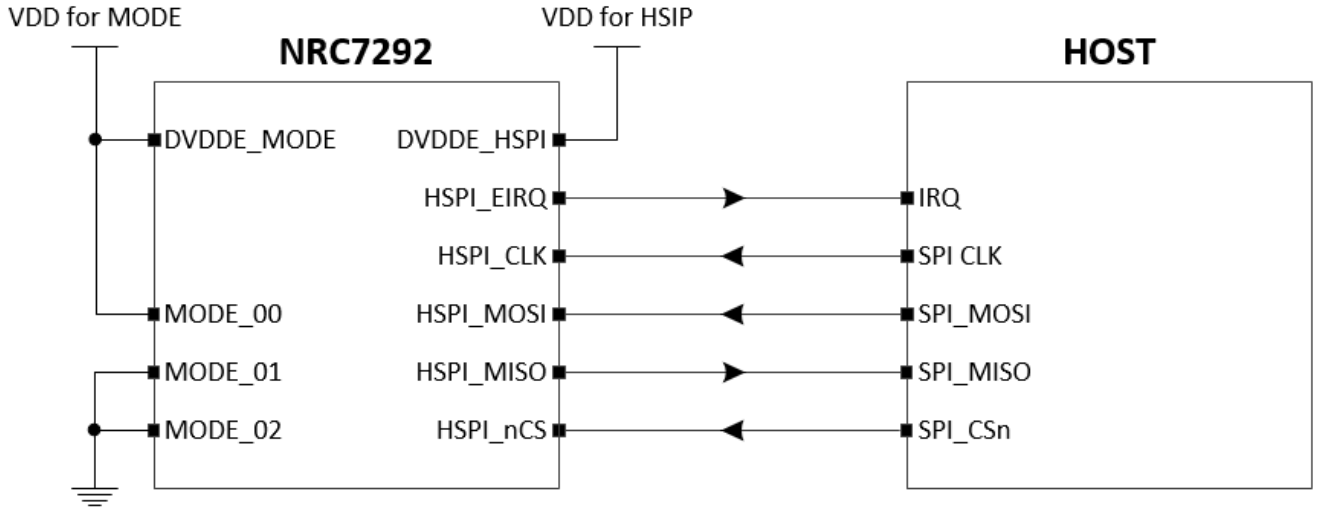

Figure 4.2 An example of H/W configuration for HSPI interface

SPI slave engine asserts HSPI_EIRQ interrupt when its TX QUEUE has data to send or the status of RX QUEUE is changed. Therefore, when the HSPI_EIRQ is asserted, the external host needs to check what triggers the interrupt after clearing the interrupt by reading EIRQ_CLEAR register(0x12).

As shown in Figure 4.3, SPI slave engine supports SPI mode 0 (CPOL = 0 & CPHA = 0). The HSPI_nCS must be held low for entire read/write cycle and must be taken high at least one clock period after the read/write cycle completed.

