



Linux PCIe MHI Driver

User Guide

Version: 1.0

Date: 2025-11-19

Status: Released



At Quectel, our aim is to provide timely and comprehensive services to our customers. If you require any assistance, please contact our headquarters:

Quectel Wireless Solutions Co., Ltd.

Building 5, Shanghai Business Park Phase III (Area B), No.1016 Tianlin Road, Minhang District, Shanghai 200233, China

Tel: +86 21 5108 6236

Email: info@quectel.com

Or our local offices. For more information, please visit:

<http://www.quectel.com/support/sales.htm>.

For technical support, or to report documentation errors, please visit:

<http://www.quectel.com/support/technical.htm>.

Or email us at: support@quectel.com.

Legal Notices

We offer information as a service to you. The provided information is based on your requirements and we make every effort to ensure its quality. You agree that you are responsible for using independent analysis and evaluation in designing intended products, and we provide reference designs for illustrative purposes only. Before using any hardware, software or service guided by this document, please read this notice carefully. Even though we employ commercially reasonable efforts to provide the best possible experience, you hereby acknowledge and agree that this document and related services hereunder are provided to you on an “as available” basis. We may revise or restate this document from time to time at our sole discretion without any prior notice to you.

Use and Disclosure Restrictions

License Agreements

Documents and information provided by us shall be kept confidential, unless specific permission is granted. They shall not be accessed or used for any purpose except as expressly provided herein.

Copyright

Our and third-party products hereunder may contain copyrighted material. Such copyrighted material shall not be copied, reproduced, distributed, merged, published, translated, or modified without prior written consent. We and the third party have exclusive rights over copyrighted material. No license shall be granted or conveyed under any patents, copyrights, trademarks, or service mark rights. To avoid ambiguities, purchasing in any form cannot be deemed as granting a license other than the normal non-exclusive, royalty-free license to use the material. We reserve the right to take legal action for noncompliance with abovementioned requirements, unauthorized use, or other illegal or malicious use of the material.

Trademarks

Except as otherwise set forth herein, nothing in this document shall be construed as conferring any rights to use any trademark, trade name or name, abbreviation, or counterfeit product thereof owned by Quectel or any third party in advertising, publicity, or other aspects.

Third-Party Rights

This document may refer to hardware, software and/or documentation owned by one or more third parties ("third-party materials"). Use of such third-party materials shall be governed by all restrictions and obligations applicable thereto.

We make no warranty or representation, either express or implied, regarding the third-party materials, including but not limited to any implied or statutory, warranties of merchantability or fitness for a particular purpose, quiet enjoyment, system integration, information accuracy, and non-infringement of any third-party intellectual property rights with regard to the licensed technology or use thereof. Nothing herein constitutes a representation or warranty by us to either develop, enhance, modify, distribute, market, sell, offer for sale, or otherwise maintain production of any our products or any other hardware, software, device, tool, information, or product. We moreover disclaim any and all warranties arising from the course of dealing or usage of trade.

Privacy Policy

To implement module functionality, certain device data are uploaded to Quectel's or third-party's servers, including carriers, chipset suppliers or customer-designated servers. Quectel, strictly abiding by the relevant laws and regulations, shall retain, use, disclose or otherwise process relevant data for the purpose of performing the service only or as permitted by applicable laws. Before data interaction with third parties, please be informed of their privacy and data security policy.

Disclaimer

- a) We acknowledge no liability for any injury or damage arising from the reliance upon the information.
- b) We shall bear no liability resulting from any inaccuracies or omissions, or from the use of the information contained herein.
- c) While we have made every effort to ensure that the functions and features under development are free from errors, it is possible that they could contain errors, inaccuracies, and omissions. Unless otherwise provided by valid agreement, we make no warranties of any kind, either implied or express, and exclude all liability for any loss or damage suffered in connection with the use of features and functions under development, to the maximum extent permitted by law, regardless of whether such loss or damage may have been foreseeable.
- d) We are not responsible for the accessibility, safety, accuracy, availability, legality, or completeness of information, advertising, commercial offers, products, services, and materials on third-party websites and third-party resources.

Copyright © Quectel Wireless Solutions Co., Ltd. 2025. All rights reserved.

About the Document

Revision History

Version	Date	Author	Description
-	2025-11-19	Schmidt Li	Creation of the document
1.0	2025-11-19	Schmidt Li	First official release

Contents

About the Document.....	3
Contents	4
Table Index	5
Figure Index	6
1 Introduction	7
1.1. Applicable Modules	7
2 Hardware Preparation and Connection	8
2.1. Module Pin Definition	8
2.2. Analysis of Key Differences Between eFuse and Non-eFuse Modules	10
2.3. PCIe Enumeration Timing	12
3 Software Architecture and Driver Mechanism	13
3.1. MHI Overview	13
3.2. MHI Driver Device Node Introduction	14
3.3. PCIe MHI NIC Driver Configuration	14
4 Driver Porting	16
4.1. Preparation Before Porting and General Process	16
4.2. Port PCIe MHI Driver to OpenWrt Platform	17
4.3. Port PCIe MHI Driver to Embedded Linux Platform	22
4.4. Verify Driver Loading	23
5 Functional Testing and Verification	25
5.1. Test AT Command Functionality.....	25
5.2. Test QMI Data Call Functionality.....	25
6 Debugging and Troubleshooting	29
6.1. Performance Optimization Suggestions	29
6.2. Troubleshooting of Common Issues	30
6.3. Issue Reporting and Log Collection	32
7 Appendix References	34

Table Index

Table 1: Applicable Modules.....	7
Table 2: Pins/Interface Used to Connect the Host and the Module.....	9
Table 3: Feature Differences Between eFuse and Non-eFuse Modules.....	10
Table 4: Driver Device Node Description	14
Table 5: MHI NIC Driver Function Mode Configuration	14
Table 6: Device Nodes Expected to Be Generated in Different Working Modes	23
Table 7: CPU Bitmask Reference Table.....	30
Table 8: Related Documents	34
Table 9: Terms and Abbreviations	34

Figure Index

Figure 1: Connection Between the Host and the Module	8
Figure 2: PCIe Enumeration Timing	12
Figure 3: MHI Logical Architecture	13

1 Introduction

This document describes how to integrate the Quectel PCIe MHI driver into the Linux operating system of a target host device, and how to verify and test module functionality after successful driver deployment.

1.1. Applicable Modules

Table 1: Applicable Modules

Product Line	Module
5G	RG500Q
	RG501Q
	RG502Q
	RM500Q
	RM502Q
	RM505Q
	RM510Q
	RG520N
	RM520N
	RM530N
	RM255C
LTE-A	EM120R
	EM160R

2 Hardware Preparation and Connection

This chapter introduces the hardware interface design between the host and the module (also known as the device) based on the PCIe MHI hardware integration architecture of Quectel modules, and analyzes the differences in key functions between eFuse and non-eFuse modules, providing a hardware design basis for subsequent driver porting and functional verification.

The main content includes:

- **Physical Connection:** The PCIe pin definition and functional interaction logic between the host and the module, covering key interfaces such as power control, reset signal, and data transmission channel.
- **eFuse/Non-eFuse Module Difference Analysis:** Core differences between eFuse and non-eFuse modules in enumeration timing, firmware upgrade, RAM Dump capture, and other functions.
- **Enumeration Process:** The timing requirements and signal interaction mechanism of PCIe link training, clarifying the collaborative workflow between the host and the module.

