EC2x-Quecopen
Modifying and Configuring UART Console

LTE Standard Module Series

Rev. EC2x-Quecopen_Modifying_and _Configuring_UART_Console_V1.0

Date: 2018-05-31

Status: Preliminary

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

Quectel Wireless Solutions Co., Ltd.
7th Floor, Hongye Building, No.1801 Hongmei Road, Xuhui District, Shanghai 200233, China
Tel: +86 21 5108 6236
Email: info@quectel.com

Or our local office. 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 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

About the Document

History

Revision	Date	Author	Description
1.0	2018-05-31	Matthew MA	Initial

Contents

1 Introduction

This document mainly introduces how to configure UART, modify or cancel the console on EC2x modules in order to support the customers to select UART functions and configure specified consoles according to specific situations. Please refer to *Quectel_EC2X-QuecOpen_UART_Development_Guide.pdf* for UART introductions.

This document mainly applies to the global market. Currently LTE Standard modules that support this includes:

- EC2x: EC20 R2.1/EC25/EC21

2 Debug UART Release

The default console of Linux is debug UART. To change the console, please first release the debug UART.

There are mainly three steps to release the debug UART.

A. Disabling log output of aboot;
B. Disabling log output in the process of Linux booting;
C. Cancelling Linux console.

2.1. Disabling Log Output of Aboot

During the process of Aboot booting, it will print messages to debug UART by default. To cancel these printed messages, it is necessary to modify and recompile aboot.

A. Turn off aboot print macro.

```
$ vim project/mdm9607.mk +19
```

```
ol@ql-Ubuntu:~/Perforce/Matthew_Linux_PC01/Qualcomm/MDM9x07/OpenLinux/MCU_R06_update01/apps_proc/oe-core/build/tmp-glibc/deploy/sdk/ql-ol-sdk$ ls
EC20CE_FAG_changelist  Makefile  ql-ol-bootloader  ql-ol-crosstool  ql-ol-crosstool.tar.bz2  ql-ol-extsdk  ql-ol-kernel  ql-ol-rootfs  ql-ol-usrdata  target
ol@ql-Ubuntu:~/Perforce/Matthew_Linux_PC01/Qualcomm/MDM9x07/OpenLinux/MCU_R06_update01/apps_proc/oe-core/build/tmp-glibc/deploy/sdk/ql-ol-sdk$ vim ql-ol-bootloader/project/mdm9607.mk +19
ol@ql-Ubuntu:~/Perforce/Matthew_Linux_PC01/Qualcomm/MDM9x07/OpenLinux/MCU_R06_update01/apps_proc/oe-core/build/tmp-glibc/deploy/sdk/ql-ol-sdk$
```

Assign WITH_DEBUG_UART to 0.

```
13 endif
14
15 ifeq ($(TARGET_BOOTIMG_SIGNED),true)
16 CFLAGS += -D_SIGNED_KERNEL=1
17 endif
18
19 DEFINES += WITH_DEBUG_UART=0
20 DEFINES += WITH_DEBUG_LOG_BUF=1
21 DEFINES += DEVICE_TREE=1
22 DEFINES += CONTIGUOUS_MEMORY=1
23
24 DEFINES += SPMI_CORE_V2=1
25 DEFINES += BAM_V170=1
```

B. Recompile aboot

```
$ make kernel
```

2.2. Disabling Log Output in the Process of Linux Booting

During the process of Linux booting, it will print some messages. To remove these messages, it is necessary to modify the boot parameters that aboot passes to Linux kernel.

A. To disable log output in the process of Linux booting, please execute the following command in sdk directory ($ is the command line prompt).

```
$ sed -i 's/console=ttyHSL0,115200,n8/console=disable/g' ql-ol-extsdk/tools/quectel_mkboot/mkqcomboot
```

B. Recompile kernel.

```
$ make kernel_menuconfig
$ make kernel
```

NOTE

1. Restore the log when Linux starts by executing **sed -i 's/console=disable/console=ttyHSL0,115200,n8/g' ql-ol-extsdk/tools/quectel_mkboot/mkqcomboot**.
2. To disable log, please do not execute **make debug_kernel_menuconfig** to configure the kernel, otherwise the kernel will crash.

