NRC7292 Evaluation Kit User Guide

(Standalone)

Ultra-low power & Long-range Wi-Fi

Ver 1.8
Oct 22, 2021

NEWRACOM, Inc.

NRC7292 Evaluation Kit User Guide (Standalone)
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 document introduces the NRC7292 Software Development Kit (SDK) that allows users to develop their application program running on NRC7292 Evaluation Boards (EVB) without external hosts. Figure 1.1 to Figure 1.3 show the pictures of NRC7292 EVB.

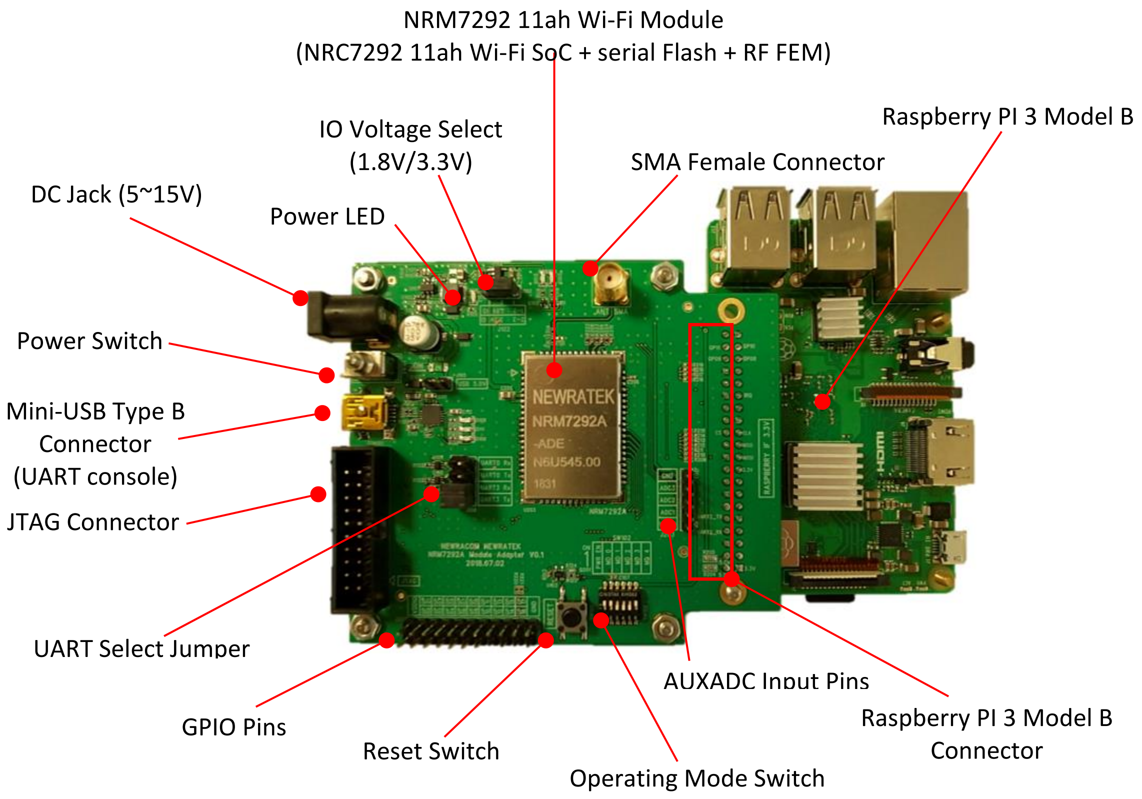

NRM7292 11ah Wi-Fi Module
(NRC7292 11ah Wi-Fi SoC + serial Flash + RF FEM)
IO Voltage Select (1.8V/3.3V)
Raspberry PI 3 Model B
SMA Female Connector
DC Jack (5~15V)
Power LED
Power Switch
Mini-USB Type B Connector (UART console)
JTAG Connector
UART Select Jumper
AUXADC Input Pins
GPIO Pins
Reset Switch
Raspberry PI 3 Model B Connector
Operating Mode Switch

Figure 1.1 NRC7292 evaluation board (v.1.0)

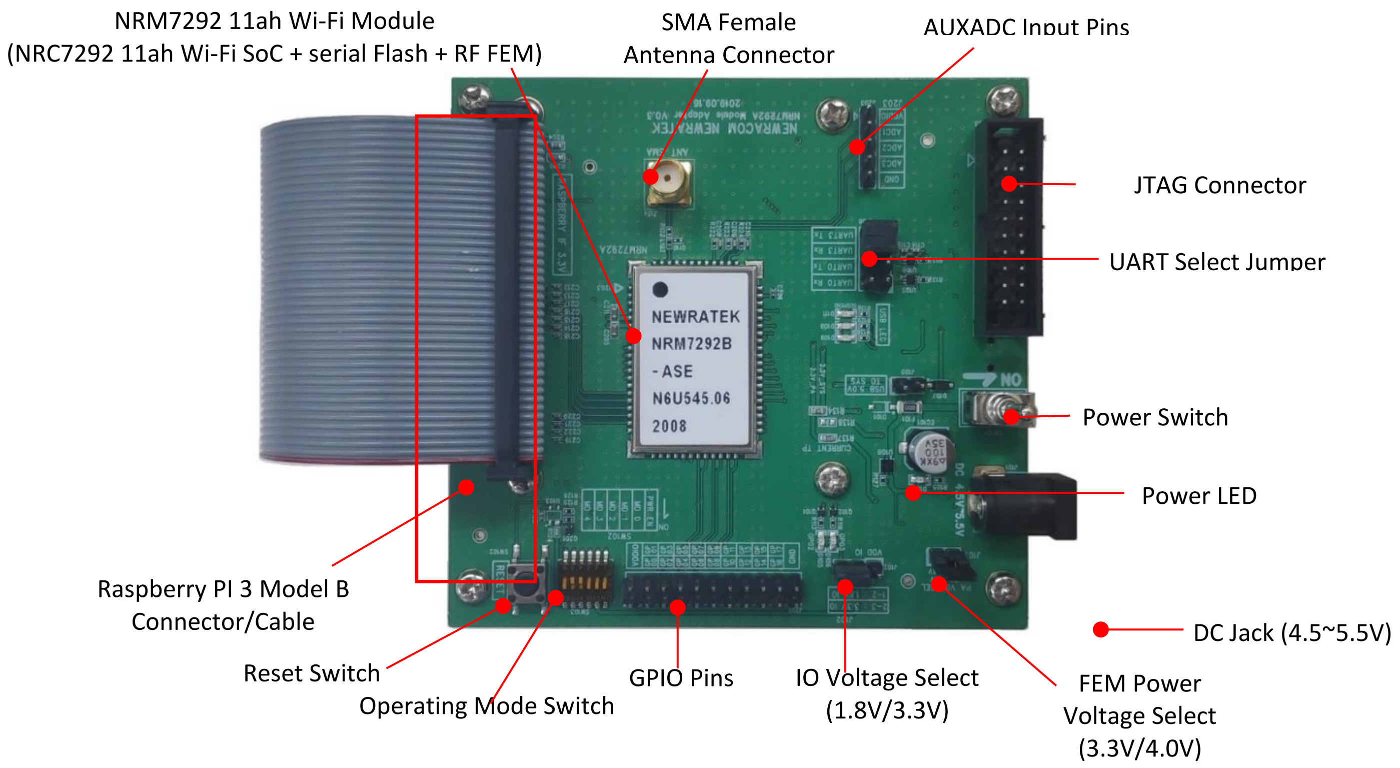


Figure 1.2 NRC7292 evaluation board (v.2.0 – top view)