2.1. Module Pin Definition

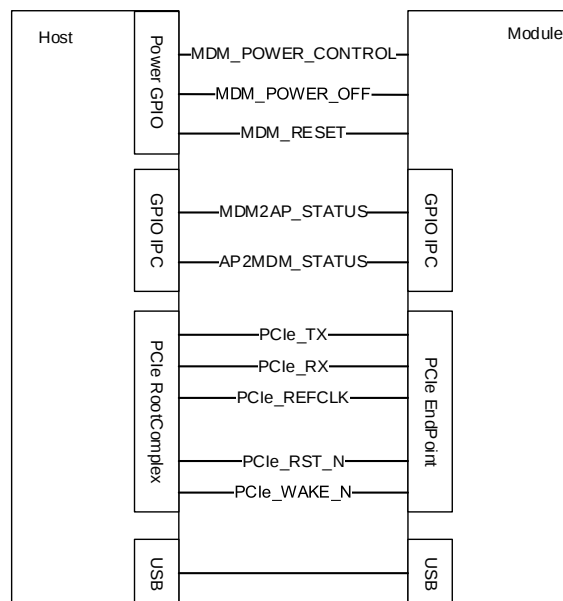


Figure 1: Connection Between the Host and the Module

Table 2: Pins/Interface Used to Connect the Host and the Module

Pin/Interface Name	Function
MDM_POWER_CONTROL	Used to transmit the power enablement control signal. The host controls the module's power-supply through the signal.
MDM_POWER_OFF	Used to transmit the module power-off control signal. The host triggers the module's power-on/off by pulling this pin high/low.
MDM_RESET	Used to transmit the module soft reset signal. The host triggers a soft reset of the module through the signal.
MDM2AP_STATUS	Used to transmit the status notification signal from the module to the host. The module notifies the host to reboot through the signal upon a crash.
AP2MDM_STATUS	Used to transmit the status notification signal from the host to the module. The host notifies the module to enter safe mode or prepare for reboot through the signal upon a crash, ensuring status consistency between the module and the host.
PCIe_TX	Used to transmit PCIe differential TX signals (TX+/TX-). Each channel consists of a pair of differential signals for the host to send data to the module.
PCIe_RX	Used to transmit PCIe differential RX signals (RX+/RX-). Each channel consists of a pair of differential signals for the host to receive data from the module.
PCIe_REFCLK	Used to transmit PCIe reference clock signals (REFCLK+/REFCLK-), which is a pair of differential clock signals provided by the host to supply a common reference clock for the PCIe link.
PCIe_RST_N	Used to transmit the PCIe reset signal (PERST#, active low), which is asserted by the host to perform a hardware reset on the module during power-on, hot reset, or wake-up.
PCIe_WAKE_N	Used to transmit the wake-up request signal (WAKE#, active low, open-drain output). When the module is in a low-power status, it uses the signal to send a wake-up request to the host.
USB	USB interface, through which the host communicates with the module to perform operations such as AT command interactions, diagnostic log capture, and firmware upgrades.

Figure 1 demonstrates a diagram of the connection between the host and the module, and **Table 2** describes the functions of the pins used to connect the host and the module. However, this chapter only helps you to understand the functional interaction between the host and the module. If you want to know the actual hardware connection details, please contact Quectel Technical Support to obtain the documents for hardware design.

NOTE

Whether the USB is required depends on whether the module is an eFuse module. Refer to **Chapter 2.2** for details.

2.2. Analysis of Key Differences Between eFuse and Non-eFuse Modules

This section explains the key differences in underlying boot architecture and functional support between eFuse and non-eFuse versions of Quectel modules. The essential difference lies in the timing of PCIe controller initialization, which directly affects the interaction timing and feature set between the module and the host.

- eFuse module: The module that enables the PCIe Early Init function. This function enables the module to complete PCIe initialization in the PBL phase by blowing the PCIE_EARLY_INIT bit inside the module.
- Non-eFuse module: The module that completes PCIe initialization during the kernel startup phase.

The feature differences of the two types of modules are listed below.

Table 3: Feature Differences Between eFuse and Non-eFuse Modules

Feature	eFuse Module	Non-eFuse Module
PCIe Initialization Timing	PBL phase	Kernel startup phase
Compatibility with X86 BIOS Timing	Supported The module meets the PC-compliant PCIe enumeration timing requirements and can be identified as a standard PCIe device in the BIOS phase.	Not supported
Boot-up Coordination	The module waits for a successful PCIe link training before continuing the boot process in the PBL phase.	This module reserves a window of about 30 seconds for PCIe link establishment in the kernel startup phase.
PCIe Controller Awareness of Module Reboot	Supported	Not supported It is recommended to use the <code>/sys/bus/pci/remove</code> and <code>/sys/bus/pci/rescan</code> nodes to re-enumerate the PCIe device. This method requires the host to actively control the module's reboot and be able to be aware

				of the module's reboot.
RAM Dump Capture via PCIe	Supported			Not supported
Firmware Upgrade via PCIe	FullFOTA supported			FullFOTA not supported but DFOTA supported. Non-eFuse modules require a USB 2.0 interface for FullFOTA upgrades. The module will reboot after a DFOTA upgrade, and the host PCIe enumeration issue needs to be solved through scripts and other methods.
QXDM log Capture via PCIe	Supported			Not supported
USB Interface Dependency	Optional All software functions can be realized via PCIe, without mandatory connection of the USB 2.0 interface.			Required PCIe does not support firmware upgrade and RAM Dump capture, so you must connect the USB 2.0 interface to support operation and maintenance functions.
PCIe Wake-up from Sleep	Supported			Not supported

According to the information above, you can get the following module selection and integration guidance.

- For X86 PC platforms: eFuse module must be selected.
- For embedded Linux platforms: Either type of module can be selected based on actual requirements.
 - If you select an eFuse module: All key functions can be realized via PCIe, without mandatory connection of the USB 2.0 interface.
 - If you select a non-eFuse module: PCIe functionality is limited, and a USB 2.0 interface must be externally connected to support firmware upgrades and log capture.

If you use a non-eFuse module, you need to execute the following AT commands to enable the PCIe interface and set the working mode.

- Set PCIe EP mode: **AT+QCFG="pcie/mode",0**
- Set the network port communication via PCIe interface and diagnostic port communication via USB interface: **AT+QCFG="data_interface",1,0**

NOTE

1. To confirm whether the current module has eFuse enabled, please contact Quectel Technical Support.

2. For details about **AT+QCFG="pcie/mode"** and **AT+QCFG="data_interface"**, refer to **documents [2], [3], [4] and [5]**.

2.3. PCIe Enumeration Timing

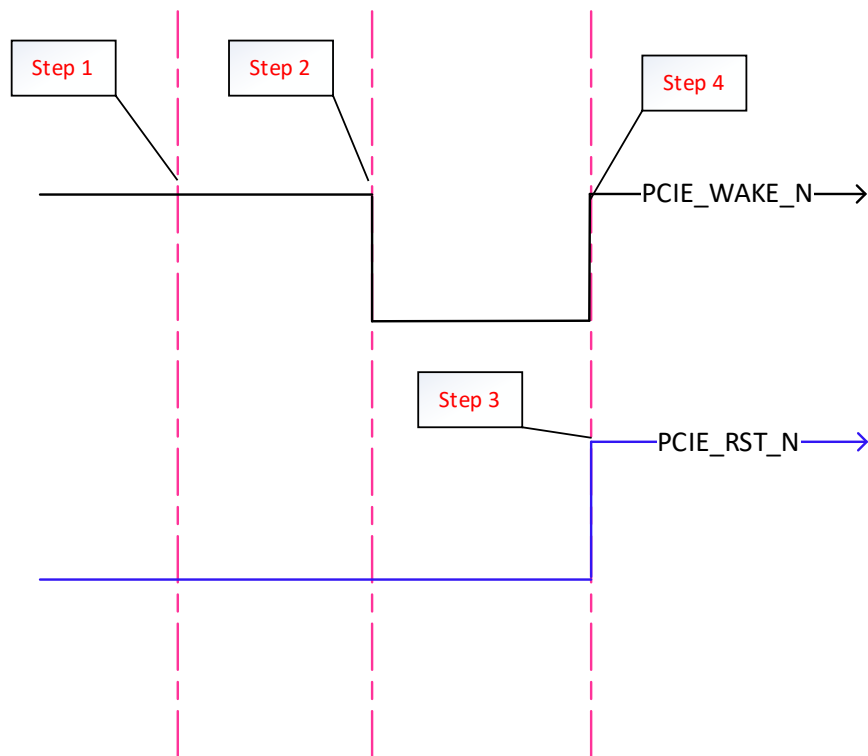


Figure 2: PCIe Enumeration Timing

The above figure shows the standard process of PCIe link establishment and device enumeration between the module and the host.