2.3. Cancelling Linux Console

The Linux console specifies in file /etc/inittab of rootfs. Cancel Linux console by modifying this file.

First, open the file inittab, e.g. vim ql-ol-rootfs/etc/inittab.

```
ps_proc/oe-core/build/tmp-glibc/deploy/sdk/ql-ol-sdk$ vim ql-ol-rootfs/etc/inittab
```

Comment out the sentences of the specified console.

```
45 m2:5:respawn:/usr/bin/mbimd
46 m3:5:once:/sbin/usb/compositions/quec_mbim_check
47
48 #S:2345:respawn:/sbin/getty -L ttyHSL0 115200 console
-- 插入 --
```

Recompile rootfs:

```
$ make rootfs
```

2.4. Verification

After completing the steps in ***Chapter 2.1, 2.1, 2.3,*** use ADB to re-download aboot, kernel and rootfs. If debug UART no longer has **any output** after booting and it **can be logged in via ADB shell**, it means that the debug UART is successfully released.

3 Enabling UART1

Debug UART is usually enabled in the kernel. To configure other UART, it only needs to configure the corresponding DTS. This chapter takes UART1 as an example to introduce all the steps to configure one UART.

The UART driver on Linux has already existed. When kernel starts, the driver will probe the device to check if it is enabled. As long as the UART1 is enable in DTS, Linux can load it normally.

First, open the comparison table of the pins and check the pins and the names of UART1. As shown in the figure below, it can be seen that the pins of UART1 are GPIO4 and GPIO5. If hardware flow control is applied, GPIO6 and GPIO7 will be needed. The name of UART1 in DTS is UART2.

53	41	I2C_SCL	Edge	I2C interface, host only	I2C_SCL_BLSP2	GPIO_7	UART_CTS_BLSP2	GPIO_7
54	42	I2C_SDA	Edge		I2C_SDA_BLSP2	GPIO_6	UART_RTS_BLSP2	GPIO_6
55	62	GPIO6	Edge		GPIO_75	--	--	GPIO_75
56	63	UART1_TXD	Edge	UART interface	UART_TXD_BLSP2	GPIO_4		GPIO_4
57	66	UART1_RXD	Edge		UART_RXD_BLSP2	GPIO_5		GPIO_5

Under the directory of kernel, find mdm9607-mtp.dtsi under directory *arch/arm/boot/dts/qcom/*. This file is the main switch of the peripheral device. Find blsp1_uart2 and enable UART1 by changing the status from disable to ok.

```
00055:     status = "ok";
00056: };
00057:
00058: &blsp1_uart2 {
00059:     status = "ok";  //if need, user can enable by themselves
00060:     pinctrl-names ="sleep", "default";
00061:     pinctrl-0 = <&blsp1_uart2_sleep>;
00062:     pinctrl-1 = <&blsp1_uart2_active>;
00063: };
00064:
00065: &blsp1_uart6 {
00066:     status = "disabled";
```

4 Configuring UART1 Flow Control Pins (Optional)

4.1. Viewing Pins in UART1

The node of UART1 in DTS file is blsp1_uart2.

```
00055:     status = "ok";
00056: };
00057:
00058: &blsp1_uart2 {
00059:     status = "ok";  //if need, user can enable by themselves
00060:     pinctrl-names ="sleep", "default";
00061:     pinctrl-0 = <&blsp1_uart2_sleep>;
00062:     pinctrl-1 = <&blsp1_uart2_active>;
00063: };
00064:
00065: &blsp1_uart6 {
00066:     status = "disabled";
```

