

EC25&EC21&EC20 R2.0

QuecOpen Developer Guide

LTE Module Series

Rev. EC25&EC21&EC20 R2.0_QuecOpen_Developer_Guide_V1.0

Date: 2017-05-18



Our aim is to provide customers with timely and comprehensive service. For any assistance, please contact our company headquarters:

Quectel Wireless Solutions Co., Ltd.

Office 501, Building 13, No.99, Tianzhou Road, Shanghai, China, 200233

Tel: +86 21 5108 6236

Email: info@quectel.com

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

<http://www.quectel.com/support/salesupport.aspx>

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

<http://www.quectel.com/support/techsupport.aspx>

Or email to: Support@quectel.com

GENERAL NOTES

QUECTEL OFFERS THE INFORMATION AS A SERVICE TO ITS CUSTOMERS. THE INFORMATION PROVIDED IS BASED UPON CUSTOMERS' REQUIREMENTS. QUECTEL MAKES EVERY EFFORT TO ENSURE THE QUALITY OF THE INFORMATION IT MAKES AVAILABLE. QUECTEL DOES NOT MAKE ANY WARRANTY AS TO THE INFORMATION CONTAINED HEREIN, AND DOES NOT ACCEPT ANY LIABILITY FOR ANY INJURY, LOSS OR DAMAGE OF ANY KIND INCURRED BY USE OF OR RELIANCE UPON THE INFORMATION. ALL INFORMATION SUPPLIED HEREIN IS SUBJECT TO CHANGE WITHOUT PRIOR NOTICE.

COPYRIGHT

THE INFORMATION CONTAINED HERE IS PROPRIETARY TECHNICAL INFORMATION OF QUECTEL CO., LTD. TRANSMITTING, REPRODUCTION, DISSEMINATION AND EDITING OF THIS DOCUMENT AS WELL AS UTILIZATION OF THE CONTENT ARE FORBIDDEN WITHOUT PERMISSION. OFFENDERS WILL BE HELD LIABLE FOR PAYMENT OF DAMAGES. ALL RIGHTS ARE RESERVED IN THE EVENT OF A PATENT GRANT OR REGISTRATION OF A UTILITY MODEL OR DESIGN.

Copyright © Quectel Wireless Solutions Co., Ltd. 2017. All rights reserved.

About the Document

History

Revision	Date	Author	Description
1.0	2017-05-18	Stanley YONG	Initial

Contents

About the Document.....	2
Contents	3
Figure Index	5
Table Index.....	6
1 Introduction	7
1.1. Applicative Modules	7
2 QuecOpen™ Platform.....	8
2.1. System Architecture	8
2.2. Open System Resources	9
2.2.1. Processor	9
2.2.2. Host System	9
2.2.3. Flash Space.....	9
2.2.4. RAM Space	10
2.3. Open Hardware Resources.....	10
2.3.1. UARTs	13
2.3.2. GPIOs.....	14
2.3.3. Interrupts	14
2.3.4. I2C	14
2.3.5. SPI.....	14
2.3.6. ADC.....	14
2.3.7. PCM.....	14
2.3.8. SDIO.....	14
2.3.9. SGMII	15
2.3.10. USB.....	15
3 Work with QuecOpen™	16
3.1. Set up Host Environment	17
3.1.1. System Requirements.....	17
3.1.2. Install USB Driver	17
3.1.3. Install and Set up ADB Driver on PC	17
3.1.4. Install Cross Compiler	19
3.2. Synchronize SDK and Module	19
3.3. Compilation.....	20
3.3.1. Compiling	20
3.3.2. Compiling Output	20
3.4. Download.....	21
3.4.1. During Development Phase	21
3.4.1.1. With ADB	21
3.4.1.2. With fastboot.....	22
3.4.2. For Production.....	22
3.5. Launch Application	24

3.6.	Debug Application.....	24
4	Programming Reference	25
4.1.	System.....	25
4.1.1.	Time.....	25
4.1.2.	Timer	25
4.1.3.	Multitasking	27
4.2.	AT & URC	27
4.3.	I/O Interfaces	28
4.3.1.	GPIOs.....	28
4.3.2.	EINT	28
4.3.3.	UARTs	29
4.3.3.1.	Pin 63/66 as UART	30
4.3.3.2.	Pin 38/39 as UART	31
4.3.3.3.	Pin 67/68 as UART	31
4.3.3.4.	Programming reference.....	32
4.3.4.	ADC.....	32
4.3.5.	I2C.....	33
4.3.6.	SPI.....	34
4.3.6.1.	SPI Mode	34
4.3.6.2.	Install SPI Driver.....	35
4.3.6.3.	SPI Parameters	36
4.4.	File System.....	37
4.5.	SD Card/eMMC Flash	38
4.6.	Audio.....	39
4.7.	TTS.....	39
4.8.	Voice Call.....	40
4.9.	SMS.....	40
4.10.	Network Service	40
4.11.	Data Service	40
4.11.1.	DSI_NetCtrl	41
4.11.2.	Sockets.....	43
4.12.	GNSS.....	45
4.13.	Wi-Fi	45
4.14.	(U)SIM Card	46
5	Customization	47
5.1.	Pin Multiplexing Function	47
5.1.1.	With Linux Kernel Code	47
5.1.2.	Without Linux Kernel Code	47
5.2.	Rootfs	48
6	Appendix A References.....	49

Figure Index

FIGURE 1: QUECOPEN™ ARCHITECTURE	8
FIGURE 2: HOW TO START WORKING WITH QUECOPEN™	16
FIGURE 3: BUILD DIRECTORY	21
FIGURE 4: DOWNLOAD DIRECTORY	21
FIGURE 5: DOWNLOAD APPLICATION VIA QFLASH	23
FIGURE 6: SPI 4-LINE MODE	34
FIGURE 7: SPI 6-LINE MODE	35
FIGURE 8: SPI 8 CHANNELS.....	36
FIGURE 9: DISASSEMBLE BOOT.IMG	48

Quectel
Confidential

Table Index

TABLE 1: DATA STORAGE PLACES FOR DEVELOPERS.....	9
TABLE 2: MULTIPLEXING PINS	10
TABLE 3: UART PINS.....	29
TABLE 4: UART TTY	30
TABLE 5: I2C PINS.....	33
TABLE 6: SPI PINS	34
TABLE 7: RELATED DOCUMENTS.....	49
TABLE 8: TERMS AND ABBREVIATIONS.....	49

Quectel
Confidential

1 Introduction

Quectel QuecOpen[™] is an open embedded platform that is built on Linux system. It is designed to simplify the development for IoT (Internet of Things) applications.

This guide document mainly provides the following information for developers:

- General information about QuecOpen[™] platform
- How to start working with QuecOpen[™]
- How to build an application
- How to compile an application
- How to download an application
- How to get the application working

1.1. Applicative Modules

QuecOpen[™] is applicable to the following Quectel LTE modules:

- EC25
- EC21
- EC20 R2.0

2 QuecOpen™ Platform

2.1. System Architecture

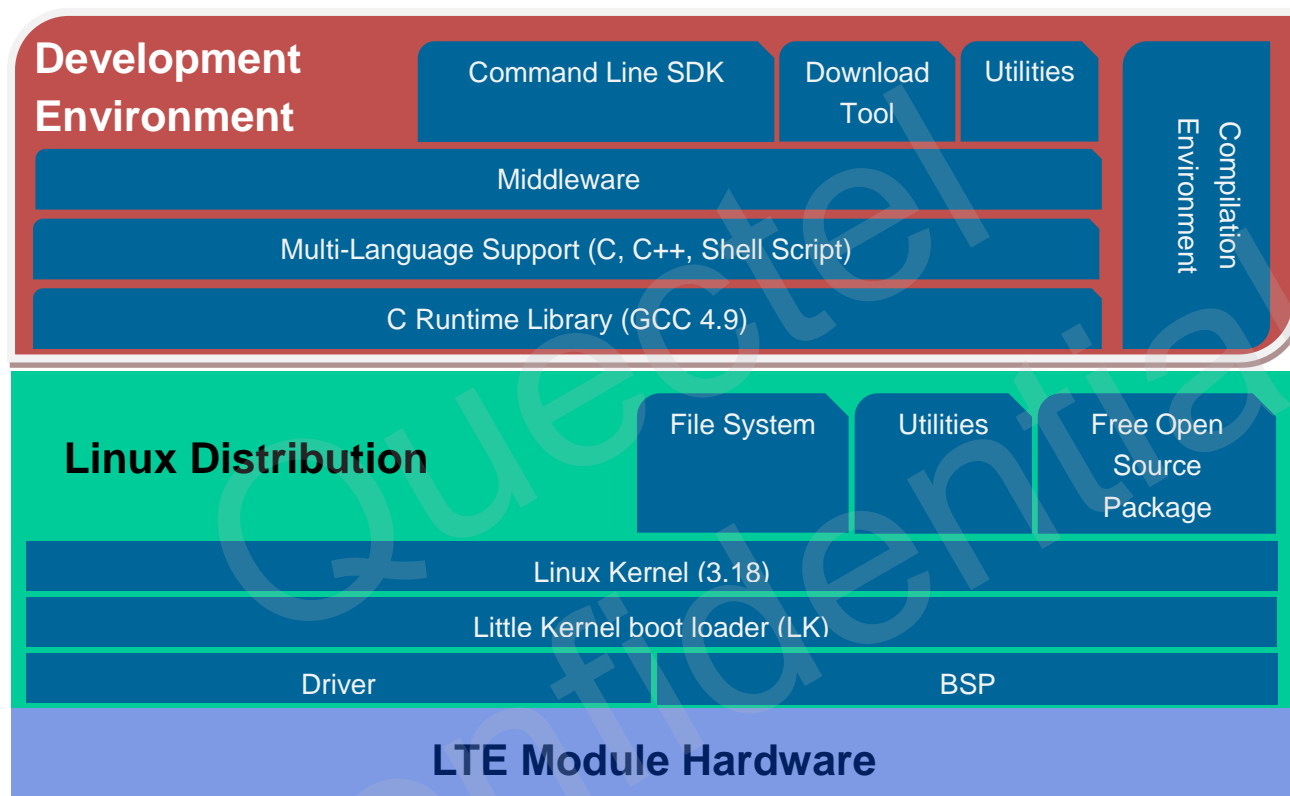


Figure 1: QuecOpen™ Architecture

2.2. Open System Resources

2.2.1. Processor

Cortex-A7 1.2GHz, ARMv7.