Step 1: Module boots up.

Step 2: After the PCIe initialization of the module is completed, the module pulls down **PCIE_WAKE_N** to notify the host.

Step 3: The host pulls up **PCIE_RST_N** to start PCIe enumeration.

Step 4: After receiving the signal that **PCIE_RST_N** has been pulled up, the module pulls up **PCIE_WAKE_N** to start PCIe enumeration.

The above process ensures that the module's PCIe is successfully enumerated. However, please note that in some embedded systems, especially during the U-Boot phase, **PCIE_RST_N** might be pulled up too early, causing the module to incorrectly enter the enumeration program. When the host enters the kernel startup phase later and initiates PCIe enumeration again, enumeration failure may occur.

3 Software Architecture and Driver Mechanism

3.1. MHI Overview

MHI (Modem Host Interface) is a standardized host-modem communication protocol, which aims to establish a high-throughput, low-latency communication channel between the host AP and the module modem over high-speed serial interfaces (such as PCIe or USB), for transmitting control signaling, user-space data, and diagnostic information.

MHI abstracts multiple logical channels at the physical link layer, including MHI Ctrl, MHI IP Data, and MHI Diag, which are respectively used to transmit control commands, IP packets, diagnostic logs and other different types of data flows to achieve zero-copy and low-latency communication.

The Quectel PCIe MHI driver is developed based on the Linux kernel's standard MHI bus framework (mhi_bus), adapted and optimized for the hardware characteristics and firmware behavior of Quectel modules, possessing good cross-platform compatibility. Combined with Quectel's self-developed user-space tools (such as quectel-CM, QFlash, etc.), the host can realize network connection management, firmware upgrades, diagnostic log capture, and other functions for the module via the MHI channel.

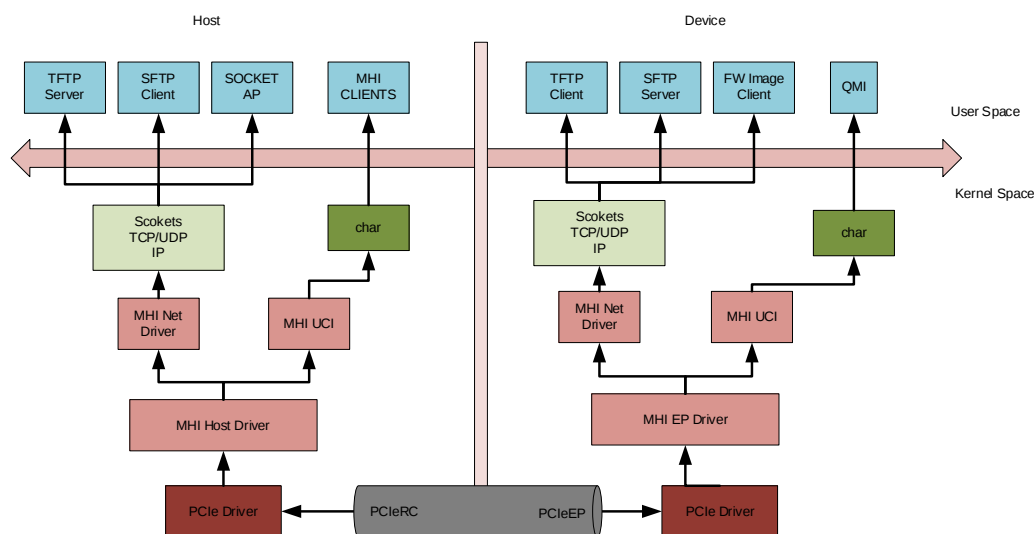


Figure 3: MHI Logical Architecture

- PCIe RC: The host in the PCIe architecture.
- PCIe EP: The device in the PCIe architecture.
- PCIe driver: The driver is developed by the chip manufacturer and is responsible for device initialization, data transmission, resource management, and implementation of advanced features.
- MHI driver: On the basis of the PCIe bus, the driver constructs logical channels such as RmNet NIC and mhi* character devices, realizing the of control of cellular modules, transmission of network data and diagnostic messages.

3.2. MHI Driver Device Node Introduction

Table 4: Driver Device Node Description

Driver Device Node Name	Interface Type	Function Description
<i>rmnet_mhi0.x</i>	Network Interface	Used to send and receive network packets.
<i>/dev/mhi_BHI</i>	Firmware Upgrade Interface	Supported only by eFuse modules, used for firmware flashing and upgrade.
<i>/dev/mhi_DIAG</i>	Diagnostic Log Interface	Used to obtain diagnostic logs from the module.
<i>/dev/mhi_DUN</i>	AT Command Interface	Used to send and receive AT commands.
<i>/dev/mhi_LOOPBACK</i>	NMEA Data Interface	Used for sending and receiving NMEA sentences.
<i>/dev/mhi_QMIO</i>	QMI Message Interface	Used to send and receive QMI messages.
<i>/dev/mhi_MBIM</i>	MBIM Message Interface	Used to send and receive MBIM messages.
<i>/dev/mhi_SAHARA</i>	RAM Dump Interface	Supported only by eFuse modules, used to capture module dumps.

3.3. PCIe MHI NIC Driver Configuration

Table 5: MHI NIC Driver Function Mode Configuration

Mode	Function	Configuration Method and Description
Single-NIC	Register only the <i>rmnet_mhi0</i> NIC. The NIC is responsible for processing QMAP packets,	Enable the <i>MHI_NETDEV_ONE_CARD_MODE</i> macro in the <i>mhi_netdev_quecte</i>

	splitting them into multiple IP packets, and submitting them to the kernel protocol stack.	<i>.c</i> file. Disabled by default.
Multi-NIC	Register multiple NICs: ● <code>rmnet_mhi0</code>: Used to receive QMAP packets and split them into IP packets. ● <code>rmnet_mhi0.x[1-8]</code>: Multiple virtual NICs, which are responsible for submitting split IP packets to the kernel protocol stack. This mode supports multiple data calls, up to 8 channels. The created NIC corresponds to <code>rmnet_mhi0.x[1-8]</code>, and any one or more channels can be configured as bridge mode.	Enable <code>static uint __read_mostly qmap_mode = N</code> ($N = 1-8$, which indicates the number of NICs to be created) in the <code>mhi_netdev_quectel.c</code> file. Example: <code>qmap_mode = 4</code> indicates creating 4 NICs. Enabled by default with <code>qmap_mode = 1</code>.
Bridge	Configure the specified virtual NIC to bridge mode. In this mode, the network card acts as a transparent bridge device, forwarding Ethernet frames based on MAC addresses so that the downlinked devices seem to be directly connected to the upper-level network.	Enable the <code>QUECTEL_BRIDGE_MODE</code> macro and enable <code>static uint __read_mostly bridge_mode = BIT(N)</code> (N indicates the NIC index) in the <code>mhi_netdev_quectel.c</code> file. Example: <code>bridge_mode = BIT(1)</code> indicates configuring <code>rmnet_mhi0.1</code> as bridge mode. Disabled by default.
QMI Port	Support sending QMI messages to the MDM via the <code>/dev/mhi_QMIO</code> device node.	Enabled by default.
MBIM Port	Support sending MBIM messages to the MDM via the <code>/dev/mhi_MBIM0</code> device node.	Enable <code>static uint __read_mostly mhi_mbim_enabled = 1</code> in the <code>mhi_netdev_quectel.c</code> file. Disabled by default.

As described in the table above, the PCIe MHI driver supports three NIC modes and applicable scenarios are as follows:

- Single-NIC mode: Suitable for hosts with only a single CPU, limited computing capability, or single application scenarios without complex network policies.
- Multi-NIC mode: Suitable for hosts with multiple CPUs, allowing load adjustment to fully leverage NIC performance.
- Bridge mode: Suitable for devices like routers and gateways, enabling connected devices to directly access the upper-layer network.

4 Driver Porting

4.1. Preparation Before Porting and General Process

This section details how to port the Quectel PCIe MHI driver to a target Linux system. The general process should be followed regardless of whether the target platform is OpenWrt or another embedded Linux OS.