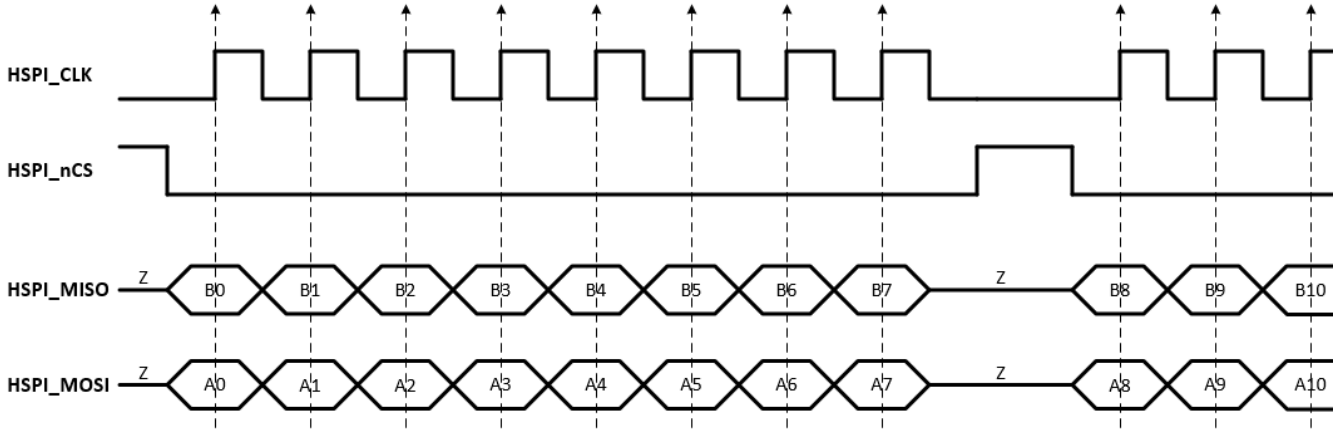

Figure 4.3 Timing diagram of HSPI interface

4.1 Single transfer mode

The single and burst transfer modes are defined. In single transfer mode, only one byte of data can be transferred from host to slave and vice versa. Figure 4.4 and Figure 4.5 represent the frame format for HSPI master write and read, respectively in single transfer mode. The single transfer mode starts with the command period and ends with the response period. The command period is composed of a 32-bits argument and a 16-bits cyclic redundancy check (CRC) part. The host (HSPI master) should check the acknowledgment (ACK) before sending the next command when writing data to HSPI slave. The ACK in response period is sent from HSPI slave to HSPI master to inform that it receives command correctly. When reading data, the HSPI slave transmit the data in response period.

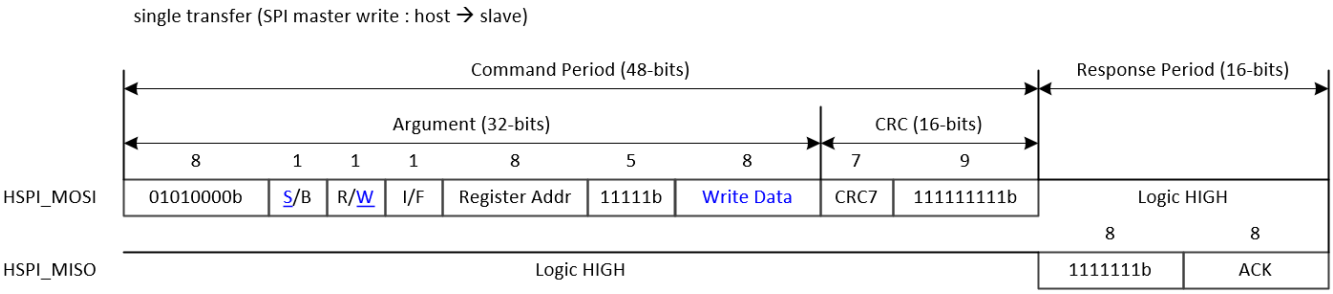

Figure 4.4 Frame format of HSPI master write in single transfer mode

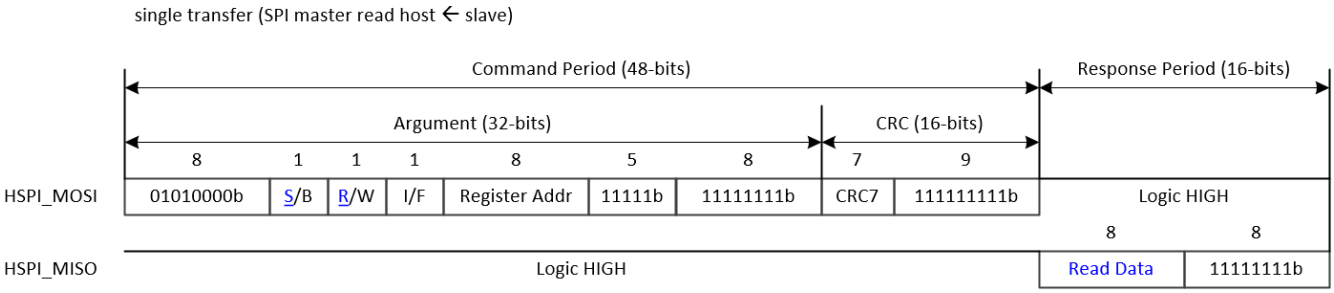

Figure 4.5 Frame format of HSPI master read in single transfer mode

Table 4.2 Field description of frame in single transfer mode

Field		# bits	Description
Argument	01010000b	8	header of the command period
	S/B	1	0 : single transfer 1 : burst transfer
	R/W	1	0 : read 1 : write
	I/F	1	0 : address increment 1 : address fix
	Register Addr	8	register address to access
	11111b	5	stuff bits
	Write Data / Stuff bits	8	when R/W = 0 : 0xFF when R/W = 1 : write data
CRC	CRC7	7	7 bits CRC calculation based on 32-bits argument
	111111111b	9	stuff bits
Response	Read Data / Stuff bits	8	when R/W = 0 : read data @ register address when R/W = 1 : 0xFF
	ACK	8	0x47

4.2 Burst transfer mode

Burst transfer mode also starts with the command period followed by a response period like the single transfer mode. However, additional data period follows the response period as shown in Figure 4.6 and Figure 4.7. In the data period, HSPI master writes or reads a number of data.