Find the pinctrl attribute of &blsp1_uart2 in mdm9607-mtp.dtsi or mdm9607.dtsi under the directory of *arch/arm/boot/dts/qcom/*. As shown in the figure above, the pinctrl attribute quotes **blsp1_uart2_active** and **blsp1_uart2_sleep**. blsp1_uart2_active and blsp1_uart2_sleep can be found in mdm9607-pinctrl.dtsi under the same directory. Please see the figure shown below.

```
00085:         blsp1_uart2_sleep: blsp1_uart2_sleep {
00086:             mux {
00087:                 pins = "gpio4", "gpio5";
00088:                 function = "gpio";
00089:             };
00090:             config {
00091:                 pins = "gpio4", "gpio5";
00092:                 drive-strength = <2>;
00093:                 bias-pull-down;
00094:                 output-low;
00095:             };
00096:         };
00097:
00098:         blsp1_uart2_active: blsp1_uart2_active {
00099:             mux {
00100:                 pins = "gpio4", "gpio5";
00101:                 function = "blsp_uart2";
00102:             };
00103:             config {
00104:                 pins = "gpio4", "gpio5";
00105:                 drive-strength = <2>;
00106:                 bias-disable;
00107:             };
00108:         };
```

Compare the pins of UART1:

52	40	SPI_CLK	Edge		SPI_CLK_BLSP6	GPIO_23	UART_CTS_BLSP6	GPIO_23	KEEPER	GPIO_23	IN/PD/2	SPI_CLK_BLSP6	out/NP
53	41	I2C_SCL	Edge	I2C interface, host only	I2C_SCL_BLSP2	GPIO_7	UART_CTS_BLSP2	GPIO_7	KEEPER	GPIO_7	IN/PD/2	I2C_SCL	out/N
54	42	I2C_SDA	Edge		I2C_SDA_BLSP2	GPIO_6	UART_RTS_BLSP2	GPIO_6	KEEPER	GPIO_6	IN/PD/2	I2C_SDA	NP/
55	62	GPIO6	Edge		GPIO_75	--	--	GPIO_75	KEEPER	GPIO_75	In/PD	GPIO_75	In/PD
56	63	UART1_TXD	Edge	UART interface	UART_TXD_BLSP2	GPIO_4		GPIO_4	out/PD/2	UART_TXD_BLSP2	out/NP/2	UART_TXD_BLSP2	out/N
57	66	UART1_RXD	Edge		UART_RXD_BLSP2	GPIO_5		GPIO_5	out/PD/2	UART_RXD_BLSP2	IN/NP	UART_RXD_BLSP2	IN/NP
58	64	MAIN_CTS	Edge	MAIN UART	UART_CTS_BLSP3	GPIO_3	--	GPIO_3	out-Low/NP/2	UART_CTS_BLSP3	IN/NP/2	UART_CTS_BLSP3	out/N
59	65	MAIN_RTS	Edge		UART_RTS_BLSP3	GPIO_2	--	GPIO_2	out-Low/NP/2	UART_RTS_BLSP3	IN/NP/2	UART_RTS_BLSP3	IN/NP

Currently UART1 only uses two pins: RX and TX. Because GPIO6 and GPIO7 are used as I2C, it is necessary to disable I2C and add flow control pins if users apply the flow control of UART1.

4.2. Applying Hardware Flow Control of UART1

Please refer to the following configuration:

A. Add RTS and CTS pins.

```
- [Mdm9607-pinctrl.dtsi (arch\...\qcom)]
Options View Window Help
                    pins = "gpio8", "gpio9","gpio10", "gpio11";
                    drive-strength = <2>;
                    bias-pull-down;
                };
            };

            blsp1_uart2_sleep: blsp1_uart2_sleep {
                mux {
                    pins = "gpio4", "gpio5","gpio6", "gpio7";
                    function = "gpio";
                };
                config {
                    pins = "gpio4", "gpio5","gpio6", "gpio7";
                    drive-strength = <2>;
                    bias-pull-down;
                    output-low;
                };
            };

            blsp1_uart2_active: blsp1_uart2_active {
                mux {
                    pins = "gpio4", "gpio5","gpio6", "gpio7";
                    function = "blsp_uart2";
                };
                config {
                    pins = "gpio4", "gpio5","gpio6", "gpio7";
                    drive-strength = <2>;
                    bias-disable;
                };
            };
```