1. Host System Requirements

1) PCIe and Interrupt Requirements

- MSI/MSI-X Support: The host platform must support the PCIe MSI or MSI-X mechanism.
- Number of Interrupt Vectors: The system should provide **at least one** independent MSI vector. For optimal performance and functional integrity, it is strongly recommended to support multiple interrupt vectors to allocate independent interrupts for different transmission channels or event types.
- Kernel Configuration: You must ensure that **CONFIG_PCI_MSI** is enabled when compiling the kernel. This option is the fundamental dependency for PCIe MSI interrupt functionality.

2) Kernel Version Compatibility

- Supported versions: Linux kernel versions 3.10 to 6.12.

2. Prerequisite: Host System Can Normally Enumerate PCIe Devices

Before loading the MHI driver, you must ensure that the host system can identify and enumerate the module through the physical layer.

Verification method: Execute the following **lspci** command. This command can list all PCIe devices on the bus, including the Quectel module.

Successful enumeration (taking the RM520N-GL module mounted on host as an example):

```
root@OpenWrt:/# lspci
01:00.0 Class ff00: 17cb:0308
00:00.0 Class 0604: 17cb:1004
```

17cb:0308 is the PID and VID of the RM520N-GL module. The returned information indicates that the module has been successfully enumerated.

3. General Porting Process

The following is a general process for porting the Quectel PCIe MHI driver to a target system.

Step 1: Verify PCIe Enumeration

Execute the **lspci** command to confirm the host system can normally enumerate PCIe devices.

Step 2: Obtain Driver Source Codes

Contact Quectel Technical Support to obtain the latest version of the PCIe MHI driver source code package.

Step 3: Configure DTS

Configure the PCIe controller and MHI nodes according to the host platform.

Step 4: Integrate Driver into Kernel

Integrate the driver source code into the kernel source tree or build system.

Step 5: Avoid Driver Conflicts

Disable the native MHI driver provided by the SoC manufacturer to prevent it from occupying the device.

Step 6: Compile, Download, and Verify

Compile the system image, download it, and check whether the driver node is generated.

4.2. Port PCIe MHI Driver to OpenWrt Platform

This section takes the third-party IPQ series platform as an example, detailing how to port the Quectel PCIe MHI driver to an OpenWrt system.

Quectel provides a resource package named *quectel_qsdk.zip*, and its directory structure is as follows:

```
├── qsdk
│   ├── package
│   │   └── quectel
│   │       ├── feeds
│   │       │   ├── pcie_mhi
│   │       │   ├── QFirehose
│   │       │   ├── QLog
│   │       │   ├── qmi_wwan_q
│   │       │   └── quectel-CM
│   │       └── src
│   │           ├── pcie_mhi
│   │           ├── QFirehose
│   │           ├── QLog
│   │           ├── qmi_wwan_q
│   │           └── quectel-CM
│   └── qca
│       └── src
```

```
| | | | datarmnet
| | | | | 0001-rmnet-nss-support-qmapv1.patch
| | | | | 0002-rmnet-newlink-update-ul-aggr.patch
| | | | | linux-4.4
| | | | | | patches
| | | | | linux-5.4
| | | | | | patches
| | | | ql-build.sh
| | | | | quectel.add.config
| | | | | | readme.txt
```

For more details about the source package, please read the *readme.txt* file in the package.

NOTE

1. Please contact Quectel Technical Support to obtain the latest resource package *quectel_qsdk.zip*.
2. If the target platform for porting is an OpenWrt platform not based on IPQ host, there is no need to use the patches under the *qca/src/* directory within the resource package, while the remaining resources are still applicable.

● Porting Steps

Step 1: Ensure that the PCIe controller and related MHI nodes are correctly enabled in the DTS file of the host platform.

- 1) Check the definitions of *pcie_x1*, *pcie_x1phy* and *pcie_x1_rp* in the DTS file. Usually, *pcie_x1*, *pcie_x1phy* and *pcie_x1_rp* are defined in the platform DTS, and you only need to ensure *status = "ok"*.
- 2) Configure the IOMMU access method of the PCIe as *qcom,iommu-dma = "disabled"*.

NOTE

IOMMU access methods in different host platforms are variable. Please refer to the platform-specific codes for specific configurations.

Here takes *qcom-ipq5018-mp03.6-c1.dts* on the IPQ5018 platform as an example.

```
&pcie_x1 {
    .....
    status = "ok";
    .....
};
```

```
&pcie_x1phy {
    status = "ok";
};
&pcie_x1_rp {
    status = "ok";
    .....

    mhi_0: qcom,mhi@0 {
        reg = <0 0 0 0 0 >;
        .....
        /* controller specific configuration */
        qcom,iommu-dma = "disabled";
        .....
    }
```

Step 2: Copy *qsdk/package/quectel/src/pcie_mhi* in *quectel_qsdk.zip*, the source code of PCIe MHI driver, to the IPQ5018 QSDK source code directory *qsdk/package/quectel/src*.

Step 3: Create a Makefile in the IPQ5018 QSDK source code directory *qsdk/package/quectel/feeds/pcie_mhi/*.

```
include $(TOPDIR)/rules.mk
include $(INCLUDE_DIR)/kernel.mk

PKG_NAME:=pcie_mhi
PKG_RELEASE:=1

#include $(INCLUDE_DIR)/local-development.mk
ifeq ($(DUMP)$(PKG_VERSION),)
PKG_VERSION:=1.0
endif

QCA_MC_SNOOPING_MAKE_OPTS:= \
    CROSS_COMPILE=$(KERNEL_CROSS) \
    ARCH=$(LINUX_KARCH) \
    KERNELPATH=$(LINUX_SRC_DIR) \
    KBUILDPATH=$(LINUX_DIR) \
    KERNELRELEASE=$(LINUX_RELEASE) \
    MDIR=$(PKG_BUILD_DIR)

include $(INCLUDE_DIR)/package.mk

define KernelPackage/pcie_mhi
    SECTION:=kernel
    CATEGORY:=Kernel modules
    SUBMENU:=Network Support
```

```
URL:=http://www.qca.qualcomm.com
MAINTAINER:=Qualcomm Atheros, Inc.
TITLE:=Quectel Linux PCIE MHI Driver
#DEPENDS+=kmod-rmnet-core
KCONFIG:=
FILES:=$(PKG_BUILD_DIR)/pcie_mhi.ko
AUTOLOAD:=$(call AutoLoad,53,pcie_mhi)
endif

define KernelPackage/pcie_mhi/description
    Quectel Linux PCIE MHI Driver
endif

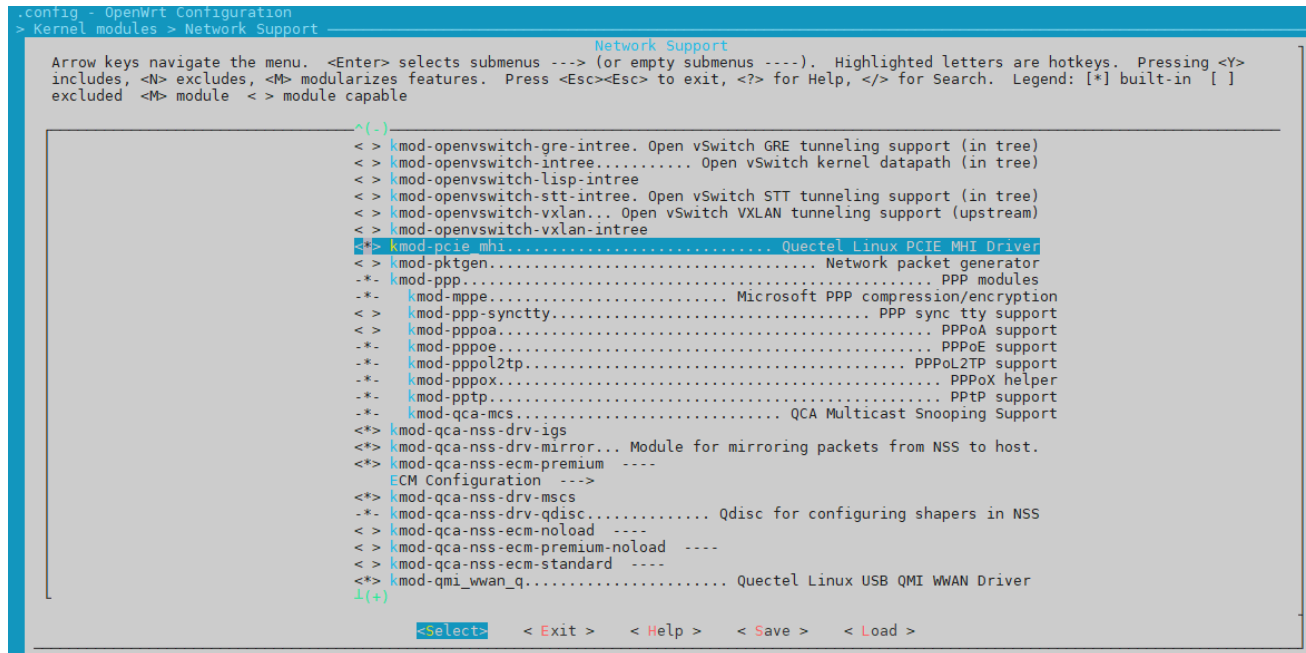
define Build/InstallDev
endif

define Build/Prepare
    mkdir -p $(PKG_BUILD_DIR)
    $(CP) $(TOPDIR)/package/quectel/src/${PKG_NAME}/* $(PKG_BUILD_DIR)/
endif

define Build/Compile
    $(MAKE) -C $(LINUX_DIR) M=$(PKG_BUILD_DIR) $(strip
$(QCA_MC_SNOOPING_MAKE_OPTS)) \
    EXTRA_CFLAGS="-I$(STAGING_DIR)/usr/include/qca-nss-driv $(EXTRA_CFLAGS)"
endif

$(eval $(call KernelPackage,pcie_mhi))
```