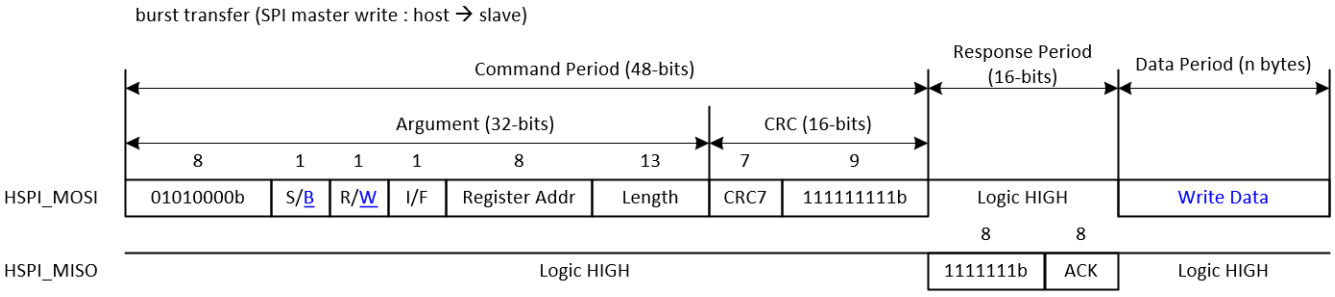

Figure 4.6 Frame format of HSPI master write in burst transfer mode

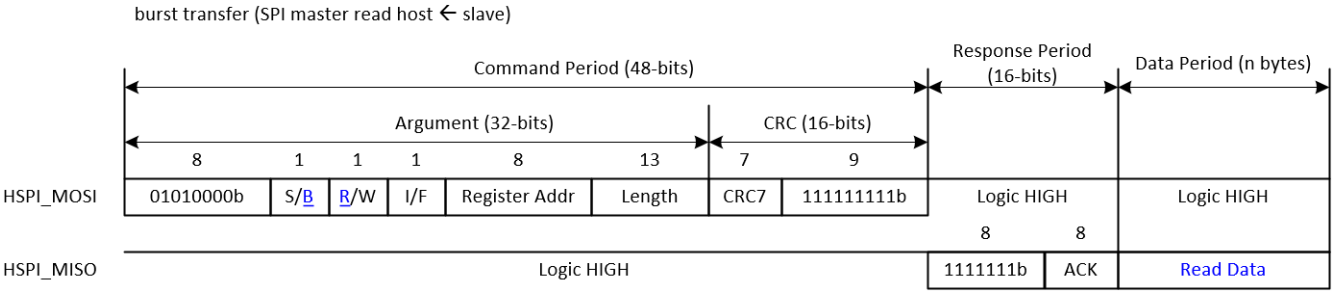

Figure 4.7 Frame format of HSPI master read in burst transfer mode

The frame structure of burst transfer mode is almost the same as the structure of the single transfer mode except for the length field at the end of the 32-bit argument. This 13-bit length field can represent up to 8K bytes and indicates the data size in byte.

Table 4.3 Field description of frame in burst transfer mode

Field		# bits	Description
Argument	01010000b	8	header of the command period
	S/B	1	0 : single transfer 1 : burst transfer
	R/W	1	0 : read 1 : write
	I/F	1	0 : address increment 1 : address fix
	Register Addr	8	register address to access
	Length	13	data length in byte
CRC	CRC7	7	7 bits CRC calculation based on 32-bits argument
	111111111b	9	stuff bits
Response	Read Data / Stuff bits	8	when R/W = 0 : read data @ register address when R/W = 1 : 0xFF
	ACK	8	0x47