2.2.2. Host System

Linux system with kernel 3.18.

2.2.3. Flash Space

In QuecOpen™, customer applications and other data are stored in Linux file system. The places that can be used by developers are listed in the following table.

Table 1: Data Storage Places for Developers

Filesystem	Type	Size	Used	Available	Use (%)	Mounted on
ubi0:rootfs	ubifs	55.8M	36.8M	19.0M	66%	/
ubi0:usrfs	ubifs	31.9M	868.0K	31.0M	3%	/data
/dev/ubi0_2	ubifs	48.3M	24.0K	45.8M	0%	/cache

As defined in the table above, there are three UBI partitions that can be used to store application images or data in Linux file system. They are “rootfs”, “usrfs” and “cache” partitions.

- *ubi0:rootfs (/)*

The space available is around 19MB.

- *ubi0:usrfs (/data)*

The space available is around 31MB. And the partition is mounted to */data*. By default, customer applications are put in this partition.

- */dev/ubi0_2 (/cache)*

The space available is around 45MB. And it is mounted to */cache* by default. However, this UBI partition is shared with FOTA option, which means FOTA option will erase and write this partition when there is firmware (such as modem image, or Linux image) upgraded by FOTA. So developers may temporarily store some data in */dev/ubi0_2*. However, do remember the data will be cleaned up when doing upgrade by FOTA.

Totally, the fixed flash space (for application and data) is around 50MB (19MB+31MB), and can be expanded to around 95MB (50MB+45MB). The runtime flash space is around 45MB.

2.2.4. RAM Space

RAM available: 50M bytes

2.3. Open Hardware Resources

Table 2: Multiplexing Pins

Pin No.	Pin Name	Pin Location	Combined Interface (default)	Pin Multiplexing			Power Domain	Reset ²⁾	Wake-up Interrupt ³⁾	Remark
				Primary Function ¹⁾	Alternate Function 1	Alternate Function 2				
1	GPIO1	Edge		GPIO_25	--	--	1.8V	B-PD,L	✓	BOOT_CONFIG_2
2	GPIO2	Edge		GPIO_10	--	--	1.8V	B-PD,L	x	
3	GPIO3	Edge		GPIO_42	--	--	1.8V	B-PD,L	✓	
4	GPIO4	Edge		GPIO_11	--	--	1.8V	B-PD,L	✓	
5	GPIO5	Edge		GPIO_24	--	--	1.8V	B-PD,L	x	BOOT_CONFIG_1
11	DBG_RXD	Edge	Debug UART	DBG_RXD	--	--	1.8V	B-PD,L	--	
12	DBG_TXD	Edge		DBG_TXD	--	--	1.8V	B-PD,L	--	
13	USIM_PRESENCE	Edge		USIM_PRESENCE	GPIO_34	--	1.8V	B-PD,L	✓	
23	SD_CARD_DET	Edge		SD_CARD_DET	GPIO_26	--	1.8V	B-PD,L	✓	
24	PCM_IN	Edge		PCM_IN	GPIO_76	--	1.8V	B-PD,L	✓	
25	PCM_OUT	Edge	PCM interface	PCM_OUT	GPIO_77	--	1.8V	B-PD,L	x	
26	PCM_SYNC	Edge		PCM_SYNC	GPIO_79	--	1.8V	B-PD,L	✓	BOOT_CONFIG_7
27	PCM_CLK	Edge		PCM_CLK	GPIO_78	--	1.8V	B-PD,L	x	BOOT_CONFIG_8
28	SDC2_DATA3	Edge	SDIO2 (SD card)	SDC2_DATA3	--	--	LOW SD Voltage:	B-PD,L	--	Dedicated pin

			interface)				1.8V. HIGH SD Voltage: 2.85V.		
							LOW SD Voltage: 1.8V. HIGH SD Voltage: 2.85V.	B-PD,L	Dedicated pin
29	SDC2_ DATA2	Edge	SDC2_ DATA2	--	--		1.8V. HIGH SD Voltage: 2.85V.		
							LOW SD Voltage: 1.8V. HIGH SD Voltage: 2.85V.	B-PD,L	Dedicated pin
30	SDC2_ DATA1	Edge	SDC2_ DATA1	--	--		1.8V. HIGH SD Voltage: 2.85V.		
							LOW SD Voltage: 1.8V. HIGH SD Voltage: 2.85V.	B-PD,L	Dedicated pin
31	SDC2_ DATA0	Edge	SDC2_ DATA0	--	--		1.8V. HIGH SD Voltage: 2.85V.		
							LOW SD Voltage: 1.8V. HIGH SD Voltage: 2.85V.	B-PD,L	Dedicated pin
32	SDC2_CLK	Edge	SDC2_CLK	--	--		1.8V. HIGH SD Voltage: 2.85V.	B-NP,L	Dedicated pin
							LOW SD Voltage: 1.8V. HIGH SD Voltage: 2.85V.		
33	SDC2_CMD	Edge	SDC2_CMD	--	--		1.8V. HIGH SD Voltage: 2.85V.	B-PD,L	Dedicated pin
							LOW SD Voltage: 1.8V. HIGH SD Voltage: 2.85V.		
34	VDD_SDIO	Edge	VDD_SDIO	--	--		1.8V. HIGH SD Voltage: 2.85V.	L	Power, Dedicated pin
							LOW SD Voltage: 1.8V. HIGH SD Voltage: 2.85V.		
37	SPI_CS_N	Edge	SPI_CS_N_B LSP6	GPIO_22	UART_RTS_ BLSP6	1.8V	B-PD,L	✓	
38	SPI_MOSI	Edge	SPI_MOSI_B LSP6	GPIO_20	UART_TXD_ BLSP6	1.8V	B-PD,L	✓	

39	SPI_MISO	Edge		SPI_MISO_B LSP6	GPIO_21	UART_RXD_ BLSP6	1.8V	B-PD,L	✓	
40	SPI_CLK	Edge		SPI_CLK_ BLSP6	GPIO_23	UART_CTS_ BLSP6	1.8V	B-PU,H	x	BOOT_ CONFIG_4
41	I2C_SCL	Edge	I2C	I2C_SCL_ BLSP2	GPIO_7	UART_CTS_ BLSP2	1.8V	B-PD,L	x	
42	I2C_SDA	Edge	interface, host only	I2C_SDA_ BLSP2	GPIO_6	UART_RTS_ BLSP2	1.8V	B-PD,L	x	
62	GPIO6	Edge		GPIO_75	--	--	1.8V	B-PD,L	✓	
63	UART1_ TXD	Edge	UART	UART_TXD_ BLSP2	GPIO_4	UART_TXD_ BLSP2	1.8V	B-PD,L	x	
66	UART1_ RXD	Edge	interface	UART_RXD_ BLSP2	GPIO_5	UART_RXD_ BLSP2	1.8V	B-PD,L	✓	
64	MAIN_CTS	Edge		UART_CTS_ BLSP3	GPIO_3	--	1.8V	B-PD,L	✓	
65	MAIN_RTS	Edge	Main UART	UART_RTS_ BLSP3	GPIO_2	--	1.8V	B-PD,L	x	
67	MAIN_TXD	Edge		UART_TXD_ BLSP3	GPIO_0	--	1.8V	B-PD,L	x	
68	MAIN_RXD	Edge		UART_RXD_ BLSP3	GPIO_1	--	1.8V	B-PD,L	✓	
119	EPHY_ RST_N	Bottom		EPHY_ RST_N	GPIO_29	--	1.8V/ 2.85V	BH-PD,L	✓	
120	EPHY_ INT_N	Bottom		EPHY_ INT_N	GPIO_30	--	1.8V	B-PD,L	✓	
121	SGMII_ DATA	Bottom		SGMII_ DATA	GPIO_28	--	1.8V/ 2.85V	BH-PD,L	✓	
122	SGMII_CLK	Bottom	SGMII	SGMII_CLK	GPIO_27	--	1.8V/ 2.85V	BH-PD,L	x	
123	SGMII_ TX_M	Bottom	interface	SGMII_ TX_M	--	--	Analog signal	L	--	Dedicated pin
124	SGMII_ TX_P	Bottom		SGMII_ TX_P	--	--	Analog signal	L	--	Dedicated pin
125	SGMII_ RX_P	Bottom		SGMII_ RX_P	--	--	Analog signal	L	--	Dedicated pin
126	SGMII_ RX_M	Bottom		SGMII_ RX_M	--	--	Analog signal	L	--	Dedicated pin
129	SDC1_ DATA3	Bottom	SDIO1	SDC1_ DATA3	GPIO_12	--	1.8V	B-PD,L	✓	
130	SDC1_ DATA2	Bottom	(WLAN interface)	SDC1_ DATA2	GPIO_13	--	1.8V	B-PD,L	✓	
131	SDC1_ DATA1	Bottom		SDC1_ DATA1	GPIO_14	--	1.8V	B-PD,L	x	

132	SDC1_	Bottom		SDC1_	GPIO_15	--	1.8V	B-PD,L	x	
	DATA0			DATA0						
133	SDC1_CLK	Bottom		SDC1_CLK	GPIO_16	--	1.8V	B-NP,L	✓	
134	SDC1_CMD	Bottom		SDC1_CMD	GPIO_17	--	1.8V	B-PD,L	✓	
135	WLAN_	Bottom		WLAN_	GPIO_59	--	1.8V	B-PD,L	✓	
	WAKE			WAKE						
136	WLAN_EN	Bottom	WLAN coexistence	WLAN_EN	GPIO_38	--	1.8V	B-PD,L	✓	BOOT_
										CONFIG_12
137	COEX_	Bottom	control/ management	COEX_	GPIO_37	--	1.8V	B-PD,L	✓	FORCE_
	UART_RXD			UART_RXD						USB_BOOT
138	COEX_	Bottom		COEX_	GPIO_36	--	1.8V	B-PD,L	x	BOOT_
	UART_TXD			UART_TXD						CONFIG_3

NOTES

- ¹⁾ By default, the module pins support the primary function.
- ²⁾ “B”: Bidirectional digital with CMOS input
“BH”: High-voltage tolerant bidirectional digital with CMOS input
“PD”: Contain an internal pull-down device
“PU”: Contain an internal pull-up device
“L”: Low level
“H”: High level
- ³⁾ All GPIOs support interrupt option. But not all interrupts can wake up the sleeping module.
“✓” means wake-up interrupt is supported, that is, the pin can wake up the sleeping module.
“x” means wake-up interrupt is not supported, that is, the pin cannot be used to wake up the sleeping module.
- GPIO_27 ~ GPIO_30 are not available currently.
- BOOT_CONFIG_xx & FORCE_USB_BOOT means these pins MUST be kept in the “Reset” status when powering up the module. Otherwise, the module might not be able to boot properly.