Step 4: Execute **make menuconfig** to enter the curses-based graphical configuration interface, and enable **<*> kmod-pcie-mhi** to enable the compilation of the PCIe MHI driver.



```
.config - OpenWrt Configuration
> Kernel modules > Network Support

Arrow keys navigate the menu. <Enter> selects submenus --- (or empty submenus ----). Highlighted letters are hotkeys. Pressing <Y>
includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*] built-in [ ]
excluded <M> module <> module capable

^(-)
<> kmod-openvswitch-gre-intree. Open vSwitch GRE tunneling support (in tree)
<> kmod-openvswitch-intree..... Open vSwitch kernel datapath (in tree)
<> kmod-openvswitch-lisp-intree
<> kmod-openvswitch-stt-intree. Open vSwitch STT tunneling support (in tree)
<> kmod-openvswitch-vxlan... Open vSwitch VXLAN tunneling support (upstream)
<> kmod-openvswitch-vxlan-intree
[*] kmod-pcie-mhi..... Quectel Linux PCIE MHI Driver
<> kmod-pktgen..... Network packet generator
<M> kmod-ppp..... PPP modules
-* kmod-mppe..... Microsoft PPP compression/encryption
<> kmod-ppp-synctty..... PPP sync tty support
<> kmod-pppoe..... PPPoE support
-* kmod-pppoe..... PPPoE support
-* kmod-pppol2tp..... PPPoL2TP support
-* kmod-pppox..... PPPoX helper
-* kmod-pptp..... PPTP support
-* kmod-qca-mcs..... QCA Multicast Snooping Support
<*> kmod-qca-nss-driv-igs
<*> kmod-qca-nss-driv-mirror... Module for mirroring packets from NSS to host.
<*> kmod-qca-nss-ecm-premium ----
ECM Configuration --->
<*> kmod-qca-nss-driv-mscs
-* kmod-qca-nss-driv-qdisc..... Qdisc for configuring shapers in NSS
<> kmod-qca-nss-ecm-noload ----
<> kmod-qca-nss-ecm-premium-noload ----
<> kmod-qca-nss-ecm-standard ----
<*> kmod-qmi_wwan_q..... Quectel Linux USB QMI WWAN Driver
i(+)

<Select> < Exit > < Help > < Save > < Load >
```

After the above steps are completed, the PCIe MHI driver will be compiled into the kernel as a KO.

Step 5: Disable the native MHI driver included in the platform (This step only applies to platforms where the platform includes a native MHI driver. Please ignore for other platforms).

Find the `linux-xxx/drivers/bus/mhi` directory, disable support for specific PID/VID devices in the kernel configuration or source code, that is, comment out the corresponding device entries in the `mhi_pcie_device_id[]` array, so that the driver will not be loaded.

```
static struct pci_device_id mhi_pcie_device_id[] = {
/*
{PCI_DEVICE(MHI_PCIE_VENDOR_ID, 0x0300)},
{PCI_DEVICE(MHI_PCIE_VENDOR_ID, 0x0301)},
{PCI_DEVICE(MHI_PCIE_VENDOR_ID, 0x0303)},
{PCI_DEVICE(MHI_PCIE_VENDOR_ID, 0x0304)},
{PCI_DEVICE(MHI_PCIE_VENDOR_ID, 0x0305)},
{PCI_DEVICE(MHI_PCIE_VENDOR_ID, 0x0306)},
{PCI_DEVICE(MHI_PCIE_VENDOR_ID, 0x0308)},
*/
{PCI_DEVICE(MHI_PCIE_VENDOR_ID, MHI_PCIE_DEBUG_ID)},
{0},
};
```

After all the above steps are completed, the driver porting is finished, and you can download the firmware for verification.

4.3. Port PCIe MHI Driver to Embedded Linux Platform

This section takes the platform equipped with Quectel SG865W series modules (hereinafter referred to as the "SG865W platform") as an example, detailing how to port the Quectel PCIe MHI driver to an embedded Linux system.

Step 1: Correctly configure PCIe-related resources and enable them in the DTS file (Please note that the IOMMU access method of PCIe should be configured as *iommu-dma = "fastmap"*).

```
&pcie1_rp {
    .....
    /* controller specific configuration */
    qcom,iommu-group = <&mhi_0_iommu_group>;
    mhi_0_iommu_group: mhi_0_iommu_group {
        qcom,iommu-dma-addr-pool = <0x20000000 0x1ffffff>;
        qcom,iommu-dma = "fastmap";
        qcom,iommu-pagetable = "coherent";
    };
    .....
}
```

NOTE

IOMMU access methods in different host platforms are variable. Please refer to the platform-specific codes for specific configurations.

Step 2: Copy *qsdk/package/quectel/src/pcie_mhi* in *quectel_qsdk.zip*, the source code of PCIe MHI driver, to the SG865W platform source code directory *kernel/driver/*. The PCIe MHI driver will be compiled into the kernel as a KO by default.

Step 3: Disable the native MHI driver included in the platform.

Find the *kernel/msm-xxx/drivers/bus/mhi* directory, disable support for specific PID/VID devices in the kernel configuration or source code, that is, comment out the corresponding device entries in the *mhi_pcie_device_id[]* array, so that the driver will not be loaded.

```
static struct pci_device_id mhi_pcie_device_id[] = {
    /*
     {PCI_DEVICE(MHI_PCIE_VENDOR_ID, 0x0300)},
     {PCI_DEVICE(MHI_PCIE_VENDOR_ID, 0x0301)},
     {PCI_DEVICE(MHI_PCIE_VENDOR_ID, 0x0302)},
     {PCI_DEVICE(MHI_PCIE_VENDOR_ID, 0x0303)},
     {PCI_DEVICE(MHI_PCIE_VENDOR_ID, 0x0304)},
     {PCI_DEVICE(MHI_PCIE_VENDOR_ID, 0x0305)},
     //{PCI_DEVICE(MHI_PCIE_VENDOR_ID, 0x0306)},
    */
    {PCI_DEVICE(MHI_PCIE_VENDOR_ID, MHI_PCIE_DEBUG_ID)},
    {0},
};
```

After the above steps are completed, recompile the kernel and rootfs, and you can download the firmware for verification.