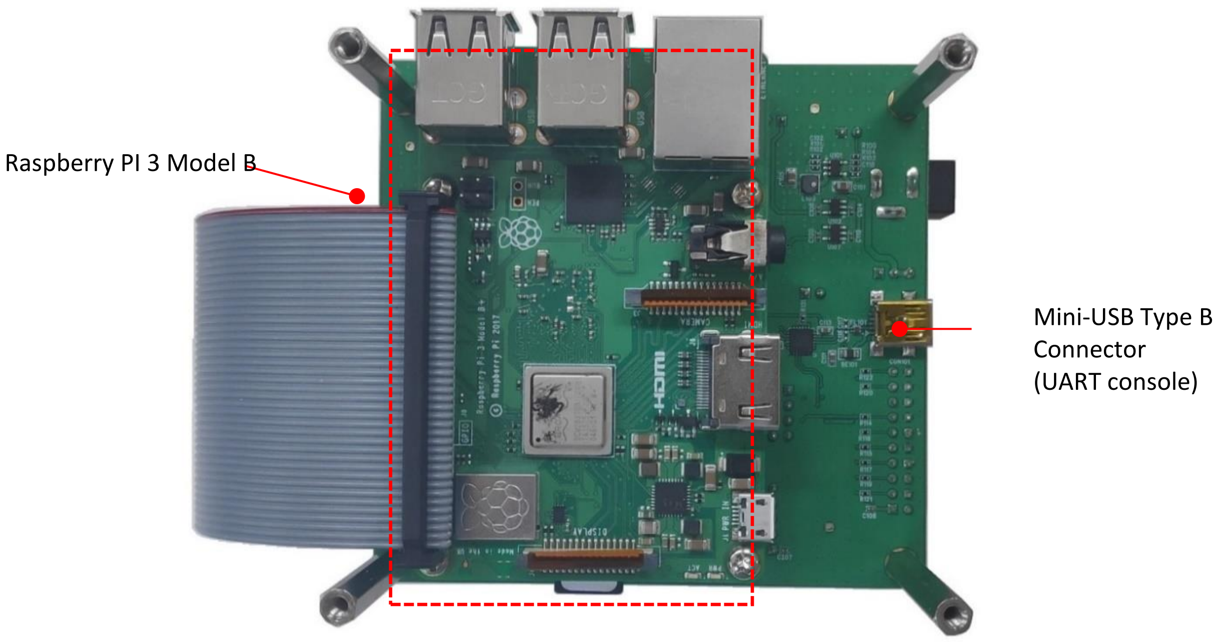


Figure 1.3 NRC7292 evaluation board (v.2.0 – bottom view)

1.1 H/W list

The NRC7292 EVB consists of three components: an 11ah Wi-Fi module, an adapter board, and a host board.

1.1.1 NRC7292 module board

The NRC7292 module contains the IEEE 802.11ah Wi-Fi SoC solution. It also includes an RF front end module to amplify the transmission power up to 23 dBm. The on-board serial flash memory can be used for Over-The-Air (OTA) firmware update, user configuration data storage. The module also supports Execution-In-Place (XIP) functionality along with the 32KB cache in the SoC.

1.1.2 NRC7292 adapter board

The NRC7292 adapter board mainly acts as a communication interface to sensors or an external host. It also supplies main power to the NRC7292 Wi-Fi module.

1.1.3 Host board

The use of a Raspberry Pi 3 (RPi3) host is optional for standalone operation. The NRC7292 module can be used either as a standalone or a slave to a host processor via serial peripheral interface (SPI) or universal asynchronous receive transmitter (UART). The RPi 3 host can be used for test or AT-command operation.