2.3.1. UARTs

QuecOpen module provides four serial ports: one Debug UART and three application UARTs.

- Debug UART:** Pin 11/12 (RXD/TXD), used to debug the AP system and QuecOpen™ applications.
- Application UARTs:**
 - Main UART: pin 67/68 (TXD/RXD) + pin 64/65 (CTS/RTS).
 - The other two application UART ports: pin 63/66 (TXD/RXD) + pin 41/42 (CTS/RTS)
pin 38/39 (TXD/RXD) + pin 37/40 (RTS/CTS)

Please see **Chapter 4.3.3** for details about how to program these serial interfaces.

2.3.2. GPIOs

There are more than 30 I/O pins that can be configured as general purpose I/O pins. They are multiplexed with other functional pins. All GPIO pins can be accessed by API functions.

Please refer to **Chapter 4.3.1** for the API functions for programming GPIO.

2.3.3. Interrupts

All pins that can be multiplexed as GPIO can be interrupt pins. However, not all interrupt pins can wake up the module that is in low-power-consumption mode. Please see the field "Wake-up Interrupt" in **Table 2**.

Please refer to **Chapter 4.3.2** for the API functions for programming interrupt.

2.3.4. I2C

QuecOpen module provides one hardware I2C interface. Please refer to **Chapter 4.3.5** for details.

2.3.5. SPI

QuecOpen module provides one hardware SPI interface. Please refer to **Chapter 4.3.6** for details.

2.3.6. ADC

There are two analogue input pins that can be ADC. The pin numbers are 45 and 44. Please refer to **document [2]** for the electrical characteristics of ADC interface.

2.3.7. PCM

QuecOpen module provides a PCM interface. It is designed for audio codec by default. Combined with I2C interface, the application can control and manage the codec chip using **AT+QIIC** command or API functions defined in **Chapter 4.3.5**.

2.3.8. SDIO

QuecOpen module provides two SDIO interfaces (SDC1 and SDC2). Both of them are 4-bit bidirectional data bus.

SDC2 is designed for SD card or eMMC flash, and SDC1 is designed for Wi-Fi.

2.3.9. SGMII

QuecOpen module provides an SGMII interface.

2.3.10. USB

The USB interface can be mapped into several different functional interfaces, as shown below:

- USB-AT port
- USB-DM port
- USB-NMEA port
- USB-Modem port
- USB-Network adapter

In QuecOpen module, the GNSS NMEA is outputted to application through a virtual serial port instead of USB-NMEA port by default.

The USB-DM port can be used to download firmware to module, and debug the module system. So, developers MUST design the USB interface for the convenience of downloading and debugging.

Please refer to **document [1]** for USB details.

3 Work with QuecOpen™

This chapter introduces how to start working with QuecOpen™. The flow for working with QuecOpen™ is described as below.

Before compiling an application, developers need to update some files according to the current firmware version. Please refer to the following flow diagram and **Chapter 3.2** for more details.

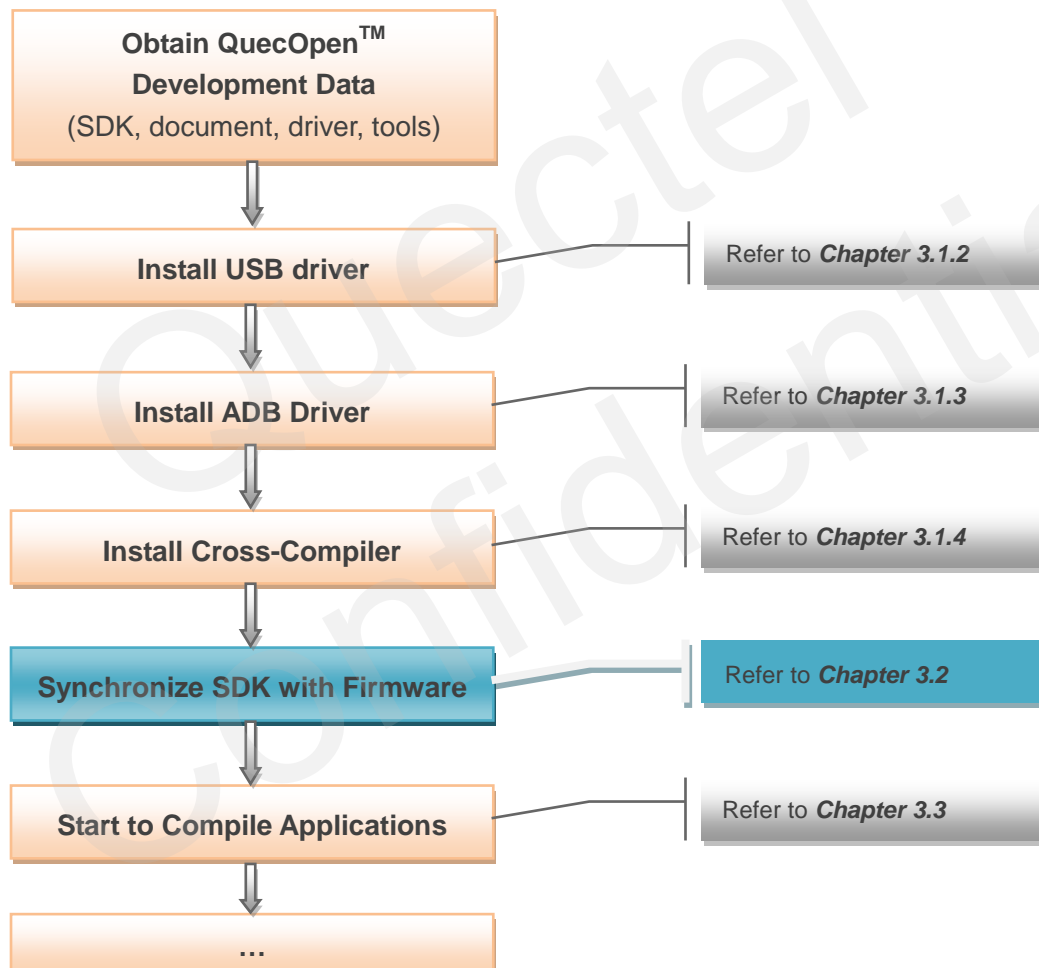


Figure 2: How to Start Working with QuecOpen™

3.1. Set up Host Environment

3.1.1. System Requirements

QuecOpen™ provides the ready-made compilation software package which includes the host operating system, compiler and the other required tools.

- **Operating system**
Ubuntu 64-bit OS, version 12.04 or 14.
- **Compiler**
arm-oe-linux-gnueabi
- **ADB**
Android Debug Bridge version 1.0.31.
- **Fastboot**

3.1.2. Install USB Driver

Please refer to **document [1]** for details about USB driver installation, and make sure the USB interface can work normally.

3.1.3. Install and Set up ADB Driver on PC

ADB can be used to upload or download files between the host computer and QuecOpen module, and execute shell commands of the module.

Please follow the commands below to install ADB on your host system.

```
sudo apt-get update
sudo apt-get install android-tools-adb
```

If the above installation command fails, please try again with the series of commands shown below.

```
sudo add-apt-repository ppa:nilarimogard/webupd8
sudo apt-get update
sudo apt-get install android-tools-adb
```

After ADB installation, developers can check whether it is installed successfully by executing adb command.

```
will@will-OptiPlex-790:~$ adb
Android Debug Bridge version 1.0.31
```

- List ADB devices

```
sudo adb devices
```

If the USB device cannot be listed, follow the steps below.

```
will@will-OptiPlex-790:~$ adb devices
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
List of devices attached
```

No device listed

- Add module USB VID into ADB configuration

There is a hidden directory name `.android` in your work directory. Please create the file `adb_usb.ini` in `.android` directory, and write into the USB VID in a new line.

```
will@will-OptiPlex-790:~$ ls .android/ -al
total 16
drwxrwxrwx  2 root root 4096 Jul  6 2016
drwxrwxrwx 54 will will 4096 Mar  8 18:28
-rwxrwxrwx  1 root root 1704 Oct 27 2015 adbkey
-rwxrwxrwx  1 root root  723 Oct 27 2015 adbkey.pub
```

```
cd
```

```
sudo gedit .android/adb_usb.ini &
```

Or

```
sudo vi .android/adb_usb.ini
```

Input the USB VID in a new line in the file, for example 0x2C7C. Developers can check the USB VID with `lsusb` command.

```
will@will-OptiPlex-790:~$ ls .android/ -al
total 20
drwxrwxrwx  2 root root 4096 Mar 14 10:36
drwxrwxrwx 54 will will 4096 Mar  8 18:28
-rwxrwxrwx  1 root root 1704 Oct 27 2015 adbkey
-rwxrwxrwx  1 root root  723 Oct 27 2015 adbkey.pub
-rw-r--r--  1 root root    7 Mar 14 10:36 adb_usb.ini
```

- Restart ADB Service

```
sudo adb kill-server
```

```
sudo adb devices
```

```
stanley@stanley-OptiPlex-7020:~$ sudo adb devices
```

```
List of devices attached
```

```
???????????? device adb device
```

Now the ADB is ready.