B. Disable I2C

[Mdm9607-mtp.dtsi (arch\...\qcom)]

Options View Window Help

```
00025:          qca,bt-vdd-xtal-voltage-level = <1800000 1800000>;
00026:      };
00027: };
00028:
00029: &soc {
00030: };
00031:
00032: &blsp1_uart5 {
00033:     status = "ok";
00034:     pinctrl-names ="sleep", "default";
00035:     pinctrl-0 = <&uart_console_sleep>;
00036:     pinctrl-1 = <&uart_console_active>;
00037: };
00038:
00039: //2016-01-21, modify by jun.wu, change i2c-4 to i2c-2
00040: &i2c_2 {
00041:     status = "disable";
00042: };
00043:
00044: &i2c_4 {
00045:     status = "ok";
00046: };
00047:
00048: //2016-01-19, comment out by jun.wu, remove UART3 && spi_1 from device tree
00049:
00050: &spi_6 {
00051:     status = "ok";
```

5 Increasing Flow Control Pins for Debug UART (Optional, Supported by Individual Hardware)

For individual users' hardware, it is only valid when debug UART connects RTS and CTS configuration externally. Please refer to the previous chapter for the configuring process. The flow control pins of debug UART are not multiplexed, so it is unnecessary to disable other peripheral devices. The configured pins are shown below:

```
- [Mdm9607-pinctrl.dtsi (arch\...\qcom)]
Options  View  Window  Help
00051:                 };
00052:                 config {
00053:                     pins = "gpio42";
00054:                         drive-strength = <2>;
00055:                         bias-disable;
00056:                         output-high;
00057:                 };
00058:             };
00059:
00060:             uart_console_sleep: uart_console_sleep {
00061:                 mux {
00062:                     pins = "gpio8", "gpio9","gpio10", "gpio11";
00063:                     function = "gpio";
00064:                 };
00065:                 config {
00066:                     pins = "gpio8", "gpio9","gpio10", "gpio11";
00067:                     drive-strength = <2>;
00068:                     bias-pull-down;
00069:                     output-low;
00070:                 };
00071:             };
00072:
00073:             uart_console_active: uart_console_active {
00074:                 mux {
00075:                     pins = "gpio8", "gpio9","gpio10", "gpio11";
00076:                     function = "blsp_uart5";
00077:                 };
00078:                 config {
00079:                     pins = "gpio8", "gpio9","gpio10", "gpio11";
00080:                     drive-strength = <2>;
00081:                     bias-pull-down;
00082:                 };
00083:             };
00084:
```

6 Configuring UART1 as the Console

6.1. Correspondence of UART1 on Linux Device

To take UART1 as the console, please first understand the corresponding device of UART1 on Linux system. For example, the debug UART on Linux is /dev/ttyHSL0.

Open the file mdm9607-mtp.dtsi under the directory *arch/arm/boot/dts/qcom/.* The UART configured to ok are blsp1_uart5 (debug UART), blsp1_uart3 (High-speed UART) and blsp1_uart2 (UART1). High-speed UART is not ttyHSL* device on Linux system, so the device files on Linux system corresponding to debug UART and UART1 are /dev/ttyHSL0 and /dev/ttyHSL1 respectively according to the sequence of their appearance

```
&blsp1_uart5 {
    status = "ok";
    pinctrl-names ="sleep", "default";
    pinctrl-0 = <&uart_console_sleep>;
    pinctrl-1 = <&uart_console_active>;
};

//2016-01-21, modify by jun.wu, change i2c-4 to i2c-2
&i2c_2 {
    status = "ok";
};

&i2c_4 {
    status = "ok";
};

//2016-01-19, comment out by jun.wu, remove UART3 && spi_1 from device tree

&spi_6 {
    status = "ok";
};

&blsp1_uart3 {
    status = "ok";
};

&blsp1_uart2 {
    status = "ok";  //if need, user can enable by themselves
    pinctrl-names ="sleep", "default";
    pinctrl-0 = <&blsp1_uart2_sleep>;
    pinctrl-1 = <&blsp1_uart2_active>;
};
```