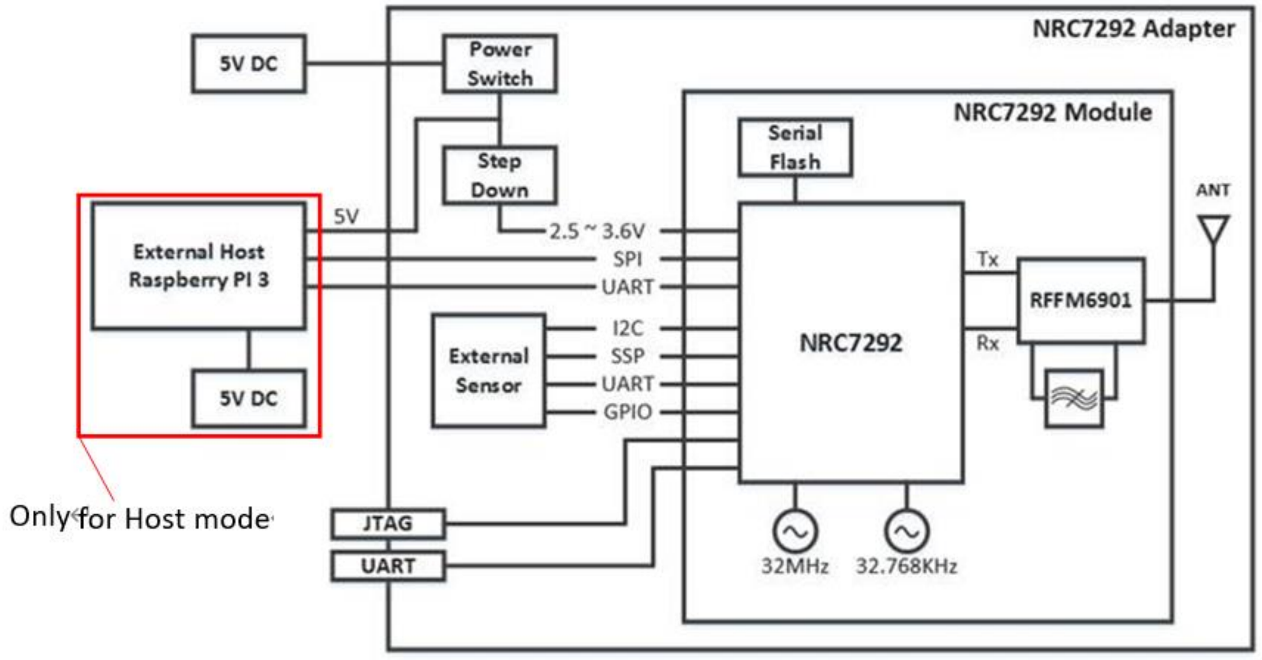


Figure 1.4 NRC7292 evaluation board block diagram

1.2 S/W list

1.2.1 NRC7292 SDK

The NRC7292 SDK can be used to develop user's application program running on NRC7292 EVB. The SDK includes various types of Application Program Interfaces (APIs) for controlling Wi-Fi connectivity, Transport Control Protocol/Internet Protocol (TCP/IP) communication, peripherals, timer, memory, etc. In addition, users can attach various sensors on the evaluation board and communicate with them via UART, SPI, or I2C APIs.

As all standalone applications run on FreeRTOS, users can take advantage of FreeRTOS features including multi-tasking, Inter-Task Communication (ITC), memory management, etc.
(Refer to at https://www.freertos.org for more information)

1.2.2 NRC7292 standalone firmware downloader

The NRC7292 Standalone Firmware Downloader for Windows included in the release package can be used to download a unified binary onto the flash memory on the EVB.

2 Setup S/W build environment

The NRC7292 SDK supports a Linux environment. This chapter describes how to set up the development environment, build a user's application program, and download the binary on the EVB.

2.1 Toolchain setup

GNU ARM embedded toolchain is required to build the user's application program. Note that users should use 64-bit Linux machine to build successfully.

- Ubuntu 16.04 LTS (64-bit PC(AMD64) desktop image) or later
- GCC toolchain for ARM embedded processors
  Download the GNU Arm embedded toolchain v7-2018Q2
  gcc-arm-none-eabi-7-2018-q2-update-linux.tar.bz2 (version must exactly match)
  https://developer.arm.com/tools-and-software/open-source-software/developer-tools/gnu-toolchain/gnu-rm/downloads

Before installing the ARM embedded toolchain, users need some additional packages which can be easily installed through the standard package manager (apt-get) for Ubuntu. The following instructions discuss which packages are required, with instructions on how to install them.

```
sudo apt-get update
sudo apt-get install build-essential python2.7 python-pip git
```

Once the required packages are successfully installed, download the GCC toolchain (gcc-arm-none-eabi-7-2018-q2-update-linux.tar.bz2) from the ARM developer website, copy the file to $HOME location, and extract it.

```
tar -xvf gcc-arm-none-eabi-7-2018-q2-update-linux.tar.bz2
```

The GCC toolchain will be extracted into ~/ gcc-arm-none-eabi-7-2018-q2-update-linux directory. If the PATH environmental variable is set as shown below, users can run the GCC toolchain anywhere without giving the complete path for the toolchain.

```
export PATH=”$PATH:$HOME/gcc-arm-none-eabi-7-2018-q2-update/bin”
```

2.2 Download SDK

Users can download the NRC7292 SDK from GitHub (https://github.com/newracom/nrc7292_sdk.git).

```
git clone https://github.com/newracom/nrc7292_sdk.git
```

The NRC7292 SDK in the repository consists of several subdirectories: doc, lib, make, sdk, apps, and tool. The doc directory contains all the documents for the users, including the user guides. The lib directory holds various third-party library codes including FreeRTOS, LwIP, Mbed TLS, etc. along with the SDK modem library. The make and apps directories carry makefiles and sample application programs, respectively. The tool holds the NRC7292 Standalone Firmware Downloader, AT cmd test tool and board data editor.

```
.
├──── doc
├──── lib
|     ├──── FreeRTOS
|     ├──── aws_iot
|     ├──── cJSON
|     ├──── hostap
|     ├──── libcoap
|     ├──── lwip
|     ├──── mbedtls
|     ├──── modem
|     ├──── mxml
|     ├──── paho.mqtt
|     ├──── tinycbor
|     └──── tr6260
├──── make
├──── sdk
|     └──── inc
|     └──── apps
└──── tools
      └──── downloader
      └──── AT_CMD_Test_Tool
      └──── BoardDataEditor
```

2.3 SDK application program

2.3.1 Sample application programs

```
├──── atcmd
├──── hello_world
├──── ........................................
├──── sample_tcp_client
├──── sample_tcp_server
├──── ........................................
└──── wifi-common
```

The package provides various sample application programs in the ‘nrc7292_sdk/package/standalone/sdk/ apps’ directory. However, the ‘atcmd’ and the ‘wifi-common’ directory in the ‘sdk/ apps’ are not a sample program but contains header and source files for AT-command and Wi-Fi connectivity, respectively. If users want to use Wi-Fi functionalities, they must include the header files in their application program.

The ‘nrc7292_sdk/package/standalone/sdk/inc’ directory contains the API header files for GPIO, I2C, UART, etc. Only the header files for these APIs are provided.

2.3.2 Application program project structure

Except for the AT-command application, every project directory has the .config, Makefile, and main source file that is the same name as the project. For example, as shown below, the sample application project for the TCP client has the main source file named ‘sample_tcp_client’, which is the same as its project name.

The main source file should contain the ‘void user_init(void)’ function that serves as the entry point of the application.

```
├──── sample_tcp_client
│     ├──── .config
│     ├──── Makefile
│     └──── sample_tcp_client.c
└──── wifi_common
      ├──── wifi_common.make
      ├──── wifi_config.h
      ├──── wifi_config_setup.c
      ├──── wifi_config_setup.h
      ├──── wifi_connect_common.c
      └──── wifi_connect_common.h
```

As mentioned above, to use the Wi-Fi connectivity, users should include the header files of 'wifi_common' in their main source file as well as adding 'wifi_common.make' at the end of the Makefile as shown below. Also, the application source files must be listed following the CSRCS variable in the Makefile.

```
CSRCS += \
          sample_tcp_client.c

include $(SDK_WIFI_COMMON)/wifi_common.make
```

Some third-party libraries (CJSON, MQTT, MXML, AWS, TINYCBOR, COAP, etc.) are provided in the package. Users can easily include these libraries by selecting each of them in the .config file. For example, if the application requires the CJSON library, the user can simply change 'n' to 'y' in the .config file for use.

```
CONFIG_CJSON  = y
CONFIG_MQTT  = n
CONFIG_MXML = n
CONFIG_AWS    = n
CONFIG_TINYCBOR = n
CONFIG_COAP = n
```

2.3.3 Build application program

Users can use the 'make' command at the standalone directory (nrc7292_sdk/package/standalone) to build the application program. Before running the 'make' command, however, users must create the build-target file (.build-target) which specifies the makefile used for build and the name of the application project.