3.1.4. Install Cross Compiler

Quectel provides the development environment software package (please consult Quectel Technical Supports for the compiler). Developers just need to decompress the package *QuecOpen_CrossCompiler.tar.gz*:

```
sudo tar -xf QuecOpen_CrossCompiler.tar.gz -C /opt
```

After decompression, developers can verify whether the environment is setup properly:

```
/opt/ql-oe/sysroots/x86_64-linux/usr/bin/arm-oe-linux-gnueabi/arm-oe-linux-gnueabi-gcc -v
```

If the environment is properly setup, the information of GCC version will be shown.

```
-style=gnu --enable-linker-build-id --with-ppl=no --with-
.9.2 --with-sysroot=/not/exist --with-build-sysroot=/ho
directories --with-mpfr=/home/sdc/stanley/MDM9x07/Open
ch=armv7-a
Thread model: posix
gcc version 4.9.2 (GCC)
```

3.2. Synchronize SDK and Module

Since the application image (including the customizations on roofs) compiled under SDK environment is merged into the Linux file system on QuecOpen module, developers need to do some preprocessing before compiling an application to .ubi image.

- Check the firmware version that is being used on the module with **ATI** command.
Suppose that the checked firmware version is *EC20CEFAR02A01M4G_OCPU*.
- Synchronize the following files in */SDK/target/9x07/* with the ones with the same name in the original firmware package of version *EC20CEFAR02A01M4G_OCPU*.

File 1: *partition.mbn*

File 2: *mdm9607-perf-boot.img*

File 3: *mdm-perf-image-mdm9607-perf.tar.gz*

File 4: *ENPRG9x07.mbn*

File 5: *NPRG9x07.mbn*

Also see READ ME in SDK about this.

3.3. Compilation

3.3.1. Compiling

In QuecOpen™, compiling commands are executed in command line. The clean and compiling commands are defined as below.

```
sudo ./build.sh clean
sudo ./build.sh new <path of makefile>
```

QuecOpen™ SDK provides some examples for developers' reference. The following illustrates an example for compiling "hello world".

```
Quectel_QuecOpen_SDK_V1.0/
├──example
│   ├──adc
│   ├──at
│   ├──gpio
│   └──hello_world
...
```

Developers need to enter the root directory of SDK, and execute the following command to compile helloworld.c. After compiling, an executable file *ApplImageBinV01* is generated in *SDK/build/*.

```
sudo ./build.sh new example/hello_world
```

3.3.2. Compiling Output

In command-line, the processing information of compiler will be outputted during compiling. All WARNINGS and ERRORS are recorded in */SDK/build/build.log*.

If the compiling succeeds, the executable program file will be outputted in *build/*. Additionally, the compilation environment will create a *.ubi* file in *build/*, and generate the downloadable software package for QFlash tool in *download/*. Totally, the outputted information includes:

- Executable file
- .ubi file
- *download/* directory for QFlash tool

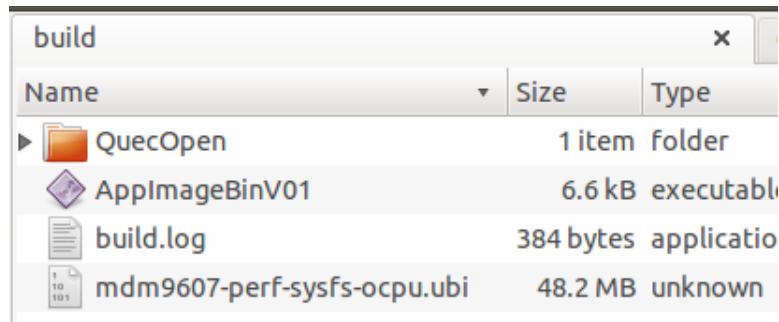


Figure 3: Build Directory

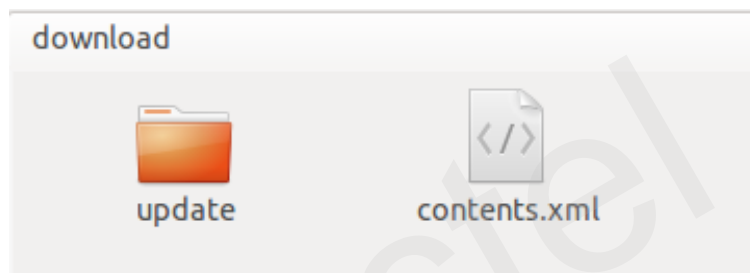


Figure 4: Download Directory

3.4. Download

3.4.1. During Development Phase

3.4.1.1. With ADB

Developers can upload the application executable file to the file system of module Linux system by ADB shell command.

```
sudo adb push <local path> <module path>
```

An example is shown below:

```
adb push /home/stanley/MyWork/Quectel_QuecOpen_V1.0/example/helloWorld/helloWorld /home/root/
```

The executing result is:

```
stanley@stanley-OptiPlex-7020:~/MyWork/Quectel_QuecOpen_V1.0/example/helloWorld$ sudo adb
push /home/stanley/MyWork/Quectel_QuecOpen_V1.0/example/helloWorld/helloWorld /home/root
83 KB/s (3692 bytes in 0.043s)
stanley@stanley-OptiPlex-7020:~/MyWork/Quectel_QuecOpen_V1.0/example/helloWorld$
```

3.4.1.2. With fastboot

Developers can download the .ubi file (in *build/* directory) to QuecOpen module's flash, and then reset the module. The application will be loaded and run automatically.

3.4.2. For Production

In order to improve the production efficiency, Quectel provides the special fixture and download tool which can download firmware to several modules synchronously. Mass production customers can consult Quectel Technical Supports for that if needed.

Of course, developers can also download the application (.ubi) during the stage of development.

The download port is **USB-DM** interface.

Quectel
Confidential

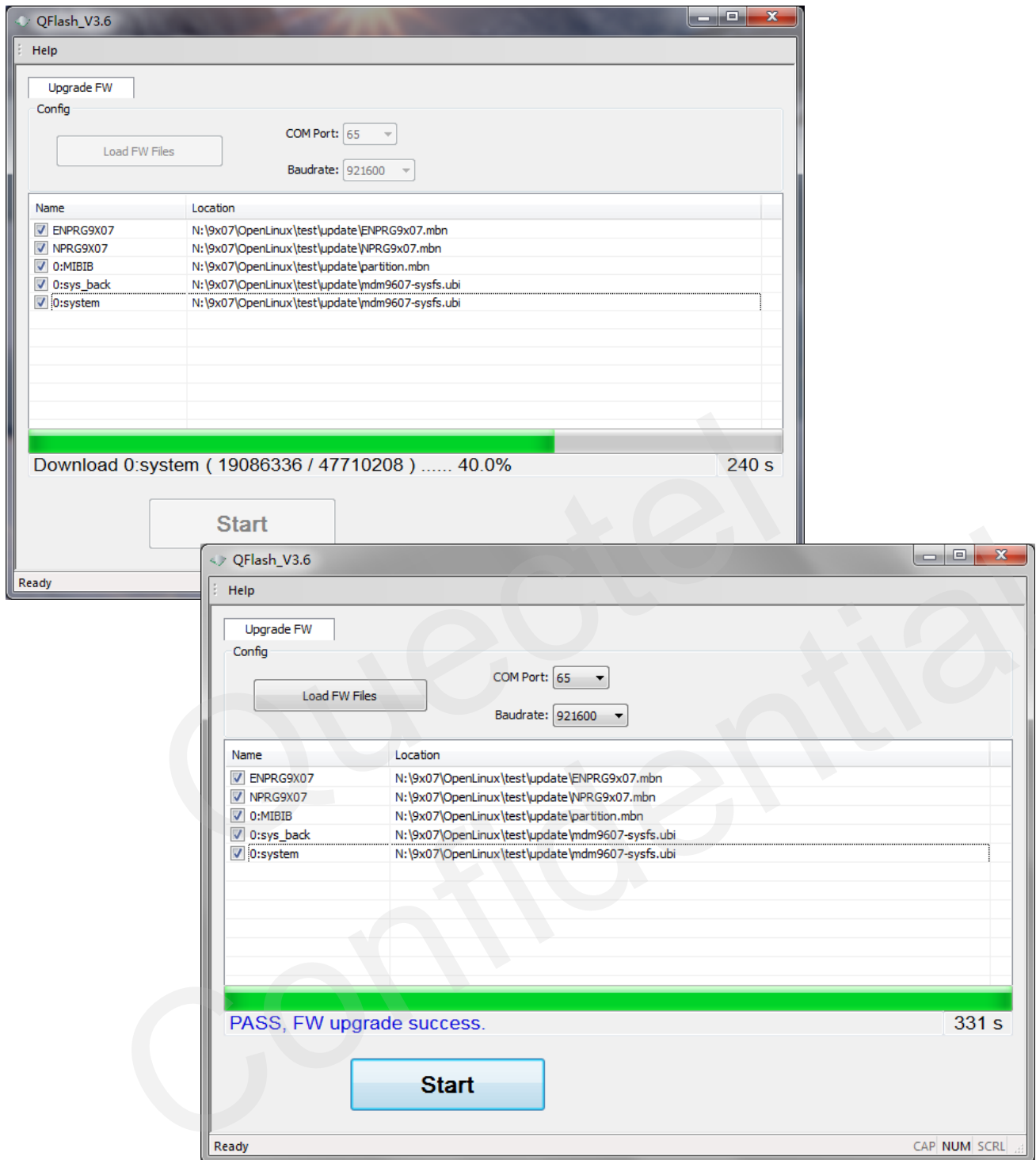


Figure 5: Download Application via QFlash

QuecOpen module will boot automatically after the download finishes, and the application will be loaded and run.

3.5. Launch Application

If developers want to download the executable file, they can use adb shell to login the Linux console of QuecOpen module and run the executable program directly.

```
stanley@stanley-OptiPlex-7020:~/MyWork/Quectel_QuecOpen_V1.0/example/helloWorld$ sudo adb
shell
/ # ls /home/root/ -l
total 4
-rwxrwxrwx    1 root    root          3692 Jul  5  2016 helloworld
/ # ./home/root/helloworld
<Hello QuecOpen !>
atoi("19.7")=19
/ #
```

If the application (.ubi) is downloaded by PC tool QFlash, the application will be loaded and run automatically after the downloading finishes.