NOTE

If you need to compile both the Quectel PCIe MHI driver and the chip manufacturer's native MHI driver into the kernel simultaneously and avoid conflicts, you must rename functions in the PCIe MHI driver code. Please contact Quectel Technical Support for modification suggestions.

4.4. Verify Driver Loading

After the driver porting is successful, you can verify whether the driver loads normally and creates a device node in the following methods.

Method 1: Check Device Nodes

According to the different working modes of the module, the MHI driver creates different device nodes, which can be used as one of the bases for determining the success of driver loading. You can execute **\$ ls /dev/mhi*** to view the return value. The device nodes expected to be generated in different working modes are shown in the following table.

Table 6: Device Nodes Expected to Be Generated in Different Working Modes

Working Mode	Device Nodes Expected to Be Generated			
QMI Mode	/dev/mhi_BHI /dev/mhi_QMIO	/dev/mhi_DIAG	/dev/mhi_DUN	/dev/mhi_LOOPBACK
MBIM Mode	/dev/mhi_BHI /dev/mhi_MBIMI	/dev/mhi_DIAG	/dev/mhi_DUN	/dev/mhi_LOOPBACK
Dump Mode	/dev/mhi_BHI	/dev/mhi_SAHARA		

Method 2: Check Network Interface

Execute **ifconfig** to check the NIC created by the MHI driver.

```
ifconfig
rmnet_mhi0: flags=128<NOARP>  mtu 1500
    unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00  txqueuelen 1000  (UNSPEC)
    RX packets 0  bytes 0 (0.0 B)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 0  bytes 0 (0.0 B)
    TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

```
rmnet_mhi0.1: flags=128<NOARP> mtu 1500
    ether 02:50:f4:00:00:01 txqueuelen 1000 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

rmnet_mhi0 and *rmnet_mhi0.x* displayed in the return value indicate that the driver loads successfully.

5 Functional Testing and Verification

This chapter details how to test and verify the basic functions and data connections after the PCIe MHI driver has been ported and successfully loaded.

5.1. Test AT Command Functionality

After the driver is loaded, the corresponding MHI driver node `/dev/mhi_DUN` is created. This node serves as the AT command port, which can be used to send and receive AT commands.

Use a Linux serial terminal tool, such as minicom or busybox microcom, to open this device node and send an AT command for testing. If "OK" is returned, the command has been executed successfully.

Example:

```
root@OpenWrt:/# microcom /dev/mhi_DUN
ATI
Quectel
RM520N-GL
Revision: RM520NGLAAR03A04M4G

OK
```

5.2. Test QMI Data Call Functionality

Quectel provides the QConnectManager (quectel-CM) tool, which supports QMI protocol data call for establishing and managing cellular network data connections. To obtain the latest version of the QConnectManager toolkit, please contact Quectel Technical Support. For detailed information about QConnectManager, please refer to **document [1]**.

Step 1: Compile quectel-CM

Obtain the project files and source code from the QConnectManager toolkit. Linux operating system users can compile QConnectManager based on the project files and source code. Execute the following compilation commands to generate the executable program *quectel-CM*.

- For X86 Linux PC

```
# make
```

- For embedded Linux platforms

```
# make CROSS-COMPILE=arm-none-linux-gnueabi-
```

NOTE

Replace "arm-none-linux-gnueabi-" with the cross-compilation toolchain of the target platform to ensure that the generated executable program can run on the target platform.

Step 2: Prepare busybox udhcpc tool.

quectel-CM calls the busybox udhpc tool to get the IP address and DNS server. After receiving the network configuration information, busybox udhpc will not directly configure the system, but will call a predefined script file to complete these operations. The default path for this script is `/usr/share/udhcpc/default.script`.

To ensure that the data connection is properly configured for the network, you need to:

1. Download busybox udhpc source code from <https://busybox.net/> and enable `CONFIG_UDHCPC` using the following command:

```
busybox menuconfig
```

2. Copy the script file `[BUSYBOX]/examples/udhcp/simple.script` into your Linux system and rename it to `/usr/share/udhcpc/default.script`.

Step 3: Use quectel-CM to establish a QMI data connection. For detailed information on quectel-CM parameters, please refer to **document [1]**.

Prerequisites:

- The kernel has successfully connected and the PCIe MHI driver has been loaded.
- The device node `/dev/mhi_QMI0` already exists.
- The NIC `rmnet_mhi0.1` has been created.

Data call and IP acquisition example (taking the RM520N-GL module as an example):

```
root@OpenWrt:/# quectel-CM &
root@OpenWrt:/# [05-28_01:38:39:883] QConnectManager_Linux_V1.6.6
[05-28_01:38:39:884] network interface " or qmidev " is not exist
[05-28_01:38:39:887] netcard driver = pcie_mhi, driver version = V1.3.6
[05-28_01:38:39:888] qmap_mode = 1, qmap_version = 9, qmap_size = 15360, muxid = 0x81,
```

```
qmap_netcard = rmnet_mhi0.1
[05-28_01:38:39:892] Modem works in QMI mode
[05-28_01:38:39:925] cdc_wdm_fd = 7
[05-28_01:38:39:990] Get clientWDS = 15
[05-28_01:38:39:994] Get clientDMS = 1
[05-28_01:38:39:999] Get clientNAS = 3
[05-28_01:38:40:004] Get clientUIM = 2
[05-28_01:38:40:009] Get clientWDA = 1
[05-28_01:38:40:015] requestBaseBandVersion RM520NGLAAR03A04M4G
[05-28_01:38:40:019] qmap_settings.rx_urb_size = 15360
[05-28_01:38:40:020] qmap_settings.ul_data_aggregation_max_datagrams = 11
[05-28_01:38:40:020] qmap_settings.ul_data_aggregation_max_size = 8192
[05-28_01:38:40:020] qmap_settings.dl_minimum_padding = 0
[05-28_01:38:40:041] requestGetSIMStatus SIMStatus: SIM_READY
[05-28_01:38:40:053] requestGetProfile[pdp:1 index:1] ctnet///0/IPV4V6
[05-28_01:38:40:058] requestRegistrationState2 MCC: 460, MNC: 11, PS: Attached, DataCap: 5G_SA
[05-28_01:38:40:064] requestQueryDataCall IPv4ConnectionStatus: DISCONNECTED
[05-28_01:38:40:067] ip link set dev rmnet_mhi0 down
[05-28_01:38:40:075] ip addr flush dev rmnet_mhi0.1
[05-28_01:38:40:085] ip link set dev rmnet_mhi0.1 down
[05-28_01:38:40:353] requestSetupDataCall WdsConnectionIPv4Handle: 0xe2a0fe20
[ 3987.451331] net rmnet_mhi0: link_state 0x0 -> 0x1
[05-28_01:38:40:367] ip link set dev rmnet_mhi0 up
[ 3987.464330] [I][mhi_netdev_open] Opened net dev interface
[05-28_01:38:40:385] ip link set dev rmnet_mhi0.1 up
[05-28_01:38:40:400] busybox udhcpc -f -n -q -t 5 -i rmnet_mhi0.1
udhcpc: started, v1.35.0
udhcpc: broadcasting discover
udhcpc: broadcasting select for 100.218.109.193, server 100.218.109.194
udhcpc: lease of 100.218.109.193 obtained from 100.218.109.194, lease time 7200
[05-28_01:38:40:705] udhcpc: ifconfig rmnet_mhi0.1 100.218.109.193 netmask 255.255.255.252
broadcast +
[05-28_01:38:40:721] udhcpc: setting default routers: 100.218.109.194
```

Step 4: Execute **# ifconfig rmnet_mhi0.1** to verify the configuration information of IP address, DNS server and route.

```
root@OpenWrt:/# ifconfig rmnet_mhi0.1
rmnet_mhi0.1 Link encap:UNSPEC HWaddr 02-50-F4-00-00-01-68-F9-00-00-00-00-00-00-00-00
    inet addr:100.218.109.193 Mask:255.255.255.252
    inet6 addr: fe80::50:f4ff:fe00:1/64 Scope:Link
    UP RUNNING NOARP MTU:1500 Metric:1
    RX packets:4 errors:0 dropped:0 overruns:0 frame:0
    TX packets:19 errors:0 dropped:0 overruns:0 carrier:0
    collisions:0 txqueuelen:1000
```