Example of .build-target file:

```
MAKEFILE = nrc7292.sdk.release

PARAM := -- APP_NAME=sample_tcp_client
```

Users can create the build-target file by following the instruction below at the standalone directory.

Usage of make command for build-target file:

```
make select target=nrc7292.sdk.release APP_NAME=($APP NAME)
```

For the general application programs, users need to give the name of the application project as the APP_NAME. For example, to build a sample TCP client application, users can write a command as shown below.

```
make select target=nrc7292.sdk.release APP_NAME=sample_tcp_client
```

For the AT-command application programs, pre-defined name of the application should be given as follows.

HSPI mode:

```
make select target=nrc7292.sdk.release APP_NAME=ATCMD_HSPI
```

UART mode (without hardware flow control):

```
make select target=nrc7292.sdk.release APP_NAME=ATCMD_UART
```

UART mode (with hardware flow control):

```
make select target=nrc7292.sdk.release APP_NAME=ATCMD_UART_HFC
```

Once the build-target file is created at the standalone directory, users can run the 'make' command at the same directory. This command will generate the map, elf, and unified binary file of the application program at the 'out/nrc7292/standalone_xip/{project_name}' directory.

The binary file 'nrc7292_standalone_xip_{project_name}.bin' can then be downloaded onto the module.

2.3.4 SDK APIs

Various SDK APIs in several categories are provided for user application programming as shown in Table 2.1. Please refer to UG-7292-005-Standalone SDK API in the packet for more information.

◈ *To print out logs via UART console (see Figure 1.1 and Figure 1.3), the debug UART console must first be enabled by calling the API function, "nrc_uart_console_enable()." Once the console is enabled, users can print the logs out to the UART console using the API function, "nrc_usr_print()."*

Table 2.1 NRC7292 SDK APIs

Category	Description
Wi-Fi	Wi-Fi connection
System	System configuration and Log level
Timer	Timer-based application
UART	UART peripheral I/O
GPIO	GPIO peripheral I/O
I2C	I2C peripheral I/O
ADC	ADC peripheral I/O
PWM	PWM peripheral I/O
SPI	SPI peripheral I/O
HTTP Client	HTTP Client
FOTA	Firmware Over-The-Air
Power Save	Sleep mode (Modem sleep / Deep sleep)
WPS_PBC	WPS pushbutton
System	System configuration and
HTTP Client	HTTP Client
FOTA	Firmware Over-The-Air

2.3.5 Sample applications

Table 2.2 provide the information of the various sample application programs included in the release package.

Table 2.2 NRC7292 sample applications

Category	Subcategory	Name	Description
Helloworld	Console print	hello_world	Repeatedly print hello message
AT-CMD	AT-CMD	Atcmd	AT command
Wi-Fi	Connection	sample_wifi_state	Repeat Wi-Fi connection and disconnection every 3 seconds
		sample_wps_pbc	Connect and AP with WPS-PBC
		sample_wifi_roaming	Scan external APs and connect to an AP with strong RSSI
	SoftAP	sample_softap_udp_server	Run SoftAP and receive UDP data
		sample_softap_tcp_server	Run SoftAP and receive TCP data
Protocol	UDP	sample_udp_client	Send a UDP packet every 1 second
		sample_udp_server	Receive UDP packets
	TCP	sample_tcp_client	Connect to a TCP server and send 1000 packets to the TCP server
		sample_tcp_server	Start a TCP server, wait for an incoming TCP client connection and receive data from the connected TCP client
	HTTP	sample_http	Send a HTTP request and receive the corresponding HTTP response
		sample_http_server	Run SoftAP with HTTP server and then run as STA after submitting WLAN infomation
	FOTA	sample_fota	Run FOTA operation
Peripheral	Timer	sample_timer	Start 6 timers with different periods
	Memory	sample_memory	Repeatedly allocate and free memory
	GPIO	sample_gpio	Set GPIO_01(LED) as a pin and toggle on and off every 1 second (LED blinks 1 sec)
	UART	sample_uart	Bytes fed into UART CH2 are sent to a remote UDP server
	SPI	sample_spi	Communicate with a sensor via SPI

	I2C	sample_i2c	Communicate with a sensor via I2C
	ADC	sample_adc	Communicate with a sensor via ADC
	PWM	sample_pwm	Enable PWM and configure the PWM duty cycle for an analog sensor
	Serial flash	sample_nvs	Read/write user config area
	Power Save	sample_ps_modem_standalone	Modem sleep operation
		sample_ps_standalone	Deep sleep operation
		sample_ps_tcp_client	Repeatedly send TCP data and enter the deep sleep mode
Middleware	MQTT	Sample_mqtt	Send data to MQTT server using MQTT protocol
	XML	sample_xml	Test XML creation and conversion behavior
	CJSON	sample_json	Test JSON creation and conversion behavior
	AWS	sample_aws_client	Connect to aws and publish message
	OneM2M	sample_onem2m_client	Connect to oneM2M and receive data via MQTT
	CoAP	sample_coap_server	Execute CoAP server
		sample_coap_client	Execute CoAP client and get information from server
Test	IPERF	iperf_tcp_client	iperf tcp client application
		iperf_tcp_server	iperf tcp server application
		iperf_udp_client	iperf udp client application
		iperf_udp_server	iperf udp server application

3 How to download compiled binaries

The NRC7292 Standalone Firmware Downloader in the ‘tool’ directory can be used to download the unified binary onto the EVB. The steps outlined below explain how to download the binary.

3.1 UART connection between PC and EVB

Connect the PC to the EVB using a UART-USB cable and check the corresponding COM port number using the Device Manager. The COM port number will be required in the next step.