If the application (.ubi) is downloaded by fastboot, there is a need to reset the module to run the application by the system automatically.

3.6. Debug Application

Generally, there are two ways to debug application.

- Output log message to standard output device by calling **print()**. And developers can catch the log through Debug UART or adb shell.
- Output log message through UART interface.

4 Programming Reference

4.1. System

4.1.1. Time

- Relevant header files

```
#include <time.h>
```

- API functions

/* Return a string in the form of "Day Mon dd hh:mm:ss yyyy\n" that is the representation of TP in this format. */

```
extern char *asctime (const struct tm *__tp);
```

/* Get resolution of clock CLOCK_ID. */

```
extern int clock_getres (clockid_t __clock_id, struct timespec *__res);
```

/* Get current value of clock CLOCK_ID and store it in TP. */

```
extern int clock_gettime (clockid_t __clock_id, struct timespec *__tp);
```

/* Set clock CLOCK_ID to value TP. */

```
extern int clock_settime (clockid_t __clock_id, const struct timespec *__tp);
```

- Example

Please see *example/time/example_time.c* for details.

4.1.2. Timer

- Relevant header files

```
#include <sys/timerfd.h>
```

```
#include <time.h>
```

```
#include <unistd.h>
```

- **API functions**

/* Return file descriptor for new interval timer source. */

```
extern int timerfd_create (clockid_t __clock_id, int __flags);
```

/* Set next expiration time of interval timer source UFD to UTMR. If FLAGS has the TFD_TIMER_ABSTIME flag set the timeout value is absolute. Optionally return the old expiration time in OTMR. */

```
extern int timerfd_settime (int __ufd, int __flags,  
                           const struct itimerspec *__utmr,  
                           struct itimerspec *__otmr);
```

/* Return the next expiration time of UFD. */

```
extern int timerfd_gettime (int __ufd, struct itimerspec *__otmr);
```

/* Read NBYTES into BUF from FD. Return the number read, -1 for errors or 0 for EOF. */

```
extern ssize_t read (int __fd, void *__buf, size_t __nbytes);
```

/* Create new per-process timer using CLOCK_ID. */

```
extern int timer_create (clockid_t __clock_id,  
                        struct sigevent *__restrict __evp,  
                        timer_t *__restrict __timerid);
```

/* Set timer TIMERID to VALUE, returning old value in OVALUE. */

```
extern int timer_settime (timer_t __timerid, int __flags,  
                          const struct itimerspec *__restrict __value,  
                          struct itimerspec *__restrict __ovalue);
```

/* Get current value of timer TIMERID and store it in VALUE. */

```
extern int timer_gettime (timer_t __timerid, struct itimerspec *__value);
```

- **Example**

Please see *example/timer/example_timer.c* for details.

4.1.3. Multitasking

- **Relevant header files**

```
#include <pthread.h>
```

- **API functions**

/ Create a new thread, starting with execution of START-ROUTINE getting passed ARG. Creation attributed come from ATTR. The new handle is stored in *NEWTHREAD. */*

```
extern int pthread_create (pthread_t *__restrict __newthread,  
                          const pthread_attr_t *__restrict __attr,  
                          void *(*__start_routine) (void *),  
                          void *__restrict __arg) __THROWNL __nonnull ((1, 3));
```

/ Terminate calling thread. The registered cleanup handlers are called via exception handling so we cannot mark this function with __THROW. */*

```
extern void pthread_exit (void *__retval) __attribute__((__noreturn__));
```

/ Make calling thread wait for termination of the thread TH. The exit status of the thread is stored in *THREAD_RETURN, if THREAD_RETURN is not NULL. */*

```
extern int pthread_join (pthread_t __th, void **__thread_return);
```

- **Example**

Please see *example/pthread/example_pthread.c* for details.

4.2. AT & URC

The tty device */dev/smd8* is designed to be AT port. Developers may simply open this device file and write AT commands through it.

Also, all URC messages are outputted through this port.

```
#define QUEC_AT_PORT    "/dev/smd8"  
smd_fd = open(QUEC_AT_PORT, O_RDWR | O_NONBLOCK | O_NOCTTY);  
iRet = write(smd_fd, "AT\r\n", 4);
```

- **Example**

Please see *example/at/example_at.c* for details.

4.3. I/O Interfaces

All kinds of multiplexing interfaces and the quantity of each kind are defined in **document [3]**.

4.3.1. GPIOs

All programmable GPIO pins are defined in **Table 2**. Except the dedicated pins, all other pins can be programmed as GPI or GPO.

- **Relevant header files**

```
#include "ql_om"  
#include "ql_gpio_def.h"  
#include "ql_gpio.h"  
#include "gpioSysfs.h"
```

- **API functions**

```
int QI_GPIO_Init(Enum_PinName    pinName,  
                 Enum_PinDirection dir,  
                 Enum_PinLevel   level,  
                 Enum_PinPullSel  pullSel  
                 );  
  
int QI_GPIO_SetLevel(Enum_PinName pinName, Enum_PinLevel level);  
int QI_GPIO_GetLevel(Enum_PinName pinName);  
int QI_GPIO_SetDirection(Enum_PinName pinName, Enum_PinDirection dir);  
int QI_GPIO_GetDirection(Enum_PinName pinName);  
int QI_GPIO_SetPullSelection(Enum_PinName pinName, Enum_PinPullSel pullSel);  
int QI_GPIO_GetPullSelection(Enum_PinName pinName);  
int QI_GPIO_Uninit(Enum_PinName pinName);
```

- **Example**

Please see *example/gpio/example_gpio.c* for details.

4.3.2. EINT

All programmable GPIO pins are defined in **Table 2**. Except the dedicated pins, all other pins can be programmed as GPI and interrupt.

- **Relevant header files**

```
#include "ql_oe.h"  
#include "ql_eint.h"
```

● API functions

```
int QI_EINT_Open(Enum_PinName eintPinName);
int QI_EINT_Enable(Enum_PinName eintPinName, Enum_EintType eintType);
int QI_EINT_Disable(Enum_PinName eintPinName);
int QI_EINT_Close(Enum_PinName eintPinName);
```

● Example

Please see *example/eint/example_eint.c* for details.

4.3.3. UARTs

QuecOpen module provides one debug UART and three UART interfaces for application. All of the three application UARTs support hardware handshaking. The pins are defined in the table below.

Table 3: UART Pins

Pin No.	Pin Name	Pin Location	Combined Interface (default)	Pin Multiplexing			Power Domain	Reset	Wake-up Interrupt	Remark
				Primary Function	Alternate Function 1	Alternate Function 2				
37	SPI_CS_N	Edge	SPI Interface	SPI_CS_N_ BLSP6	GPIO_22	UART_RTS_ BLSP6	1.8V	B-PD,L	✓	
38	SPI_MOSI	Edge		SPI_MOSI_ BLSP6	GPIO_20	UART_TXD_ BLSP6	1.8V	B-PD,L	✓	
39	SPI_MISO	Edge		SPI_MISO_ BLSP6	GPIO_21	UART_RXD_ BLSP6	1.8V	B-PD,L	✓	
40	SPI_CLK	Edge		SPI_CLK_ BLSP6	GPIO_23	UART_CTS_ BLSP6	1.8V	B-PU,H	x	BOOT_ CONFIG_4
41	I2C_SCL	Edge	I2C interface, host only	I2C_SCL_ BLSP2	GPIO_7	UART_CTS_ BLSP2	1.8V	B-PD,L	x	
42	I2C_SDA	Edge		I2C_SDA_ BLSP2	GPIO_6	UART_RTS_ BLSP2	1.8V	B-PD,L	x	
63	UART1_TXD	Edge	UART interface	UART_TXD_ BLSP2	GPIO_4	--	1.8V	B-PD,L	x	
66	UART1_RXD	Edge		UART_RXD_ BLSP2	GPIO_5	--	1.8V	B-PD,L	✓	
64	MAIN_CTS	Edge	Main UART	UART_CTS_BL SP3	GPIO_3	--	1.8V	B-PD,L	✓	
65	MAIN_RTS	Edge		UART_RTS_BL SP3	GPIO_2	--	1.8V	B-PD,L	x	
67	MAIN_TXD	Edge		UART_TXD_BL SP3	GPIO_0	--	1.8V	B-PD,L	x	

68	MAIN_RXD	Edge	UART_RXD_ BLSP3	GPIO_1	--	1.8V	B-PD,L	✓
----	----------	------	--------------------	--------	----	------	--------	---

The tty device for UART interfaces are defined as below.

Table 4: UART tty

Pin No.	Pin Name	Signal Direction	tty Device	Remark
63	UART1_TXD	UART TX	/dev/ttyHSL1	
66	UART1_RXD	UART RX		
38	SPI_MOSI	UART TX	/dev/ttyHSL2	Need to configure device tree first.
39	SPI_MISO	UART RX		
67	MAIN_TXD	UART TX	/dev/ttyHS0	
68	MAIN_RXD	UART RX		

As defined in the table above, the default function of pin 38/39 is SPI. So developers need to configure the Linux device tree to enable the UART option. The default pin function can be UART or GPIO by configuring Linux device tree. Please refer to the related document to configure device tree.

4.3.3.1. Pin 63/66 as UART

The group of pins generate the tty device `/dev/ttyHSL1` in Linux system. Developers may simply open this device to send/receive data.

If hardware handshaking is to be used, pin 41/42 (with CTS/RTS function) should be used together with pin 63/66. Pin 41 & 42 are multiplexing pins designed for I2C function by default, and they can offer alternate function of CTS/RTS. Developers have to switch the multiplexing mode of pin 41 & 42 by reconfiguring Linux tree before using them for hardware handshaking.

NOTE

By default, the UART interface (DCD/DTR) cannot be tested directly on EVB board. Something more (jump wire) need to be done to test the interface. There are two ways that can be utilized, as illustrated below.

- **Test Solution I: TE-A Module + EVB (×1)**

Step 1: Do not insert the TE-A module into EVB but independently supply power to it.

Step 2: Use jump wires to connect DCD/DTR on TE-A module to TXD_V1.8V/ RXD_V1.8V at J806 on EVB.

Step 3: Connect the Main UART port on EVB to PC for testing.