5 Host Interface Layer

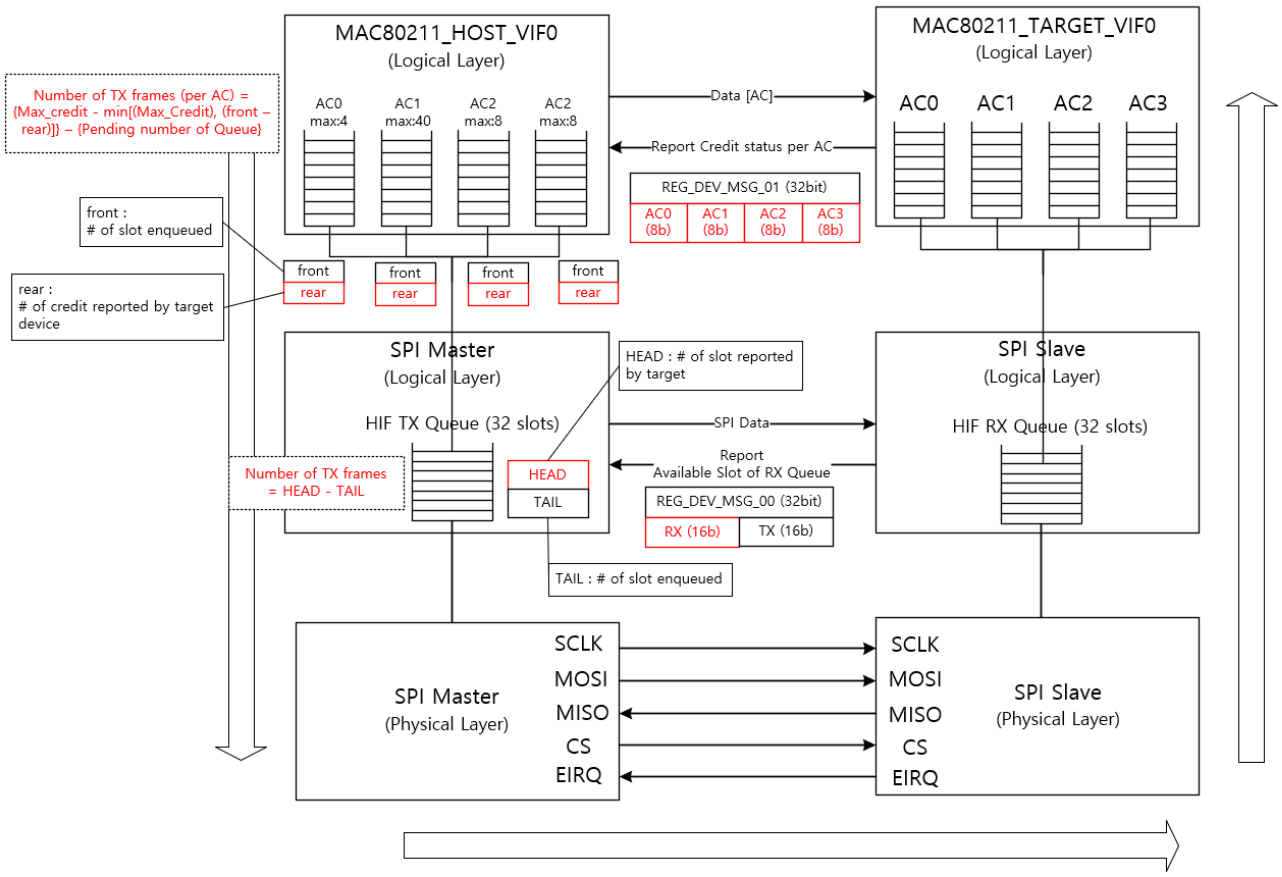

Figure 5.1 Slot and Credit (TX path on Host vs RX path on Target Device)

HIF (Host interface) layer located between MAC80211 (Linux Provide) and CSPI driver (Newracom Provide) is responsible for TRX data path. Basically, it abstracts various physical layers such as HSPI, SPI, UART, and SDIO, but we only focus on HSPI in this document.

HIF layer attaches its own header (called HIF Header) to payloads that are delivered from the upper layer (MAC80211), and then transmits them to target device via HSPI. On the contrary, HIF layer extracts some fields from HIF header generated by FW on target device whenever receiving frames from target device, and then finally delivers them to upper layer (MAC80211) without HIF header. (i.e. only MAC payload) The detailed structure of the HIF header will be described later.

As mentioned in previous chapter, there is “flow control function” that smoothly controls influx and outflux of frames between host and target device by SW. For this, two concepts are introduced, one is “slot” and the other is “credit”. ‘Slot’ is for preventing buffer overflow on target device and ‘Credit’ is for prioritization of AC (Access Category) used by 802.11 QoS. Credit is applied only for TX (not RX). Target device conditionally and repeatedly reports its TRX buffer status to host by asserting HSPI_EIRQ,

and then host should read the status before enqueueing frames on its TX queue or reading frames via HSPI.

Figure 5.1 shows this concept of slot and credit for TX Path. There are three independent layers, SPI physical layer, SPI logical layer (with slot), MAC80211 logical layer (with credit). HSPI_EIRQ that is asserted by target device makes host read the status register that informs available RX slot number on target device. Host has to decide whether or/and how many frames to transmit to target according to this slot report (refer to HEAD) and its own TX queue status (refer to TAIL). For example, there are 10 frames already enqueued in TX queue on host and host was reported with 15 available RX slots from target device. Then host can transmit only 5 frames via HSPI. In this condition, if host transmits 7 frames via HSPI, there might be overflow of RX queue on target device. The concept of credit is almost the same except that it is per AC (Access Category) and VIF (Virtual Interface). Similarly, according to credit report from target device and its TX queue status per AC (and VIF), host decides whether or/and how many frames are dequeued from each AC's queue and enqueued to HIF TX queue. However, this calculation is NOT done just by front (enqueued number of frame) and rear (reported credit number) like "rear – front". "Max Credit number per AC (AC0:4, AC1:40, AC2:8, AC3:8)" is introduced for enough influx frames to HIF TX queue. It is because handling frames on HIF TX Queue is done in the context of "workqueue" (refer to Linux workqueue concept). So credit calculation is finally done by "{Max_Credit_per_AC – min[Max_Credit, (front- rear)]} – {Pending Number of Queue per AC} " considering prioritization of each AC.