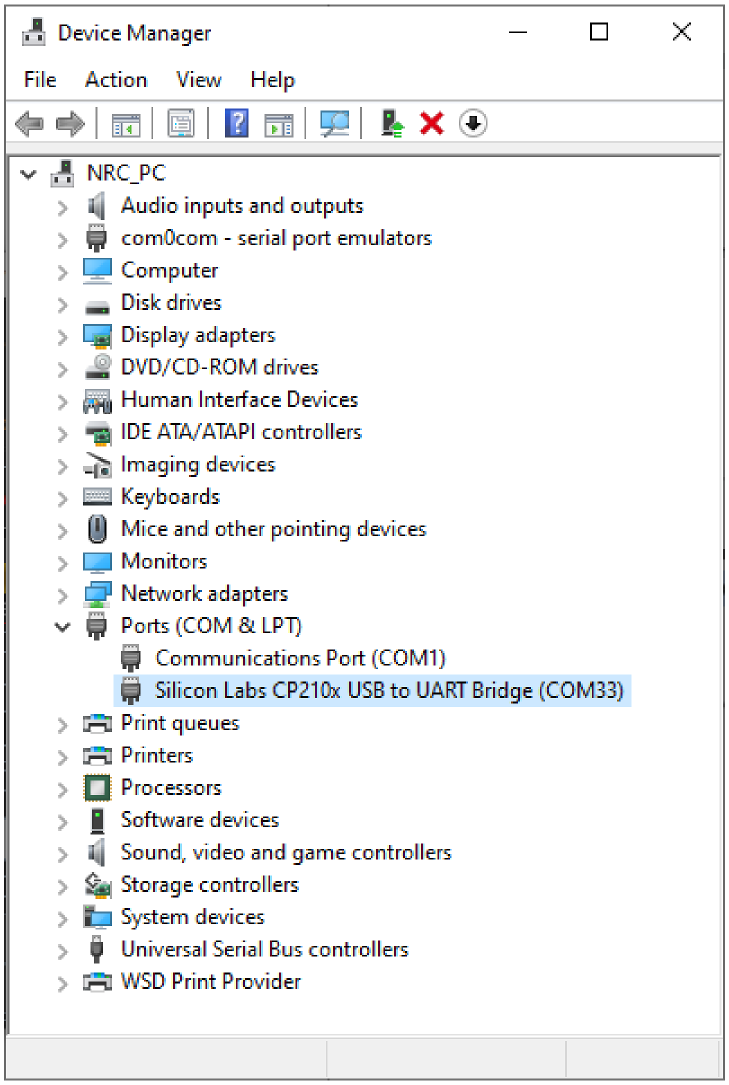
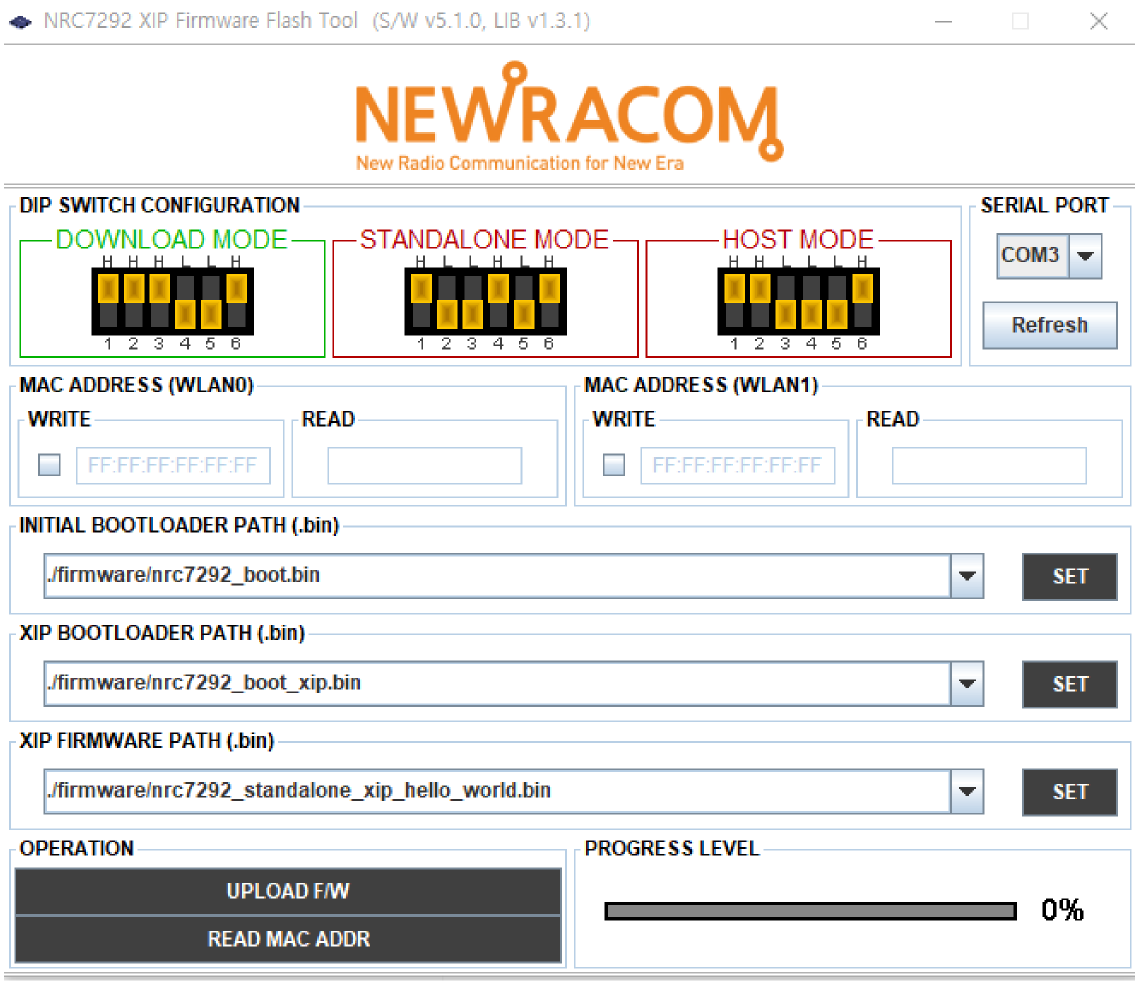


Figure 3.1 COM port in Device Manager

3.2 Upload the unified binary

Launch the NRC7292 Standalone Firmware flash tool and select the correct serial port. Either directly type in the path to the standalone XIP boot and firmware binary or press the '**SET**' button to launch the file selector. The initial bootloader and XIP Boot is located in './firmware/' folder and assigned path automatically. So, the developer does not need to change boot path. MAC addresses (for WLAN0 and WLAN1) can be read from the flash.

Before click the download f/w, the developer should be change the DIP Switch mode to DOWNLOAD MODE, 'HHHLLH' in advance.