- **Test Solution II: TE-A Module + EVB (×2)**

Step 1: Prepare two EVB boards (EVB-A and EVB-B). Get rid of the resistors in DCD and DTR lines on EVB-A.

Step 2: Insert TE-A module into EVB-A. And use jump wires to connect DCD/DTR on EVB-A to TXD_V1.8V/RXD_V1.8V at J806 on EVB-B.

Step 3: Supply power to EVB-B.

Step 4: Connect the Main UART port on EVB-B to PC for testing.

4.3.3.2. Pin 38/39 as UART

The two pins are designed for SPI function by default. Developers have to switch the multiplexing mode of them by reconfiguring Linux device tree before programming the pins.

The group of pins generate the tty device `/dev/ttyHSL2` in Linux system. Developers may simply open this device to send/receive data.

If hardware handshaking is to be used, pin 37/40 (with RTS/CTS function) should be used together with pin 38/39. Pin 37 & 40 are multiplexing pins designed for SPI function by default, and they can offer alternate function of RTS/CTS. Developers have to switch the multiplexing mode of pin 37 & 40 by reconfiguring Linux tree before using them for hardware handshaking.

Test Method:

Use jump wire to connect the pins (pin 38/39) on QuecOpen module to the UART port on Quectel EVB or your own evaluation board. And the level conversion chip (TTL-to-232) is required.

4.3.3.3. Pin 67/68 as UART

On UMTS & LTE EVB, there are all pinouts for the full-featured main UART.

The group of pins generate the tty device `/dev/ttyHS0` in Linux system. Developers may simply open this device to transfer data.

If hardware handshaking is to be used, pin 64/65 (with primary function of CTS/RTS) should be used together with pin 67/68.

4.3.3.4. Programming reference

- Relevant header files

```
#include "ql_oe.h"  
#include "ql_uart.h"
```

- API functions

```
int Ql_UART_Open(const char* port, unsigned int baudrate, Enum_FlowCtrl flowCtrl);  
int Ql_UART_Read(int fd, char* buf, unsigned int buf_len);  
int Ql_UART_Write(int fd, const char* buf, unsigned int buf_len);  
int Ql_UART_SetDCB(int fd, ST_UARTDCB *dcb);  
int Ql_UART_GetDCBConfig(int fd, ST_UARTDCB *dcb);  
int Ql_UART_IoCtl(int fd, unsigned int cmd, void* pValue);  
int Ql_UART_Close(int fd);
```

/* Check the first NFDS descriptors each in READFDS (if not NULL) for read readiness, in WRITEFDS (if not NULL) for write readiness, and in EXCEPTFDS (if not NULL) for exceptional conditions. If TIMEOUT is not NULL, time out after waiting the interval specified therein. Returns the number of ready descriptors, or -1 for errors. */

```
extern int select (int __nfds, fd_set *__restrict __readfds,  
                  fd_set *__restrict __writefds,  
                  fd_set *__restrict __exceptfds,  
                  struct timeval *__restrict __timeout);
```

- Example

Please see example/uart/example_uart.c and example_uart_at.c for details.

4.3.4. ADC

In hardware, please refer to **document [2]** for the pin definition.

In software, developers may sample ADC value by sending **AT+QADC=0/1** through the tty device `/dev/smd8`.

```
Ql_SendAT ("AT+QADC=0", "+QADC:", 3000);  
usleep(100*1000); // delay 100ms for ADC reset  
Ql_SendAT ("AT+QADC=1", "+QADC:", 3000);
```

- **Example**

Please see *example/adc/example_adc.c* for details.

4.3.5. I2C

QuecOpen module provides one I2C interface. The I2C interface can be host only.

Table 5: I2C Pins

Pin No.	Pin Name	Pin Location	Combined Interface (default)	Pin Multiplexing			Power Domain	Reset	Wake-up Interrupt	Remark
				Primary Function	Alternate Function 1	Alternate Function 2				
41	I2C_SCL	Edge	I2C interface, host only	I2C_SCL_ BLSP2	GPIO_7	UART_CTS_ BLSP2	1.8V	B-PD,L	x	
42	I2C_SDA	Edge		I2C_SDA_ BLSP2	GPIO_6	UART_RTS_ BLSP2	1.8V	B-PD,L	x	

- **Relevant header files**

```
#include "ql_oe.h"
#include "ql_i2c.h"
```

- **API functions**

```
int QI_I2C_Init(unsigned char slaveAddr);
int QI_I2C_Read(int fd, unsigned short slaveAddr, unsigned char ofstAddr, unsigned char* ptrBuff, unsigned short length);
int QI_I2C_Write(int fd, unsigned short slaveAddr, unsigned char ofstAddr, unsigned char* ptrData, unsigned short length);
```

```
/* Perform the I/O control operation specified by REQUEST on FD. One argument may follow. Its presence and type depend on REQUEST. Return value depends on REQUEST. Usually -1 indicates error. */
```

```
//extern int ioctl (int __fd, unsigned long int __request, ...) __THROW;
```

- **Example**

Please see *example/i2c/example_i2c.c* for details.

4.3.6. SPI

QuecOpen module provides one SPI interface. The interface can be host only.

The baud rate of SPI can be up to 50MHz, and the default is 19.2MHz.

Table 6: SPI Pins

Pin No.	Pin Name	Pin Location	Combined Interface (default)	Pin Multiplexing			Power Domain	Reset	Wake-up Interrupt	Remark
				Primary Function	Alternate Function 1	Alternate Function 2				
37	SPI_CS_N	Edge	SPI Interface	SPI_CS_N_ BLSP6	GPIO_22	UART_RTS_ BLSP6	1.8V	B-PD,L	✓	
38	SPI_MOSI	Edge		SPI_MOSI_ BLSP6	GPIO_20	UART_TXD_ BLSP6	1.8V	B-PD,L	✓	
39	SPI_MISO	Edge		SPI_MISO_ BLSP6	GPIO_21	UART_RXD_ BLSP6	1.8V	B-PD,L	✓	
40	SPI_CLK	Edge		SPI_CLK_ BLSP6	GPIO_23	UART_CTS_ BLSP6	1.8V	B-PU,H	x	BOOT_ CONFIG_4

4.3.6.1. SPI Mode

In QuecOpen system, the SPI device drivers have been compiled to .ko modules. There are two .ko modules existing in the Linux system of QuecOpen module: spidev.ko, quec_spi.

```
cd /usr/lib/modules/3.18.20/kernel/drivers/spi#
ls
quec_spi_chn.ko  spidev.ko
```

The two SPI driver modules are for different application cases.

- **4-line mode:** spidev.ko

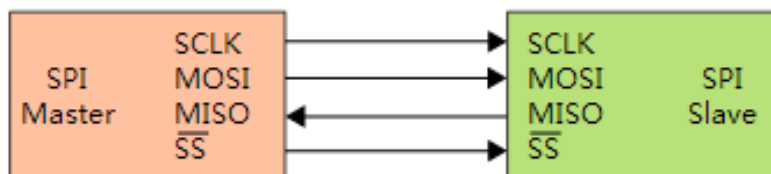


Figure 6: SPI 4-Line Mode

- **6-line mode:** quec_spi_chn.ko

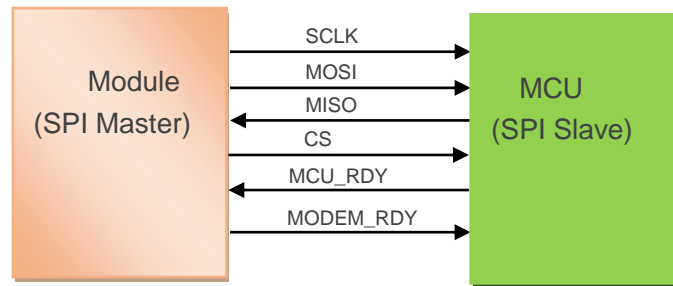


Figure 7: SPI 6-Line Mode

In 6-line mode SPI, from the module side, MCU_RDY should be an interrupt pin that can wake up the module, and MODEM_RDY can be any GPIO. Please refer to **Table 2** for details.

4.3.6.2. Install SPI Driver

By default, the SPI device is not present. Developers need to install the SPI driver (.ko module) before accessing to SPI device.

- **4-line mode:** spidev.ko

```
//Uninstall SPI driver module.
rmmod spidev

//Install 4-line mode SPI driver.
insmod spidev.ko busnum=6 chipselect=0

// install 4-line mode SPI driver with speed parameter (the max speed is 19.2MHz by default).
insmod spidev.ko busnum=6 chipselect=0 maxspeed=50000000

sleep(1);
```

After the SPI driver module is installed, the SPI device `/dev/spidev6.0` will be generated. Developers may simply open this device to send/receive SPI data.

- **6-line mode:** quec_spi_chn.ko

```
//Uninstall SPI driver module.
rmmod quec_spi_chn

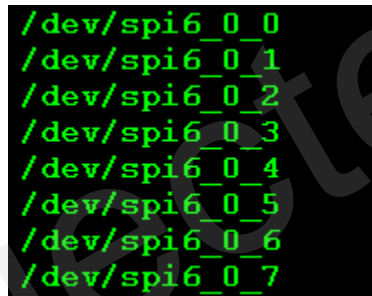
//Install 6-line mode SPI driver.
insmod quec_spi_chn.ko busnum=6 chipselect=0 gpiomodemready=78 gpiomcuready=79
```

```
//Install 6-line mode SPI driver with speed parameter (the max speed is 19.2MHz by default).
insmod quec_spi_chn.ko busnum=6 chipselect=0 gpiomodemready=78 gpiomcuready=79
maxspeed=50000000

//Install 6-line mode SPI driver with SPI mode and speed parameter.
insmod quec_spi_chn.ko busnum=6 chipselect=0 gpiomodemready=78 gpiomcuready=79 spimode=0
maxspeed=50000000

sleep(1);
```

After the SPI driver module is installed, developers may get 8 SPI devices that indicate 8 SPI channels, which can be used for different business data.



```
/dev/spi6_0_0
/dev/spi6_0_1
/dev/spi6_0_2
/dev/spi6_0_3
/dev/spi6_0_4
/dev/spi6_0_5
/dev/spi6_0_6
/dev/spi6_0_7
```

Figure 8: SPI 8 Channels

Developers may execute the previous commands by system call when initializing SPI in application.

```
system("insmod xxxxx");
sleep(1);
```

4.3.6.3. SPI Parameters

- SPI customized parameters

<busnum>	SPI bus number. Fixed to 6.
<chipselect>	0 is active.
<maxspeed>	The default speed is 19.2MHz, and the max speed can be up to 50MHz. The possible values (unit: Hz): {960000, 4800000, 9600000, 16000000, 19200000, 25000000, and 50000000}.
<spimode>	It is decided by SPI device. 0: Mode 0 CPOL=0, CPHA=0 1: Mode 1 CPOL=0, CPHA=1 2: Mode 2 CPOL=1, CPHA=0

<gpiomodemredy>	3: Mode 3 CPOL=1, CPHA=1 A GPIO number which indicates the modem is ready (not in sleep state), and simultaneously wake up the external MCU. Please refer to the “Pin Multiplexing” field for the GPIO number in Table 2 .
<gpiomcuready>	A GPIO number which indicates the external MCU is ready (not in sleep state), and simultaneously wake up the module. Please refer to the “Pin Multiplexing” field for the GPIO number in Table 2 .