6.2. Configuring UART1 as Linux Console

Linux console specifies in file /etc/inittab of rootfs. Open file inittab:

```
$ vim ql-ol-rootfs/etc/inittab
```

Restore the 48 lines commented out in ***Chapter2.3*** and change ttyHSL0 to ttyHSL1.

```
 45 m2:5:respawn:/usr/bin/mbimd
 46 m3:5:once:/sbin/usb/compositions/quec_mbim_check
 47
 48 S:2345:respawn:/sbin/getty -L ttyHSL1 115200 console
-- Insert --
```

Recompile rootfs and download it.

```
$ make rootfs
```

7 Verifying Linux Console

After modifying it according to ***Chapter 2-6***, compiling and downloading the kernel and rootfs, reboot the Linux system. It can be found that only the log that SBL outputs disappear in the original debug UART.

```
B -    370758 - sbl1_ddr_set_params, Start
B -    374509 - Pre_DDR_clock_init, Start
D -       213 - Pre_DDR_clock_init, Delta
D -         0 - sbl1_ddr_set_params, Delta
B -    387289 - pm_driver_init, Start
D -      4483 - pm_driver_init, Delta
B -    393602 - cpr_init, Start
D -        91 - cpr_init, Delta
B -    398147 - cpr_cx_mx_apc_vol_update, Start
D -        91 - cpr_cx_mx_apc_vol_update, Delta
B -    412726 - sbl1_qhsusb_al_do_fast_enum, Start
D -         0 - sbl1_qhsusb_al_do_fast_enum, Delta
B -    415989 - clock_init, Start
D -       152 - clock_init, Delta
B -    421784 - boot_flash_init, Start
D -     37362 - boot_flash_init, Delta
B -    463234 - Image Load, Start
D -     61244 - QSEE Image Loaded, Delta - (490652 Bytes)
B -    525545 - sbl1_efs_handle_cookies, Start
D -         0 - sbl1_efs_handle_cookies, Delta
B -    531584 - Devcfg Partition does not exist
B -    535854 - Image Load, Start
D -        30 - SEC Image Loaded, Delta - (0 Bytes)
B -    543540 - Image Load, Start
D -     26871 - RPM Image Loaded, Delta - (152464 Bytes)
B -    570472 - Image Load, Start
D -     39558 - APPSBL Image Loaded, Delta - (373532 Bytes)
B -    610091 - QSEE Execution, Start
D -       183 - QSEE Execution, Delta
B -    615856 - SBL1, End
D -    511577 - SBL1, Delta
S - Throughput, 3000 KB/s  (1017068 Bytes,  285935 us)
S - DDR Frequency, 240 MHz
```

After Linux boot is completed, open UART1 on the computer, then you can log in the Linux console.

```
msm 201804281307 mdm9607-perf /dev/ttyHSL1

mdm9607-perf login: root
Password:
root@mdm9607-perf:~# ls
root@mdm9607-perf:~#
```

8 Verifying Debug UART Function Test

8.1. Introduction and Compilation of Example

Here is an example of main UART, namely */dev/ttyHS0.*
In the example, the main thread writes data to the UART every second, and users can open the COM port of the host to receive data; at the same time, the child thread is monitoring RX if there is data coming in. Users send data to RX from the COM port of the host, the main UART will receive it and print it out.

```
#define QL_UART1_DEV "/dev/ttyHS0"

static int fd_uart = -1;
```