RX bytes:1242 (1.2 KiB) TX bytes:3068 (2.9 KiB)

```
root@OpenWrt:/# cat /etc/resolv.conf
search lan
nameserver 127.0.0.1
```

```
root@OpenWrt:/# ip route show
default via 100.218.109.194 dev rmnet_mhi0.1 proto static src 100.218.109.193
100.218.109.192/30 dev rmnet_mhi0.1 proto kernel scope link src 100.218.109.193
192.168.1.0/24 dev br-lan proto kernel scope link src 192.168.1.1 linkdown
```

```
root@OpenWrt:/# ping www.baidu.com
PING www.baidu.com (183.2.172.185): 56 data bytes
64 bytes from 183.2.172.185: seq=0 ttl=52 time=63.433 ms
```

Step 5: After completing the test, execute **# killall quectel-CM** to terminate the quectel-CM process and close the data connection.

```
root@OpenWrt:/# killall quectel-CM
[05-28_01:48:28:735] requestDeactivateDefaultPDP WdsConnectionIPv4Handle
root@OpenWrt:/# [ 4575.991877] net rmnet_mhi0: link_state 0x1 -> 0x0
[05-28_01:48:28:908] ip link set dev rmnet_mhi0 down
[05-28_01:48:28:948] ip addr flush dev rmnet_mhi0.1
[05-28_01:48:28:977] ip link set dev rmnet_mhi0.1 down
[05-28_01:48:29:021] QmiWwanThread exit
[05-28_01:48:29:024] qmi_main exit
```

NOTE

For more test cases and log examples, please refer to the files starting with "pcie_mhi" in the *log* folder of the quectel-CM source package.

6 Debugging and Troubleshooting

This chapter provides performance optimization suggestions, troubleshooting approaches and solutions for common faults, and guidance on how to effectively collect information for technical support.

6.1. Performance Optimization Suggestions

To fully leverage multi-core CPU performance and avoid network traffic bottlenecks caused by concentrating processing on a single CPU core (e.g., CPU0), it is recommended to perform the following load balancing configuration for RmNet NIC tasks and assign tasks from different stages to different CPU cores.

1. Interrupt Affinity (Handling QMAP Packet Reception)

PCIe interrupts are handled by CPU0 by default. First, execute the following command to confirm the interrupt numbers of the MHI driver.

```
cat /proc/interrupts | grep mhi
```

Output Example:

```
90: 0 0 GIC 448 Edge mhi
91: 6 0 GIC 449 Edge mhi
92: 0 0 GIC 450 Edge mhi
93: 0 0 GIC 451 Edge mhi
```

Execute the following command to bind interrupt handling to CPU0:

```
echo 1 > /proc/irq/90/smp_affinity
echo 1 > /proc/irq/91/smp_affinity
echo 1 > /proc/irq/92/smp_affinity
echo 1 > /proc/irq/93/smp_affinity
```

2. Assign Tasks to Different Cores (Handling QMAP Packet Unpacking and IP Packet Submission)

Assign subsequent tasks to different CPU cores through RPS, making full use of multi-core capabilities.

- Execute the following command to assign tasks for `rmnet_mhi0` (QMAP unpacking) to CPU1 (bitmask 2).

```
echo 2 > /sys/class/net/rmnet_mhi0/queues/rx-0/rps_cpus
```

- Execute the following command to assign tasks for rmnet_mhi0.1 (submitting IP packets to the protocol stack) to CPU2 (bitmask 4).

```
echo 4 > /sys/class/net/rmnet_mhi0.1/queues/rx-0/rps_cpus
```

Table 7: CPU Bitmask Reference Table

CPU Core	Bitmask
CPU0	1
CPU1	2
CPU2	4
CPU3	8

NOTE

For a big & little CPU architecture, it is recommended to assign the most computationally intensive tasks (such as IP packet submission) to the big cores.

3. Adjust Network Stack Backlog Queue

In high-speed data transmission scenarios, the default network backlog queue may be insufficient, leading to packet loss. Execute the following command to appropriately increase the pending packet queue.

```
echo 2000 > /proc/sys/net/core/netdev_max_backlog
```

6.2. Troubleshooting of Common Issues

Common causes of driver loading failure:

- PCIe device is not enumerated
- DTS configuration error
- Hardware communication is unstable

Analysis of typical issues:

Issue 1: The system experiences severe lag or even crashes during communication, severely affecting

normal business operations.

```
行 376: [ 46.879017] cnss[3a]: INFO: PCI device 519621f0 probed successfully
行 501: [ 54.354388] mhi_init Quectel_Linux_PCIE_MHI_Driver_V1.3.7
行 502: [ 54.357433] mhi_pci_probe pci_dev->name = 0002:01:00.0, domain=2, bus=1, slot=0, vendor=17CB, device=0308
行 503: [ 54.557711] mhi_q 0002:01:00.0: BAR 0: assigned [mem 0x10300000-0x10300fff 64bit]
行 504: [ 60.737546] mhi_q 0002:01:00.0: enabling device (0140 -> 0142)
行 524: [ 225.059456] Workqueue: events mhi_pm_st_worker [pcie_mhi]
行 525: [ 225.059457] PC is at mhi_write_reg_field+0x24/0x6c [pcie_mhi]
行 526: [ 225.059458] LR is at mhi_write_reg_field+0x24/0x6c [pcie_mhi]
```

Root Cause: Monitor the physical link by a PCIe analyzer, and it was found that the signal quality of PCIe link was poor, and there were a large number of transmission errors and retransmissions, resulting in unstable MHI communication and abnormal data communication.

Solution:

- Temporary solution: Run at a reduced speed
Limit the PCIe link negotiation speed to improve link stability and temporarily alleviate communication anomalies.

Operation example: Force a lower speed (e.g., reduce from Gen3 to Gen2 or Gen1) in the system configuration.
- Final solution: Hardware optimization
Through systematic hardware optimization, fundamentally optimize the PCIe physical link design to ensure stable and reliable PCIe link operation at standard speeds. Actual optimization items include, but are not limited to:
 - ✓ Adjusting impedance matching and signal integrity layout
 - ✓ Enhancing power supply filtering and reference clock quality
 - ✓ Optimizing PCB routing rules and stack-up design
 - ✓ Replacing hardware components or interface modules if necessary

Issue 2: The MHI driver loads incompletely, and no RmNet NIC or MHI device nodes is generated.

```
行 569: [ 41.498085] mhi_init Quectel_Linux_PCIE_MHI_Driver_V1.3.7
行 570: [ 41.548331] mhi_pci_probe pci_dev->name = 0002:01:00.0, domain=2, bus=1, slot=0, vendor=17CB, device=0308
行 571: [ 41.550521] [I][mhi0][mhi_arch_set_bus_request] Setting bus request to index 1
行 572: [ 41.550534] mhi_q 0002:01:00.0: BAR 0: assigned [mem 0x10300000-0x10300fff 64bit]
行 573: [ 42.091326] mhi_q 0002:01:00.0: enabling device (0140 -> 0142)
行 600: [ 43.227723] [I][mhi0][mhi_init_pci_dev] msi_required = 4, msi_allocated = 4, msi_irq = 162
行 601: [ 44.252737] [I][mhi0][mhi_power_up] dev_state:READY
行 602: [ 44.252740] [I][mhi0][mhi_async_power_up] Requested to power on
行 603: [ 44.313849] [I][mhi0][mhi_alloc_coherent] size = 98304, dma_handle = 75720000
行 604: [ 44.313854] [I][mhi0][mhi_init_dev_ctxt] mhi_ctxt->ctrl_seg = c602d67f
行 605: [ 44.492706] [I][mhi0][mhi_async_power_up] dev_state:READY ee:AMSS
行 606: [ 44.492743] [I][mhi0][mhi_async_power_up] Power on setup success
行 607: [ 44.560061] [I][mhi0][mhi_pm_st_worker] Transition to state:READY
行 608: [ 44.854753] [I][mhi0][mhi_pm_st_worker] INVALID_EE -> AMSS
行 609: [ 44.854756] [I][mhi0][mhi_ready_state_transition] Waiting to enter READY state
行 610: [ 44.858978] [I][mhi0][mhi_ready_state_transition] Device in READY State
行 611: [ 44.858981] [I][mhi0][mhi_tryset_pm_state] Transition to pm state from:POR to:POR
行 612: [ 44.858984] [I][mhi0][mhi_init_mmio] Initializing MMIO
行 613: [ 44.863366] [I][mhi0][mhi_init_mmio] CHDBOFF:0x300
行 614: [ 44.868446] [I][mhi0][mhi_init_mmio] ERDBOFF:0x700
行 615: [ 44.868448] [I][mhi0][mhi_init_mmio] Programming all MMIO values.
行 616: [ 45.313654] [I][mhi0][mhi_pci_show_link] LnkCap: Speed 16GT/s, Width x2
行 617: [ 45.313657] [I][mhi0][mhi_pci_show_link] LnkSta: Speed 8GT/s, Width x1
行 618: [ 45.313658] [I][mhi0][mhi_pci_probe] Return successful
```