- **Relevant header files**

```
#include "ql_oe.h"
```

- **API functions**

Developers may directly call the APIs in standard library, such as **open()**, **ioctl()**, to access and control the SPI device */dev/spidev6.0*.

- **Example**

Please see *example/spi/example_spi.c* for details.

4.4. File System

- **Relevant header files**

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
```

- **API functions**

```
/* Set file access permissions for FILE to MODE. If FILE is a symbolic link, this affects its target instead.
*/
```

```
extern int chmod (const char *__file, __mode_t __mode);
```

```
/* Get file attributes for FILE and put them in BUF. */
```

```
extern int stat (const char *__restrict __file, struct stat *__restrict __buf);
```

```
/* Open a file and create a new stream for it. */
```

```
extern FILE *fopen (const char *__restrict __filename, const char *__restrict __modes);
extern FILE *freopen (const char *__restrict __filename,
                     const char *__restrict __modes,
                     FILE *__restrict __stream) __wur;
```

/* Seek to a certain position on STREAM. */

```
extern int fseek (FILE *__stream, long int __off, int __whence);
```

/* Read chunks of generic data from STREAM. */

```
extern size_t fread (void *__restrict __ptr, size_t __size, size_t __n, FILE *__restrict __stream);
```

/* Read formatted input from STREAM. */

```
extern int fscanf (FILE *__restrict __stream, const char *__restrict __format, ...);
```

/* Write chunks of generic data to STREAM. */

```
extern size_t fwrite (const void *__restrict __ptr, size_t __size, size_t __n, FILE *__restrict __s);
```

/* Return the current position of STREAM. */

```
extern long int ftell (FILE *__stream);
```

/* Remove file FILENAME. */

```
extern int remove (const char *__filename);
```

/* Rename file OLD to NEW. */

```
extern int rename (const char *__old, const char *__new);
```

/* Close STREAM. */

```
extern int fclose (FILE *__stream);
```

/* Create a new directory named PATH, with permission bits MODE. */

```
extern int mkdir (const char *__path, __mode_t __mode);
```

● Example

Please see *example/file/example_file.c* for details.

4.5. SD Card/eMMC Flash

QuecOpen module provides a group of SDIO interface that can connect to SD card or eMMC flash. The pins are defined in **document [2]** and **[3]**.

By default, either SD card or eMMC flash is mounted to */media/sdcard/*.

4.6. Audio

Audio option includes record and playback. The record data is outputted in PCM stream.

QuecOpen™ supports mixer player, and supports playing audio and TTS at the same time.

- Relevant header files

```
#include "ql_oe.h"
#include "ql_audio.h"
```

- API functions

```
int Ql_AudPlayer_Open(char* device, _cb_onPlayer cb_func);
typedef int(*_cb_onPlayer)(int hdl, int result);
int Ql_AudPlayer_Play(int hdl, unsigned char* pData, unsigned int length);
int Ql_AudPlayer_PlayFrmFile(int hdl, int fd, int offset);
int Ql_AudPlayer_Pause(int hdl);
int Ql_AudPlayer_Resume(int hdl);
void Ql_AudPlayer_Stop(int hdl);
void Ql_AudPlayer_Close(int hdl);
```

```
typedef int(*_cb_onRecorder)(int result, unsigned char* pBuf, unsigned int length);
int Ql_AudRecorder_Open(char* device, _cb_onRecorder cb_fun);
int Ql_AudRecorder_StartRecord(void);
int Ql_AudRecorder_Pause(void);
int Ql_AudRecorder_Resume(void);
void Ql_AudRecorder_Stop(void);
void Ql_AudRecorder_Close(void);
```

- Example

Please see *audio/example_audio.c* for details.

4.7. TTS

Developers may start/stop TTS player by sending **AT+QTTS** through the tty device */dev/smd8*.

```
Ql_SendAT("ATE0", "OK", 1000);
Ql_SendAT("AT", "OK", 1000);
Ql_SendAT("ATI", "OK", 3000);
Ql_SendAT("AT+QTTS=2,\"a b c d e f,g,h,i,j,k\"\\0\", \"+QTTS:\", 60*1000);
usleep(300*1000); // delay 300ms for tts player ready
Ql_SendAT("AT+QTTS=2,\"hi, it's friday today\"\\0\", \"+QTTS:\", 60*1000);
usleep(300*1000); // delay 300ms for tts player ready
Ql_SendAT("AT+QTTS=1,\"4F60597DFF0C4ECA5929661F671F4E94\"\\0\", \"+QTTS:\", 60*1000); // 你好，今天星期五
```


- **Example**

Please see `tts/example_tts.c` for details.

4.8. Voice Call

Developers can open the AT port `/dev/smd8` to send AT commands to program telephony option. Please refer to **document [4]** for the usage of related AT commands.

4.9. SMS

Developers can open the AT port `/dev/smd8` to send AT commands to program SMS option. Please refer to **document [4]** for the usage of related AT commands.

4.10. Network Service

- **Relevant header files**

```
#include "ql_nw.h"
```

- **API functions**

```
int QI_NW_GetSignalQuality(unsigned int* rssi, unsigned int* ber);
int QI_NW_GetRegState(int* state);
int QI_NW_GetServingCell(ST_CellInfo* cell);
int QI_NW_GetNeighborCell(ST_CellInfo* cell, unsigned int cellCnt);
int QI_NW_GetRadioAccessTech(void);
int QI_NW_GetMccMnc(char* mcc, char* mnc);
int QI_NW_GetNetworkName(char* ptrNetName, unsigned int size);
```

4.11. Data Service

QuecOpen™ uses the component “DSI_NetCtrl” to perform data services call. DSI_NetCtrl just controls and manages the data activities and will not provide methods for data transfer explicitly. Developers can adopt BSD sockets or Unix sockets to create data connections.

4.11.1. DSI_NetCtrl

DSI_NetCtrl library API contains a common interface for clients to perform data services call control in the applications that requires WWAN data services functionality.

Compared to data service QCMAP, DSI_NetCtrl enables developers to create multiple profiles for different PDN. So if there is a need to program dual-APN, then this way should be selected for application development.

- **Relevant header files**

```
#include "ql_oe.h"  
#include "dsi_netctrl.h"  
#include "ds_util.h"
```

- **API functions**

```
//Initialize the DSI_NetCtrl library  
extern int dsi_init_ex( int mode,void (* dsi_init_cb_func)( void * ),void *dsi_init_cb_data );  
  
//Get data service handle. All subsequent functions use this handle as an input parameter.  
extern dsi_hdl_t dsi_get_data_srvc_hdl( dsi_net_ev_cb cb_fn, void * user_data );  
  
//Release a data service handle. All resources associated with the library are released.  
extern void dsi_rel_data_srvc_hdl(dsi_hdl_t hndl);  
  
//Set the data call parameter before trying to start a data call.  
//Set IP type, APN, user name and password.  
extern int dsi_set_data_call_param( dsi_hdl_t hndl, dsi_call_param_identifier_t identifier,  
dsi_call_param_value_t *info );  
  
//Start/stop a data call.  
extern int dsi_start_data_call(dsi_hdl_t hndl);  
extern int dsi_stop_data_call(dsi_hdl_t hndl);  
  
//Get the number of IP addresses (IPv4 and global IPv6) associated with the DSI interface.  
extern unsigned int dsi_get_ip_addr_count( dsi_hdl_t hndl );  
  
//Get the IP address information structure (network order).  
extern int dsi_get_ip_addr( dsi_hdl_t hndl, dsi_addr_info_t * info_ptr, int len );  
  
//Return the current data bearer technology on which a call was brought up  
(GSM/WCDMA/HSPA/LTE/...).  
dsi_data_bearer_tech_t dsi_get_current_data_bearer_tech( dsi_hdl_t hndl );
```

```
//Query the reason for a call being terminated.
extern int dsi_get_call_end_reason( dsi_hndl_t hndl, dsi_ce_reason_t * ce_reason, dsi_ip_family_t ipf );

//Clean-up the DSI_NetCtrl library.
extern int dsi_release(int mode);
```

Please refer to **document [5]** for detailed information about DSI_NetCtrl.

● Example

Before starting data call, parameters for DSI_NetCtrl should be set first. The following are the minimum settings.

//Set profile ID for both CDMA and UMTS. These settings enable the program to cover both CDMA and Non-CDMA networks.

```
param_info.buf_val = NULL;
param_info.num_val = 0;;
dsi_set_data_call_param(dsi_net_hndl.handle, DSI_CALL_INFO_UMTS_PROFILE_IDX, &param_info);

param_info.buf_val = NULL;
param_info.num_val = 0;
dsi_set_data_call_param(dsi_net_hndl.handle, DSI_CALL_INFO_CDMA_PROFILE_IDX, &param_info);
```

//Set IP version (IPV4, IPV6, IPV4V6).

```
param_info.buf_val = NULL;
param_info.num_val = DSI_IP_VERSION_4_6;
dsi_set_data_call_param(dsi_net_hndl.handle, DSI_CALL_INFO_IP_VERSION, &param_info);
```

//Set APN and user name/password.