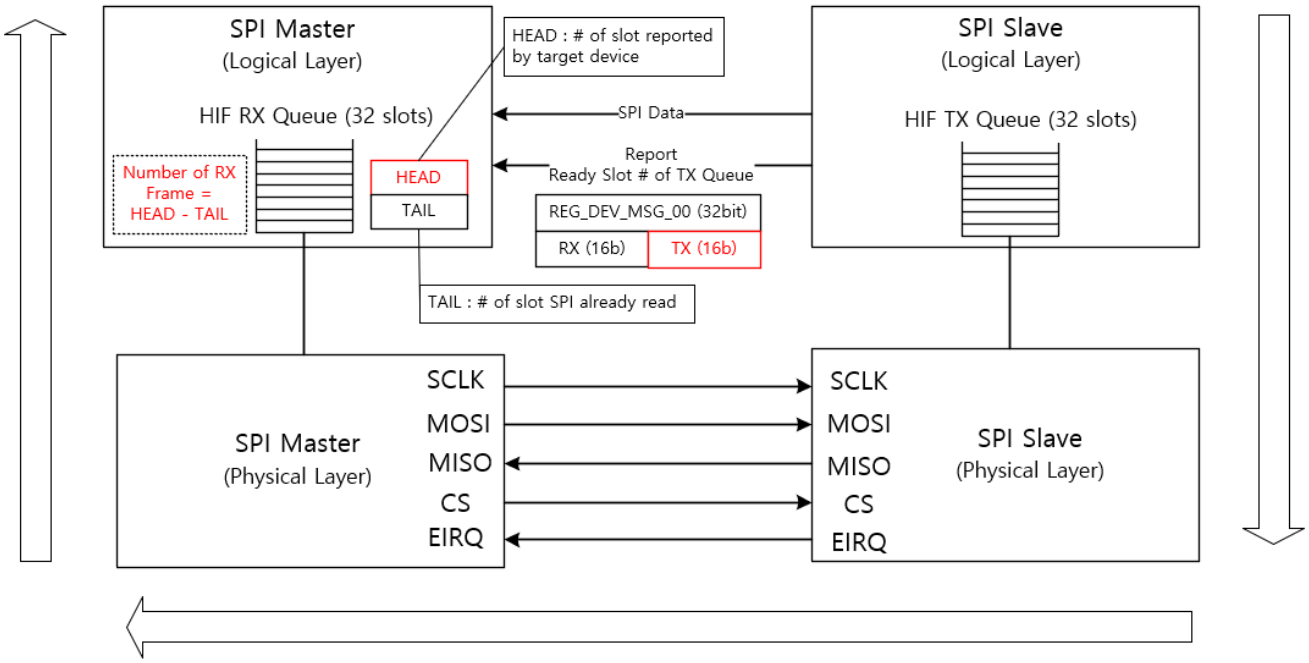

Figure 5.2 Slot and Credit (RX path on Host vs TX path on Target Device)

Figure 5.2 shows slot concept for RX path on host. It's much simpler than TX path because there is no credit concept. Host decides whether or/and how many frames to read via HSPI using TX Ready Slot Report from target device (refer to HEAD) and total number of frames read via HSPI (refer to TAIL). For

example, host is reported with 15 TX Ready slots from target device, and then host starts receiving frames until it reads 15 frames via HSPI.