NRC7292 XIP Firmware Flash Tool (S/W v5.1.0, LIB v1.3.1)

DIP SWITCH CONFIGURATION: DOWNLOAD MODE H H H L L H; STANDALONE MODE H L L H L H; HOST MODE H H L L L H

SERIAL PORT: COM3

MAC ADDRESS (WLAN0) WRITE: FF:FF:FF:FF:FF:FF

MAC ADDRESS (WLAN1) WRITE: FF:FF:FF:FF:FF:FF

INITIAL BOOTLOADER PATH (.bin): ./firmware/nrc7292_boot.bin

XIP BOOTLOADER PATH (.bin): ./firmware/nrc7292_boot_xip.bin

XIP FIRMWARE PATH (.bin): ./firmware/nrc7292_standalone_xip_hello_world.bin

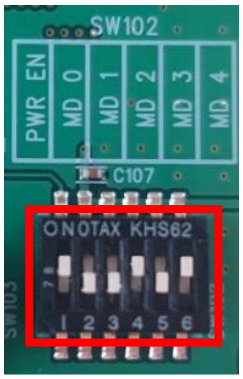
PROGRESS LEVEL: 0%

Figure 3.2 NRC7292 standalone firmware downloader

To start downloading the selected binary, click the '**DOWNLOAD F/W**' button.

3.3 Standalone operation mode

After downloading the firmware, the DIP switch must be configured to the standalone mode as shown in the figure below. Pressing the reset button on the module will start the standalone operation.



U D D U D U U(Up), D(Down)

Figure 3.3 Standalone mode DIP switch configuration

4 Performance evaluation

Iperf is a tool for network performance measurement and tuning. It has TCP/UDP client and server functionalities and can create data streams to measure the throughput between the two ends. For performance evaluation, the NRC7292 standalone release package provides sample programs for Iperf TCP/UDP client and server, which use SDK APIs and LwIP sockets.

Before building the sample programs, users need to change the bolded constants shown below as appropriate to their need. For example, the user must redefine the **COUNTRY_CODE** variable shown in the sample code to the country information that the user desires to use the program in. The **NRC_WIFI _DHCP_ENABLE** variable provides an option to use a static IP address or IP address that a DHCP server assigns.

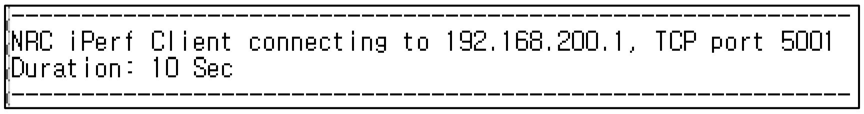
[wifi_config.h header file]

```
#ifndef __WIFI_CONFIG_H__
#define __WIFI_CONFIG_H__

#define STR_SSID "halow_demo"
#define COUNTRY_CODE "KR" /* KR:Korea, JP: Japan, US:United States, TW: Taiwan, EU: EURO */

#define NRC_WIFI_SECURE    MODE_WPA2 /* MODE_OPEN,    MODE_WPA2 */
#define NRC_WIFI_DHCP_ENABLE 0

#define NRC_REMOTE_ADDRESS "192.168.200.1"

#define TX_POWER 17 /* 17 dBm */

#define NRC_WIFI_TEST_COUNT 100
#define NRC_WIFI_TEST_INTERVAL 1000 /* ms */

#if defined(NRC_WIFI_SECURE) && (NRC_WIFI_SECURE == MODE_WPA2)
#define NRC_WIFI_PASSWORD    "12345678"
#endif

#if (NRC_WIFI_DHCP_ENABLE != 1)
#define NRC_STATIC_IP "192.168.200.13"
#endif

#define NRC_WIFI_SOFTAP_DHCP_SERVER 1
#define NRC_AP_IP "192.168.200.1"

#endif /* __WIFI_CONFIG_H__ */
```

Once the build is successfully completed, users can download the unified binary to the EVB by using the NRC7292 Standalone Firmware Downloader tool.

4.1 Iperf TCP client

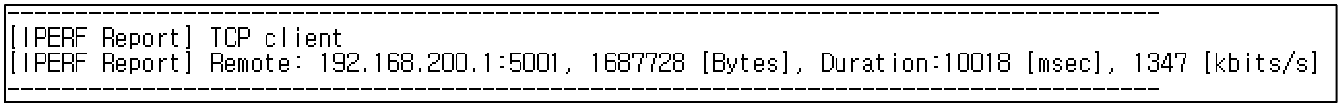
Before running the Iperf TCP client sample program, the Iperf TCP server should be ready at the AP. When the Iperf TCP client starts, users can find the start message as shown in Figure 4.1. The start message gives the IP address of the peer device (AP), TCP port information, and test time. Users can change the test time and port number by redefining the **DEFAULT_DURATION** and **REMOTE_PORT** variable, respectively, in the iperf_tcp_client.c file.



```
------------------------------------------------------------
NRC iPerf Client connecting to 192.168.200.1, TCP port 5001
Duration: 10 Sec
------------------------------------------------------------
```

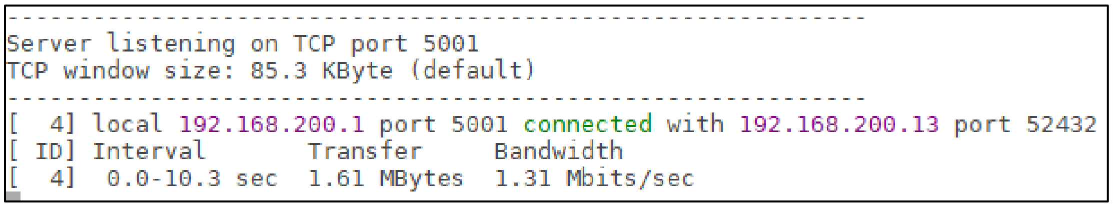
Figure 4.1 Start message of Iperf TCP client

Figure 4.2 and Figure 4.3 present the result of performing the Iperf TCP client and server, respectively, for 10 seconds.



```
---------------------------------------------------------------------------------
[IPERF Report] TCP client
[IPERF Report] Remote: 192.168.200.1:5001, 1687728 [Bytes], Duration:10018 [msec], 1347 [kbits/s]
---------------------------------------------------------------------------------
```

Figure 4.2 Test result for Iperf TCP client in standalone mode STA



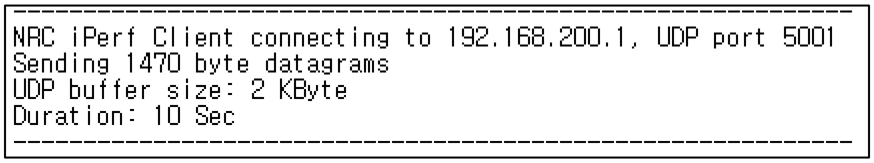
```
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  4] local 192.168.200.1 port 5001 connected with 192.168.200.13 port 52432
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-10.3 sec  1.61 MBytes  1.31 Mbits/sec
```

Figure 4.3 Test result for Iperf TCP server in host mode AP

4.2 Iperf UDP client

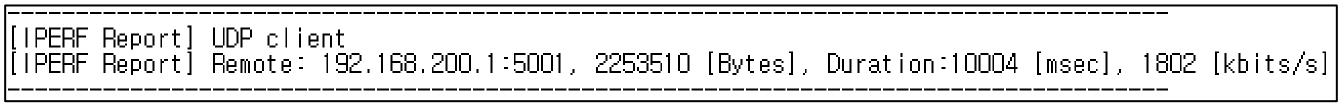
The way to start and operate in the Iperf UDP client is almost the same as the Iperf TCP client except for the contents displayed on the UART console.