Root Cause: The module logs indicate that the bar0 address allocated by the host is abnormal, causing the MHI driver to fail to obtain a valid address and thus being unable to communicate.

```

12 [ 37.090090477/ 0x2e56a094] [0x0 mhi_dev_mmio_get_mhi_state] MHICTRL is 0x0, reset:0
13 [ 37.210096884/ 0x2e79c90e] [0x0 mhi_dev_mmio_get_mhi_state] MHICTRL is 0x0, reset:0
14 [ 37.330130009/ 0x2e9cf38a] [0x0 mhi_dev_mmio_get_mhi_state] MHICTRL is 0x200, reset:0
15 [ 37.330143134/ 0x2e9cf473] [0x0 mhi_dev_enable] state:2
16 [ 37.330154124/ 0x2e9cf546] [0x0 mhi_dev_cache_host_cfg] Number of Event rings : 3, HW Event rings : 4
17 [ 37.330174697/ 0x2e9cf6d4] [0x0 mhi_dev_cache_host_cfg] MEM_ALLOC: size:16896 RING_ALLOC
18 [ 37.330621728/ 0x2e9d1870] [0x0 mhi_dev_ring_init] initializing all rings
19 [ 37.330644645/ 0x2e9d1a16] [0x0 mhi_dev_read_from_host_ipa] device 0xb01138bc8b4db000 <-- host 0x10077af5f00, size 44
20 [ 37.374199436/ 0x2ea9dcb6] [0x0 mhi_dev_read_from_host_ipa] device 0xb08a37688b4de000 <-- host 0x10077af5e00, size 132
21 [ 37.422286520/ 0x2eb7f349] [0x0 mhi_dev_cache_host_cfg] cmd ring_base:0xffffffffffffff, rp:0xffffffffffffff, wp:0xffffffffffffff
22 [ 37.422299697/ 0x2eb7f431] [0x0 mhi_dev_cache_host_cfg] ev ring_base:0xffffffffffffff, rp:0xffffffffffffff, wp:0xffffffffffffff
23 [ 37.422329853/ 0x2eb7f675] [0x0 mhi_dev_cache_host_cfg] MHI ring start failed:-12
24 [ 37.422337822/ 0x2eb7f70e] [0x0 mhi_dev_cache_host_cfg] MEM_DEALLOC: size:16896 RING_ALLOC
25 [ 37.422345947/ 0x2eb7f7a9] [0x0 mhi_dev_cache_host_cfg] MEM_DEALLOC: size:44 CMD_CTX_CACHE
26 [ 37.422356572/ 0x2eb7f876] [0x0 mhi_dev_cache_host_cfg] MEM_DEALLOC: size:132 EV_CTX_CACHE
27 [ 37.422360739/ 0x2eb7f9c5] [0x0 mhi_dev_cache_host_cfg] MEM_DEALLOC: size:132 EV_CTX_CACHE
28 [ 37.422367666/ 0x2eb7f94a] [0x0 mhi_dev_enable] Failed to cache the host config
29 root@sdxlemur:~# cat /sys/kernel/debug/ipc_logging/mhi-uci/log
30 [ 0.258725834/ 0x430303d] [mhi_uci_init] Setting up channel attributes.
31 [ 0.258746927/ 0x43031bf] [mhi_uci_init] Initializing clients
32 [ 0.258748906/ 0x43031e5] [mhi_uci_init] Registering for MHI events

```

Solution:

For host platforms such as IPQ9574, configure the PCIe IOMMU access method as *iommu-dma = "disabled"*. Please refer to **Chapter 4.2** for details.

NOTE

IOMMU access methods in different host platforms are variable. Please refer to the platform-specific codes for specific configurations.

6.3. Issue Reporting and Log Collection

Before contacting Quectel Technical Support, please systematically collect the following information to facilitate analysis by the technical support team.

1. Basic Information

Hardware: Module model, firmware version, host platform.

Software: Host kernel version, driver version and configuration (single-NIC, multi-NIC, and bridge mode)

2. Issue Description

Clearly describe the phenomenon and expected behavior, and provide detailed steps to reproduce the issue.

3. Logs (Most Critical)

Module logs:

```
dmesg
```

```
cat /sys/kernel/debug/ipc_logging/ep-pcie-short/log
cat /sys/kernel/debug/ipc_logging/ep-pcie-long/log
cat /sys/kernel/debug/ipc_logging/ep-pcie-dump/log
cat /sys/kernel/debug/ipc_logging/mhi/log
cat /sys/kernel/debug/ipc_logging/mhi-uci/log
cat /sys/kernel/debug/ipc_logging/ipa/log
cat /sys/kernel/debug/ipc_logging/gsi/log
```

Host logs:

```
dmesg
cat /sys/kernel/debug/mhi_q/*/states
cat /sys/kernel/debug/mhi_q/*/chan
cat /sys/kernel/debug/mhi_q/*/events
cat /sys/kernel/debug/mhi_q/mhi_netdev/*/states
cat /proc/interrupts | grep mhi
```

7 Appendix References

Table 8: Related Documents

Document Name
[1] Quectel_QConnectManager_Linux_User_Guide
[2] Quectel_RG50xQ&RM5xxQ_Series_AT_Commands_Manual
[3] Quectel_RG520N&RG525F&RG5x0F&RM5x0N_Series_AT_Commands_Manual
[4] Quectel_EG512R&EM1x0R_Series_AT_Commands_Manual
[5] Quectel_RG255C_Series&RM255C-GL_AT_Commands_Manual

Table 9: Terms and Abbreviations

Abbreviation	Description
AP	Application Processor
APN	Access Point Name
BIOS	Basic Input/Output System
CPU	Central Processing Unit
DNS	Domain Name System
DTS	Device Tree Source
eFuse	Electronic Fuse
EP	Endpoint
GPIO	General-Purpose Input/Output
IOMMU	Input/Output Memory Management Unit

IP	Internet Protocol
MAC	Medium Access Control
MBIM	Mobile Interface Broadband Model
MDM	Modem
MHI	Modem Host Interface
MSI	Message Signaled Interrupts
MSI-X	Message Signaled Interrupts eXtended
NIC	Network Interface Card
NMEA	National Marine Electronics Association
KO	Kernel Object
OS	Operating System
PBL	Primary Bootloader
PC	Personal Computer
PCB	Printed Circuit Board
PCIe	Peripheral Component Interconnect Express
PID	Product ID
QMAP	Qualcomm MUX and Aggregation Protocol
QMI	Qualcomm Messaging Interface
RAM	Random Access Memory
RC	Root Complex
RmNet	Remote Network
RPS	Receive Packet Steering
RX	Receive
SFTP	Secret File Transfer Protocol
SoC	System on a Chip

TCP	Transmission Control Protocol
TFTP	Trivial File Transfer Protocol
TX	Transmit
UCI	Unified Configuration Interface
UDP	User Datagram Protocol
USB	Universal Serial Bus
VID	Vendor ID