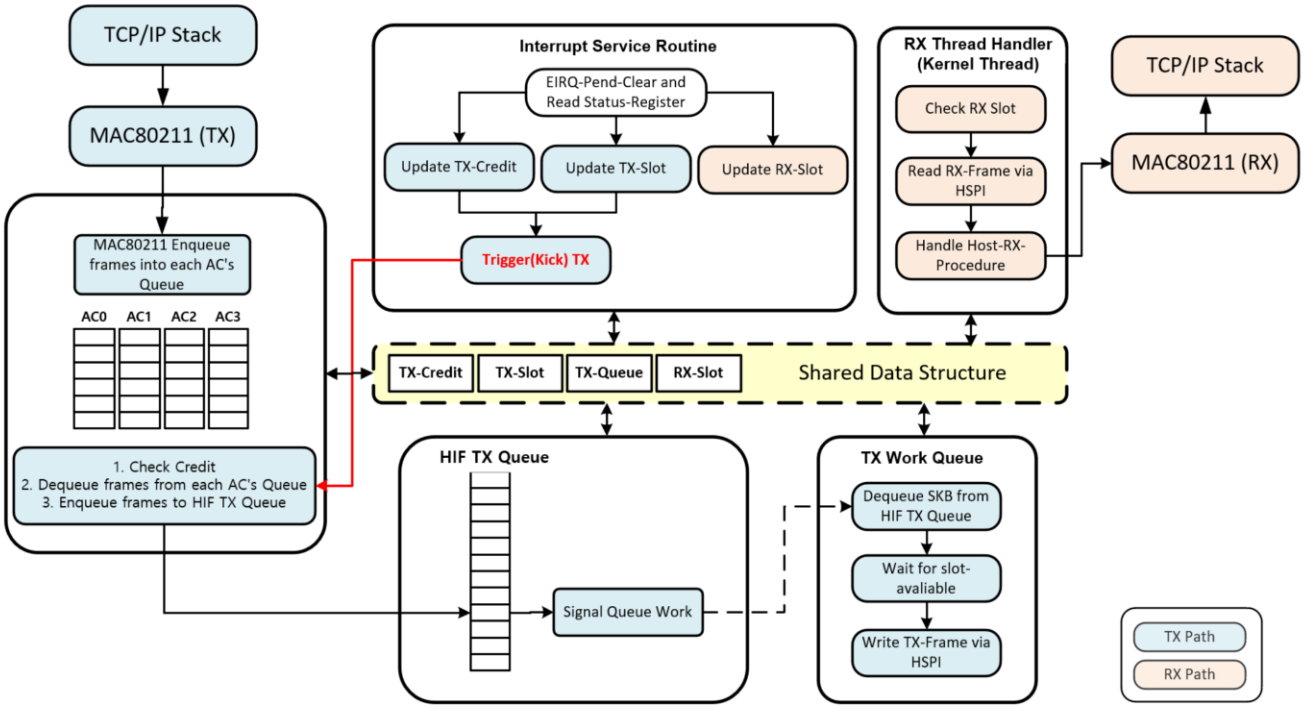

Figure 5.3 Host Interface Layer (TRX operation for ISR)

Figure 5.3 shows HIF Layer in general. For resource limitation on NRC7292 SoC, HSPI slave hardware engine has very small number TX/RX QUEUE (FIFO) inside, so TX credit and TRX slot reported using EIRQ and related registers are crucial. The condition of asserting EIRQ by target device is explained in the next section.

5.1 Interrupt Handling

Figure 5.3 shows the TRX operation with interrupt handling. Once the HSPI_EIRQ is asserted, EIRQ ISR (Interrupt Service Routine) checks what triggers interrupt after clearing interrupt by reading EIRQ_CLEAR (0x12) register. If the interrupt is caused by the “Credit Report or RX-Slot Report”, host kicks TX operation. If target device asserts EIRQ for Report TX-Slot, host stats RX operation.

NRC7292 (Target Device) asserts HSPI_EIRQ when the following events are occurred.

- TX credit updated
- TX QUEUE status updated
- RX QUEUE status updated

6 Wireless Information Message (WIM)

The WIM is used to request the operation of the firmware on target device. For example, Host Driver requests to set the frequency and BSSID after successful association. In the contrast, WIM is also used to notify host of some events occurred by firmware on target device.

The WIM protocol works base on HIF protocol structure as below.

HIF-Header(8B)	WIM-Header(4B)	TLVs(n byte)

```
struct hif_hdr {
    uint8_t type;
    uint8_t subtype;
    uint8_t flags;
    int8_t vifindex;
    uint16_t len;
    uint16_t tlv_len;
} __packed;
```

```
struct wim_hdr {
    union {
        uint16_t cmd;
        uint16_t resp;
        uint16_t event;
    };
    uint8_t seq_no;
    uint8_t n_tlvs;
} __packed;
```

HIF header indicates payload type such as Frame (data, mgnt, control), WIM, etc.

HIF_TYPE_WIM in hif_hdr should be set if the frame type is the WIM. The struct wim_hdr indicates how many Type-Length-Value (TLV) were loaded in the payload like frequency(one TLV), BSSID(one TLV), etc.