Figure 4.4 to Figure 4.6 present in sequence the start message, test result for the Iperf UDP client in standalone mode STA, and test result for the Iperf UDP server in host mode AP.



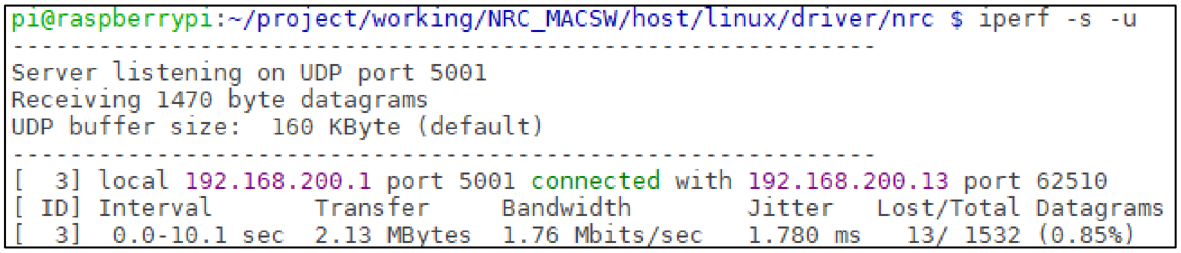
```
-------------------------------------------------------------
NRC iPerf Client connecting to 192.168.200.1, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size: 2 KByte
Duration: 10 Sec
-------------------------------------------------------------
```

Figure 4.4 Start message of Iperf UDP client



```
-------------------------------------------------------------------------------------
[IPERF Report] UDP client
[IPERF Report] Remote: 192.168.200.1:5001, 2253510 [Bytes], Duration:10004 [msec], 1802 [kbits/s]
-------------------------------------------------------------------------------------
```

Figure 4.5 Test result for Iperf UDP client in standalone mode STA

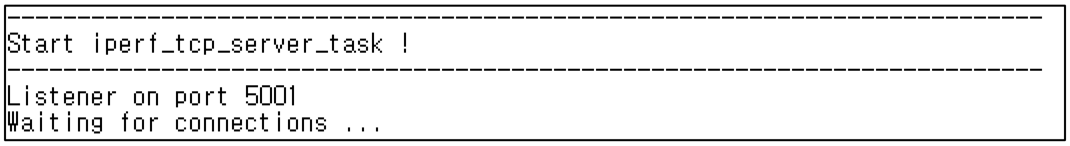


```
pi@raspberrypi:~/project/working/NRC_MACSW/host/linux/driver/nrc $ iperf -s -u
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  160 KByte (default)
------------------------------------------------------------
[  3] local 192.168.200.1 port 5001 connected with 192.168.200.13 port 62510
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  3]  0.0-10.1 sec  2.13 MBytes  1.76 Mbits/sec   1.780 ms   13/ 1532 (0.85%)
```

Figure 4.6 Test result for Iperf UDP server in host mode AP

4.3 Iperf TCP server

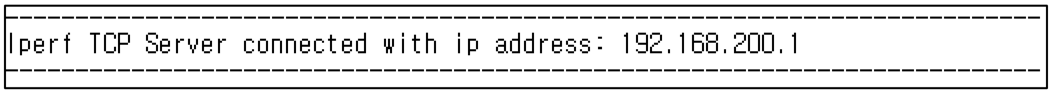
Users can find the start message as Figure 4.7 shows. When the sample program starts, it opens a TCP socket and waits for a connection from a TCP client. Note that the sample program for the Iperf TCP/UDP server can support only a single connection. Like the Iperf TCP client, users can change the port number by redefining the **LOCAL_PORT** variable in the iperf_tcp_server.c file.



```
---------------------------------------------------------------------------
Start iperf_tcp_server_task !
---------------------------------------------------------------------------
Listener on port 5001
Waiting for connections ...
```

Figure 4.7 Start message of Iperf TCP server in standalone mode STA

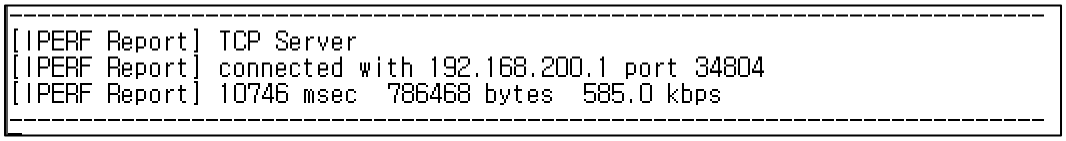
Once the Iperf TCP client is connected, users can see the message as shown in Figure 4.8, which displays the IP address of the peer device (Iperf TCP client).



```
-------------------------------------------------------------------------------
Iperf TCP Server connected with ip address: 192.168.200.1
-------------------------------------------------------------------------------
```

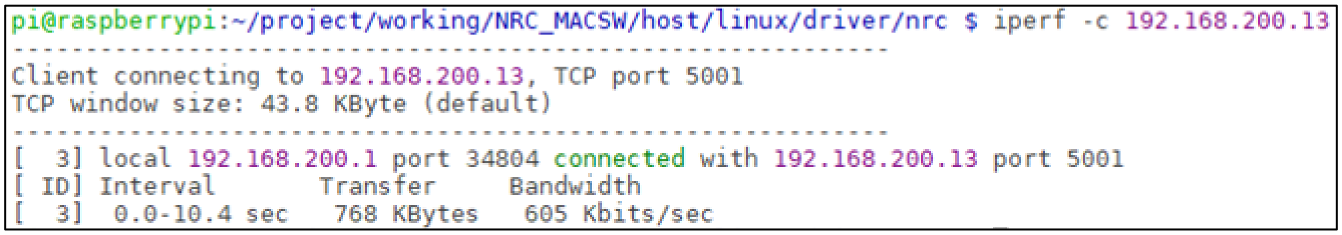
Figure 4.8 Connection message of Iperf TCP server

Figure 4.9 and Figure 4.10 show the test results for performing the Iperf TCP throughput at the standalone mode STA and the host mode AP, respectively.



```
-------------------------------------------------------------------------------
[IPERF Report] TCP Server
[IPERF Report] connected with 192.168.200.1 port 34804
[IPERF Report] 10746 msec  786468 bytes  585.0 kbps
-------------------------------------------------------------------------------
```