Enter the directory of *ql-ol-sdk/ql-ol-extsdk/example/uart*, and *make* generates executable program of *example_uart.* The premise of compiling is that the initialization of the cross-compilation environment has been completed.
Source *ql-ol-crosstool/ql-ol-crosstool-env-init*

```
gale@eve-linux02:~/MDM9x07/SDK_FAG0130/ql-ol-sdk/ql-ol-extsdk/example/uart$ make
arm-oe-linux-gnueabi-gcc  -march=armv7-a -mfloat-abi=softfp -mfpu=neon  -O2 -fexpensive-
de -I/home/gale/MDM9x07/SDK_FAG0130/ql-ol-sdk/ql-ol-crosstool/sysroots/armv7a-vfp-neon-o
eon-oe-linux-gnueabi/usr/include -I/home/gale/MDM9x07/SDK_FAG0130/ql-ol-sdk/ql-ol-crosst
-ol-crosstool/sysroots/armv7a-vfp-neon-oe-linux-gnueabi/usr/include/dsutils -I/home/gale
home/gale/MDM9x07/SDK_FAG0130/ql-ol-sdk/ql-ol-crosstool/sysroots/armv7a-vfp-neon-oe-linu
-ol-crosstool/sysroots/armv7a-vfp-neon-oe-linux-gnueabi/usr/include -I/home/gale/MDM9x07
M9x07/SDK_FAG0130/ql-ol-sdk/ql-ol-crosstool/sysroots/armv7a-vfp-neon-oe-linux-gnueabi/us
gnueabi/usr/include/dsutils -I/home/gale/MDM9x07/SDK_FAG0130/ql-ol-sdk/ql-ol-crosstool/s
osstool/sysroots/armv7a-vfp-neon-oe-linux-gnueabi/usr/include/qmi-framework  -L./ -L/hom
arm-oe-linux-gnueabi-gcc  -march=armv7-a -mfloat-abi=softfp -mfpu=neon -L./ -L/home/gale
9x07/SDK_FAG0130/ql-ol-sdk/ql-ol-extsdk/example/uart/../../lib -lrt -lpthread /home/gale
gale@eve-linux02:~/MDM9x07/SDK_FAG0130/ql-ol-sdk/ql-ol-extsdk/example/uart$
gale@eve-linux02:~/MDM9x07/SDK_FAG0130/ql-ol-sdk/ql-ol-extsdk/example/uart$
gale@eve-linux02:~/MDM9x07/SDK_FAG0130/ql-ol-sdk/ql-ol-extsdk/example/uart$ ls
example_uart  example_uart.c  example_uart.o  Makefile
```