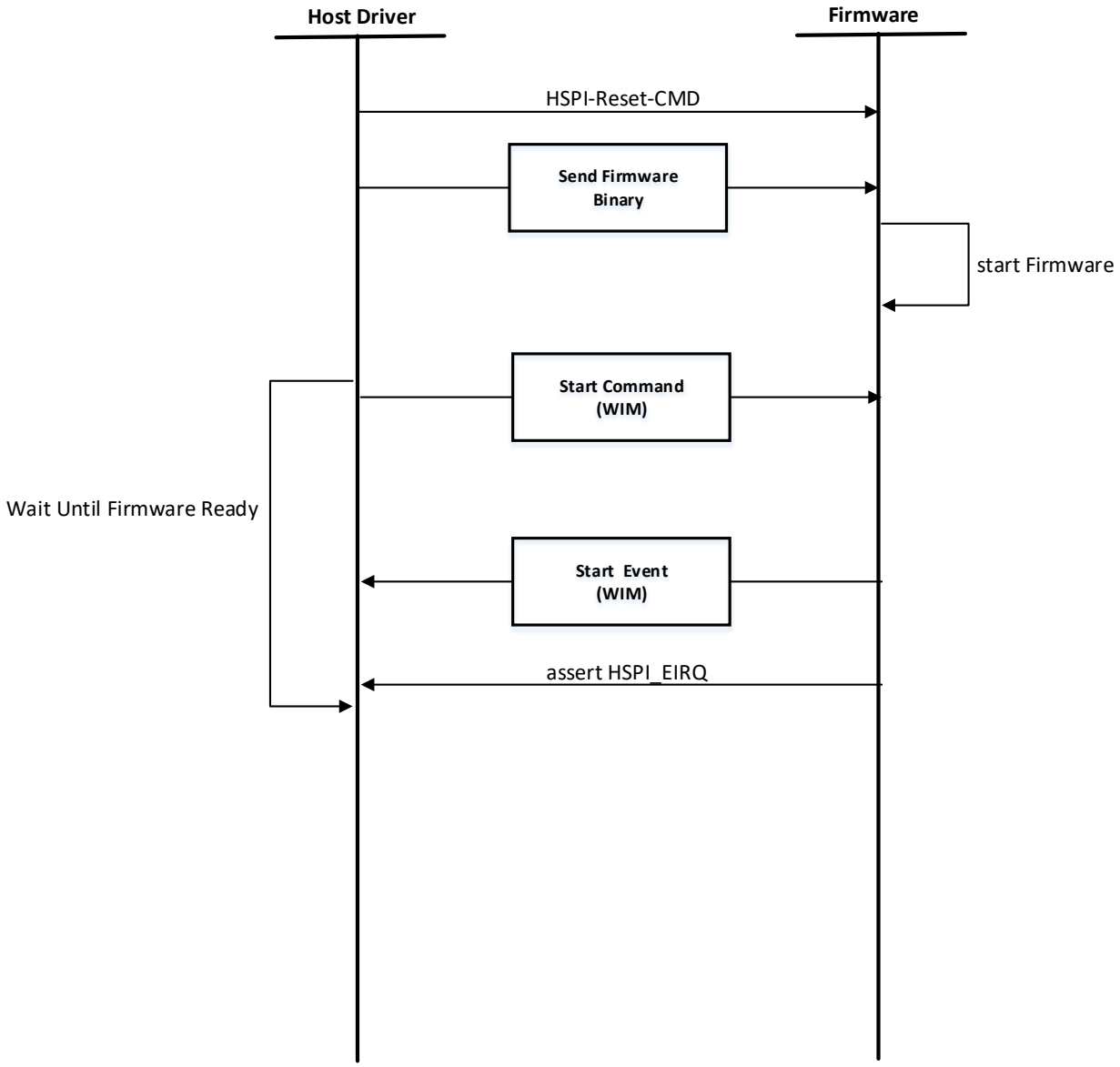

Figure 6.1. Initialize Sequence

Figure 6.1 describes how host driver downloads the firmware via HSPI and initializes it with WIM command and WIM event.

Host must wait until WIM Event comes back after issuing Start command.

6.1 WIM Command

This document explains basic WIM in the source code.

Table 6.1 WIM command

NAME	NOTE
WIM_CMD_START	Start command for firmware. Wait for start event from firmware
WIM_CMD_STOP	Start command for firmware when it unloaded the host driver
WIM_CMD_SCAN_START	Start scan command.
WIM_CMD_SCAN_STOP	Stop scan command
WIM_CMD_SET_KEY	Security key set command
WIM_CMD_DISABLE_KEY	Security key remove command
WIM_CMD_STA_CMD	Station information update command
WIM_CMD_SET	Set various TLVs command
WIM_CMD_REQ_FW	Firmware binary update command

6.2 WIM Event

Table 6.2 WIM event

NAME	NOTE
WIM_EVENT_SCAN_COMPLETED	Event to host when firmware completes the scan
WIM_EVENT_READY	Event to host when firmware completes initialization
WIM_EVENT_CREDIT_REPORT	Event to host when the credit has changed in firmware.