Figure 4.9 Test result for Iperf TCP server in standalone mode STA



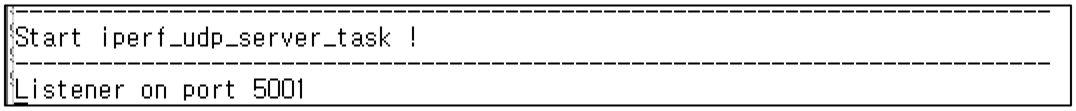
```
pi@raspberrypi:~/project/working/NRC_MACSW/host/linux/driver/nrc $ iperf -c 192.168.200.13
------------------------------------------------------------
Client connecting to 192.168.200.13, TCP port 5001
TCP window size: 43.8 KByte (default)
------------------------------------------------------------
[  3] local 192.168.200.1 port 34804 connected with 192.168.200.13 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.4 sec   768 KBytes   605 Kbits/sec
```

Figure 4.10 Test result for Iperf TCP client in host mode AP

4.4 Iperf UDP server

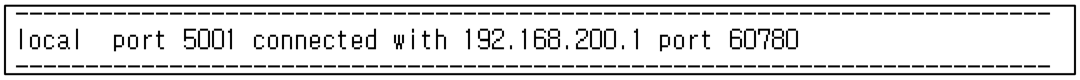
The operation and execution of the Iperf UDP server is almost the same as the Iperf TCP server except contents displayed on the UART console.

Figure 4.11 through Figure 4.14 present in sequence the start and connection message, test result for the Iperf UDP server in standalone mode STA and host mode AP.



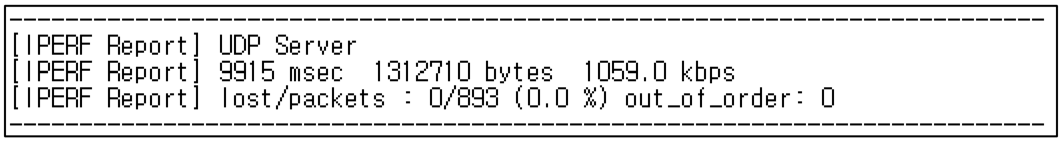
```
--------------------------------------------------------------------------
Start iperf_udp_server_task !
--------------------------------------------------------------------------
Listener on port 5001
```

Figure 4.11 Start message of Iperf UDP server in standalone mode STA



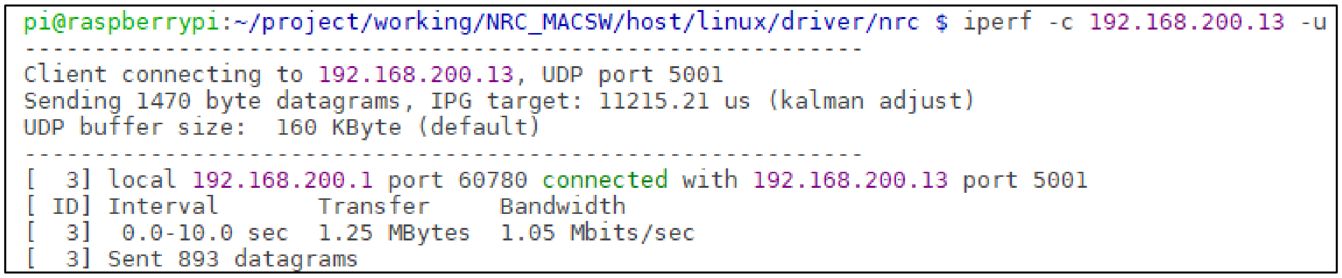
```
--------------------------------------------------------------------------
local  port 5001 connected with 192.168.200.1 port 60780
--------------------------------------------------------------------------
```

Figure 4.12 Connection message of Iperf UDP server



```
--------------------------------------------------------------------------
[IPERF Report] UDP Server
[IPERF Report] 9915 msec  1312710 bytes  1059.0 kbps
[IPERF Report] lost/packets : 0/893 (0.0 %) out_of_order: 0
--------------------------------------------------------------------------
```

Figure 4.13 Test result for Iperf UDP server in standalone mode STA



```
pi@raspberrypi:~/project/working/NRC_MACSW/host/linux/driver/nrc $ iperf -c 192.168.200.13 -u
------------------------------------------------------------
Client connecting to 192.168.200.13, UDP port 5001
Sending 1470 byte datagrams, IPG target: 11215.21 us (kalman adjust)
UDP buffer size:  160 KByte (default)
------------------------------------------------------------
[  3] local 192.168.200.1 port 60780 connected with 192.168.200.13 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec
[  3] Sent 893 datagrams
```

Figure 4.14 Test result for Iperf UDP client in host mode AP

5 Abbreviations and acronyms

Abbreviations Acronyms	Definition
ADC	Analog Digital Converter
AP	Access Point
API	Application Program Interface
AWS	Amazon Wed Service
CJSON	C JavaScript Object Notation
CoAP	Constrained Application Protocol
EVB	Evaluation Board
EVK	Evaluation Kit
FEM	Front End Module
FOTA	Firmware Over the Air
GPIO	General Purpose Input Output
HTTP	Hypertext Transfer Protocol
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
ITC	Inter-Task Communication
I2C	Inter-Integrated Circuit
LAN	Local Area Network
LwIP	Lightweight Internet Protocol
LED	Light Emitting Diode
MQTT	Message Queuing Telemetry Transport
MXML	Music Extensible Markup Language
OTA	Over-the-Air
PWM	Pulse Width Modulation
RPi3	Raspberry Pi 3
RTOS	Real Time Operating System
SDK	Software Development Kit
SoC	System on Chip
SPI	Serial Peripheral Interface
STA	Station
TCP	Transmission Control Protocol
TINYCBOR	Tiny Concise Binary Object Representation
UART	Universal Asynchronous Receive Transmitter
UDP	User Datagram Protocol
USB	Universal Serial Bus
XIP	eXecution In Place

6 Revision history

Revision No	**Date**	**Comments**
Ver 1.0	11/01/2018	Initial version for customer review created
Ver 1.1	03/25/2019	APIs and sample App for SoftAP are added
Ver 1.2	07/02/2019	Description of Sample Applications updated
Ver 1.3	11/06/2019	Update Binary download & NRC7292 SDK directories and files
Ver 1.4	07/13/2020	Added linux build environment and remove eclispe environment
Ver 1.5	09/15/2020	Added folder location for make select
Ver 1.6	12/04/2020	Update supported ubuntu image (16.04 64bit or later)
Ver 1.7	12/10/2020	Update performance evaluation
Ver 1.8	10/22/2021	Updated APIs and sample applications