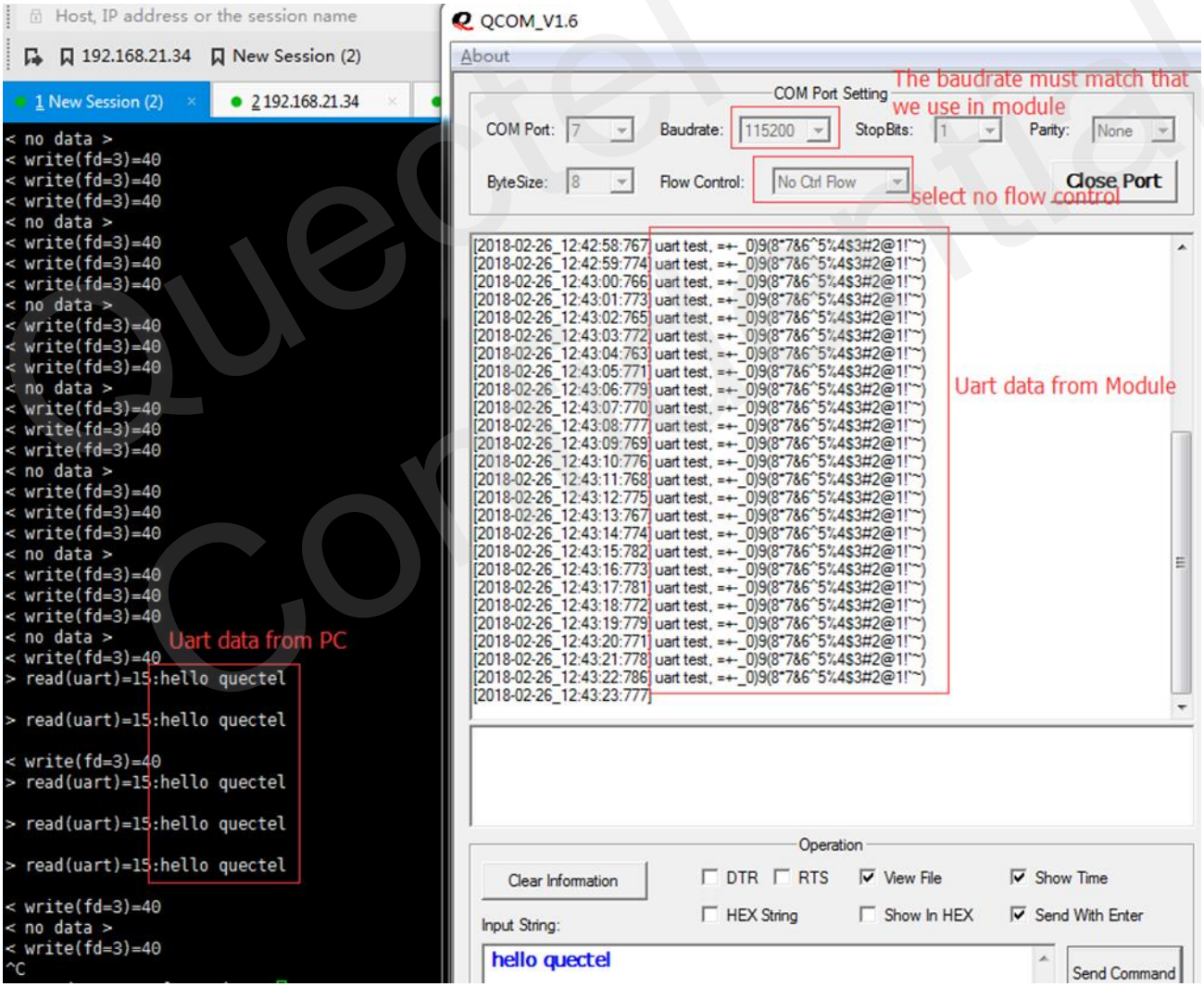
8.2. Function Test

8.2.1. Disabling Flow Control

```
fd_uart = Ql_UART_Open(QL_UART1_DEV, baudRate, FC_NONE);
printf("< open(\"%s\", %d)=%d\n", QL_UART1_DEV, baudRate, fd_uart);
```

1. Compile and upload the example_uart to the module via adb push<the path of example_uart in the host >, < internal path of the module, such as /usrdata>
   Or
   UART protocol, RZ.
2. Execute example_uart 115200, and open the corresponding COM port with corresponding baud rate and no flow control on the host as follows.

```
root@mdm9607-perf:/usrdata# ./example_uart 115200
< OpenLinux: UART example >
< open("/dev/ttyHS0", 115200)=3
```



8.2.2. Enabling Hardware Flow Control

Premise: To enable hardware flow control, it needs to connect RTS and CTS of the debug UART in the customers' EVB board externally. OPEN_EVB cannot be tested.