6.3 WIM TLVs

Table 6.3 WIM TLVs

NAME	NOTE
WIM_TLV_BSSID	uint8_t bssid[6] set bssid to firmware
WIM_TLV_MACADDR	uint8_t macaddr[6] set mac address to firmware
WIM_TLV_AID	uint16_t aid set aid to firmware
WIM_TLV_STA_TYPE	uint32_t sta_type set station type to firmware [WIM_STA_TYPE_STA, WIM_STA_TYPE_AP]
WIM_TLV_SCAN_PARAM	struct wim_scan_param set scan parameter for firmware scaning

WIM_TLV_KEY_PARAM	struct wim_key_param set security parameter for firmware security engine
WIM_TLV_STA_PARAM	struct wim_sta_param set sta parameter for firmware station information
WIM_TLV_READY	struct wim_ready_param various firmware parameter for host driver
WIM_TLV_AC_CREDIT_REPORT	struct wim_credit_report_param credit report for host driver from firmware
WIM_TLV_CH_BW	struct wim_ch_bw_param set frequency and bandwidth parameter for firmware

7 HSPI Register map

Table 7.1 Registers for HSPI

Address	Register	R/W	Description
System register			
0x00	WAKEUP	W	HSPI wakes up when writing 0x79
0x01	DEV_RESET	W	HSPI reset when writing 0xC8
IRQ register			
0x10	EIRQ_MODE	R/W	[07:03] Reserved [02] EIRQ_IO_EN (1: IO enable) [01] EIRQ_MODE1 (0: level trig., 1: edge trig.) [00] EIRQ_MODE2 (0: active low, 1: active high)
0x11	EIRQ_ENABLE	R/W	[07:04] Reserved [03] DEV_SLEEP_IRQ (1: enable) [02] DEV_READY_IRQ (1: enable) [01] TX_QUEUE_IRQ (1: enable) [00] RX_QUEUE_IRQ (1: enable)
0x12	EIRQ_CLEAR	R	EIRQ clear register Clear all interrupt when read
0x13	EIRQ_STATUS	R	[07:04] Reserved [03] DEV_SLEEP_STATUS [02] DEV_READY_STATUS [01] TX_QUEUE_STATUS [00] RX_QUEUE_STATUS
0x14	QUEUE_STATUS	R	[07:00] TX QUEUE status [47:40]
0x15			[07:00] TX QUEUE status [39:32]
0x16			[07:00] TX QUEUE status [31:24]
0x17			[07:00] TX QUEUE status [23:16]
0x18			[07:00] TX QUEUE status [15:08]
0x19			[07:00] TX QUEUE status [07:00]
0x1A			[07:00] RX QUEUE status [47:40]
0x1B			[07:00] RX QUEUE status [39:32]
0x1C			[07:00] RX QUEUE status [31:24]
0x1D			[07:00] RX QUEUE status [23:16]
0x1E			[07:00] RX QUEUE status [15:08]
0x1F			[07:00] RX QUEUE status [07:00]
Message register			
0x20	DEV_MSG_00	R	[07:00] Device message [31:24]
0x21			[07:00] Device message [23:16]

0x22			[07:00] Device message [15:08]
0x23			[07:00] Device message [07:00]
0x24	DEV_MSG_01	R	[07:00] Device message [31:24]
0x25			[07:00] Device message [23:16]
0x26			[07:00] Device message [15:08]
0x27			[07:00] Device message [07:00]
0x28	DEV_MSG_02	R	[07:00] Device message [31:24]
0x29			[07:00] Device message [23:16]
0x2A			[07:00] Device message [15:08]
0x2B			[07:00] Device message [07:00]
0x2C	DEV_MSG_03	R	[07:00] Device message [31:24]
0x2D			[07:00] Device message [23:16]
0x2E			[07:00] Device message [15:08]
0x2F			[07:00] Device message [07:00]
RX QUEUE register			
0x31	RXQUEUE_WINDOW	W	[07:00] received data from HSPI master (host)
TX QUEUE register			
0x41	TXQUEUE_WINDOW	R	[07:00] transmitted data to HSPI master (host)

8 Revision history

Revision No	Date	Comments
Ver 1.0	11/01/2018	Initial version for customer release created
Ver 1.1	11/07/2019	Add 3.1 module parameters and delete H/W dependency codes
Ver 1.2	05/30/2020	Update source-code tree
Ver 1.3	05/31/2020	Add the host driver porting guide detailed