```
param_info.buf_val = strdup(dsi_net_hndl.apn);
param_info.num_val = strlen(param_info.buf_val);
dsi_set_data_call_param(dsi_net_hndl.handle, DSI_CALL_INFO_APN_NAME, &param_info);

param_info.buf_val = strdup(dsi_net_hndl.user);
param_info.num_val = strlen(param_info.buf_val);
dsi_set_data_call_param(dsi_net_hndl.handle, DSI_CALL_INFO_USERNAME, &param_info);

param_info.buf_val = strdup(dsi_net_hndl.password);
param_info.num_val = strlen(param_info.buf_val);
dsi_set_data_call_param(dsi_net_hndl.handle, DSI_CALL_INFO_PASSWORD, &param_info);
```

//Start a data call (Activate PDN, get IP, and feedback via callback).

```
iRet = dsi_start_data_call(dsi_net_hndl.handle);
```

Please see *example /data_service/dsi_netctrl/example_dsi_netctrl.c* for details.

4.11.2. Sockets

In QuecOpen[™], Unix sockets or BSD sockets can be used to establish data connections.

- **Relevant header files**

```
#include <sys/types.h>
#include <sys/socket.h>
#include <stdio.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/shm.h>
#include <pthread.h>
#include "ql_network.h"
```

- **API functions**

/* Create a new socket of type TYPE in domain DOMAIN, using protocol PROTOCOL. If PROTOCOL is zero, one is chosen automatically. Returns a file descriptor for the new socket, or -1 for errors. */

```
extern int socket (int __domain, int __type, int __protocol);
```

/* Open a connection on socket FD to peer at ADDR (which is LEN bytes long). For connectionless socket types, just set the default address to send to and the only address from which to accept transmissions. Return 0 on success, -1 for errors. */

```
extern int connect (int __fd, __CONST_SOCKADDR_ARG __addr, socklen_t __len);
```

/* Send N bytes of BUF to socket FD. Returns the number sent or -1. */

```
extern ssize_t send (int __fd, const void *__buf, size_t __n, int __flags);
```

/* Read N bytes into BUF from socket FD. Returns the number read or -1 for errors. */

```
extern ssize_t recv (int __fd, void *__buf, size_t __n, int __flags);
```

/* Send N bytes of BUF on socket FD to peer at address ADDR (which is ADDR_LEN bytes long). Returns the number sent, or -1 for errors. */

```
extern ssize_t sendto (int __fd, const void *__buf, size_t __n,
                      int __flags, __CONST_SOCKADDR_ARG __addr,
                      socklen_t __addr_len);
```

/* Read N bytes into BUF through socket FD. If ADDR is not NULL, fill in *ADDR_LEN bytes of it with the address of the sender, and store the actual size of the address in *ADDR_LEN. Returns the number of bytes read or -1 for errors. */

```
extern ssize_t recvfrom (int __fd, void *__restrict __buf, size_t __n,  
                        int __flags, __SOCKADDR_ARG __addr,  
                        socklen_t *__restrict __addr_len);
```

/* Prepare to accept connections on socket FD. N connection requests will be queued before further requests are refused. Returns 0 on success, -1 for errors. */

```
extern int listen (int __fd, int __n);
```

/* Await a connection on socket FD. When a connection arrives, open a new socket to communicate with it, set *ADDR (which is *ADDR_LEN bytes long) to the address of the connecting peer and *ADDR_LEN to the address's actual length, and return the new socket's descriptor, or -1 for errors. */

```
extern int accept (int __fd, __SOCKADDR_ARG __addr, socklen_t *__restrict __addr_len);
```

/* Shut down all or part of the connection open on socket FD.

HOW determines what to shut down:

SHUT_RD = No more receptions;

SHUT_WR = No more transmissions;

SHUT_RDWR = No more receptions or transmissions.

Returns 0 on success, -1 for errors. */

```
extern int shutdown (int __fd, int __how);
```

/* Check the first NFDS descriptors each in READFDS (if not NULL) for read readiness, in WRITEFDS (if not NULL) for write readiness, and in EXCEPTFDS (if not NULL) for exceptional conditions. If TIMEOUT is not NULL, time out after waiting the interval specified therein. Returns the number of ready descriptors, or -1 for errors. */

```
extern int select (int __nfd, fd_set *__restrict __readfds,  
                 fd_set *__restrict __writefds,  
                 fd_set *__restrict __exceptfds,  
                 struct timeval *__restrict __timeout);
```

● Example

Please see *example/data_service/tcp_client/example_tcpClient.c* for details.

4.12. GNSS

The GNSS chip is built in the module. Developers just need to enable the GNSS option to retrieve NMEA statements.

The GNSS NMEA can be outputted through USB-NMEA port, Debug UART port and Linux SMD7 port. In QuecOpen™ application, developers may get NMEA by opening `/dev/smd7`.

Please follow the three steps below to get GNSS NMEA:

Step 1: Open `/dev/smd7`

Step 2: Set the output port of GNSS NMEA to SMD port.

Send **AT+QGPSCFG="outport","linuxsmd"** through AT port (`/dev/smd8`).

Step 3: Enable GNSS.

Send **AT+QGPS=1** to enable GNSS option through AT port (`/dev/smd8`).

Please refer to document [6] or [7] for more AT commands about GNSS configurations.

- **Example**

Please see `example/gnss/example_gps.c` for details.

4.13. Wi-Fi

QuecOpen module provides SDIO interface that can interface to Wi-Fi chip. Also, Quectel provides Wi-Fi module solution (FC20 module) that provides built-in Wi-Fi and Bluetooth functions. That means developers have two choices to realize Wi-Fi functions: using FC20 module as the Wi-Fi component, or using their own Wi-Fi chip. The relevant pins (pin 129-138) are defined in **Table 2**.

- **Example**

If Quectel FC20 module is used as the Wi-Fi component, please refer to `example/wifi/example_wifi.c`.

4.14. (U)SIM Card

- Relevant header files

```
#include "ql_sim.h"
```

- API functions

```
int QI_SIM_GetState(int* state);  
int QI_SIM_GetICCID(char *iccid);  
int QI_SIM_GetIMSI(char *imsi);  
int QI_SIM_GetPINTRIESCnt();  
int QI_SIM_EnterPIN(const char* pin);
```

Quectel
Confidential

5 Customization

5.1. Pin Multiplexing Function

All programmable GPIO pins defined in **Table 2** have the primary function by default when booting. However, some developers may hope it is one of the alternative functions when the module boots.

Take pin 38/39 as an example:

The pins are designed for SPI option by default when booting. It means the SPI driver will be loaded for the two pins when Linux system boots. If the two pins need to be configured as UART option, there are two methods available. Basically the two methods work on reconfiguring Linux device tree (DTS), and the details are illustrated in the following two sub-chapters.

5.1.1. With Linux Kernel Code

Developers can configure the several files shown as below:

```
kernel/arch/arm/boot/dts/qcom/mdm9607.dtsi  
kernel/arch/arm/boot/dts/qcom/mdm9607-pinctrl.dtsi  
kernel/arch/arm/boot/dts/qcom/mdm9607-cdp.dtsi  
kernel/arch/arm/boot/dts/qcom/mdm9607-mtp.dtsi
```

5.1.2. Without Linux Kernel Code

The DTS is included in the file *mdm9607-perf-boot.img*, which is one of released files in firmware. Please follow the steps below to finish configuration.

Step 1: Unpack *mdm9607-perf-boot.img*. The outputted files are as below.







 dt.img	475.1 kB	unknown
 dt_match_best.img	94.2 kB	unknown
 mdm9607.dtb	94.2 kB	unknown
 mdm9607.dts	122.4 kB	plain text document
 mkboot.sh	337 bytes	shell script
 zimage	5.3 MB	unknown

Figure 9: Disassemble boot.img

Step 2: Change .dts file.

Step 3: Re-pack these files to generate a new file new_boot.img.

Step 4: Download the new_boot.img.

5.2. Rootfs

In QuecOpen™, the application is packed into rootfs when compiling. The relevant processing is defined in the following script files in SDK.

SDK/build.sh
SDK/tools/packapp

If there is a need to customize the rootfs, such as creating folder, adding files, developers can change the script files to achieve it.

6 Appendix A References

Table 7: Related Documents

SN	Document Name	Remark
[1]	Quectel_WCDMA<E_Linux_USB_Driver_User_Guide	USB driver installation guide for UMTS/HSPA/LTE modules
[2]	Quectel_EC2x_QuecOpen_Hardware_Design	Hardware design for QuecOpen modules
[3]	Quectel_EC20 R2.0&EC21&EC25_QuecOpen_GPIO_Assignment_Spreadsheet	All multiplexing pins available in QuecOpen modules
[4]	Quectel_EC2x_AT_Commands_Manual	AT commands manual for EC2x modules
[5]	Qualcomm_DSI_NetCtrl_Library_API_Interface_Specification	API library for network management
[6]	Quectel_EC20_R2.0_GNSS_AT_Commands_Manual	EC20 R2.0 GNSS AT commands manual
[7]	Quectel_EC25&EC21_GNSS_AT_Commands_Manual	EC25&EC21 GNSS AT commands manual

Table 8: Terms and Abbreviations

Abbreviation	Description
ADB	Android Debug Bridge
API	Application Program Interface
CMOS	Complementary Metal-Oxide-Semiconductor Transistor
DSI_NetCtrl	Data Service Interface Network Controller
FOTA	Firmware over The Air
GNSS	Global Navigation Satellite System
GPIO	General-purpose Input/Output

I2C	Inter-Integrated Circuit
MIPS	Million Instructions Per Second
NMEA	National Marine Electronics Association
PCM	Pulse Code Modulation
QCMAP	Qualcomm Mobile Access Point
QuecOpen™	The Open Embedded Development Platform of Quectel Modules
RAM	Random Access Memory
SDIO	Secure Digital Input and Output Card
SDK	Software Development Kit
SGMII	Serial Gigabit Media Independent Interface
SPI	Serial Peripheral Interface
TTS	Text to Speech
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
WLAN	Wireless Local Area Network