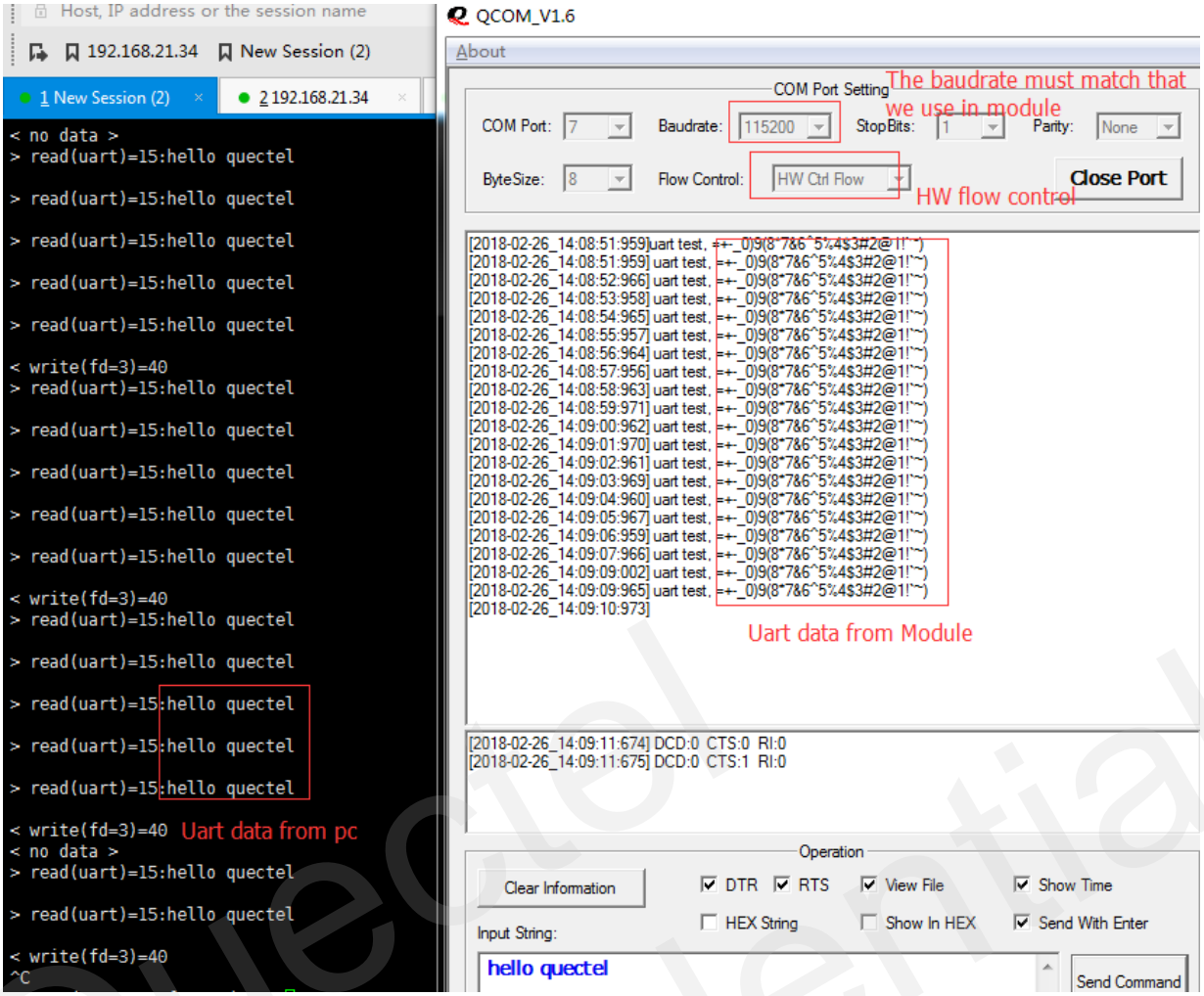
Modify example to enable hardware flow control.

```
fd_uart = Ql_UART_Open(QL_UART1_DEV, baudRate, FC_RTSCTS);
printf("< open(\"%s\", %d)=%d\n", QL_UART1_DEV, baudRate, fd_uart);
```

1. Compile and upload example_uart to the module via adb push<the path of example_uart in the host>, < internal path of the module, such as /usrdata>
   Or
   UART protocol, RZ.

2. Execute example_uart 115200, and open the corresponding COM port with corresponding baud rate and hardware flow control on the host as follows.

```
root@mdm9607-perf:/usrdata# ./example_uart 115200
< OpenLinux: UART example >
< open("/dev/ttyHS0", 115200)=3
```



8.2.3. Enabling Software Flow Control

8.2.3.1. Description of Software Flow Control XON/XOFF Character

XOFF/XON representations in ASCII

Code	Meaning	ASCII	Dec	Hex	Keyboard
XOFF	Pause transmission	DC3	19	13	Ctrl + S
XON	Resume transmission	DC1	17	11	Ctrl + Q

8.2.3.2. Software Flow Control Test

Modify example to enable hardware flow control.

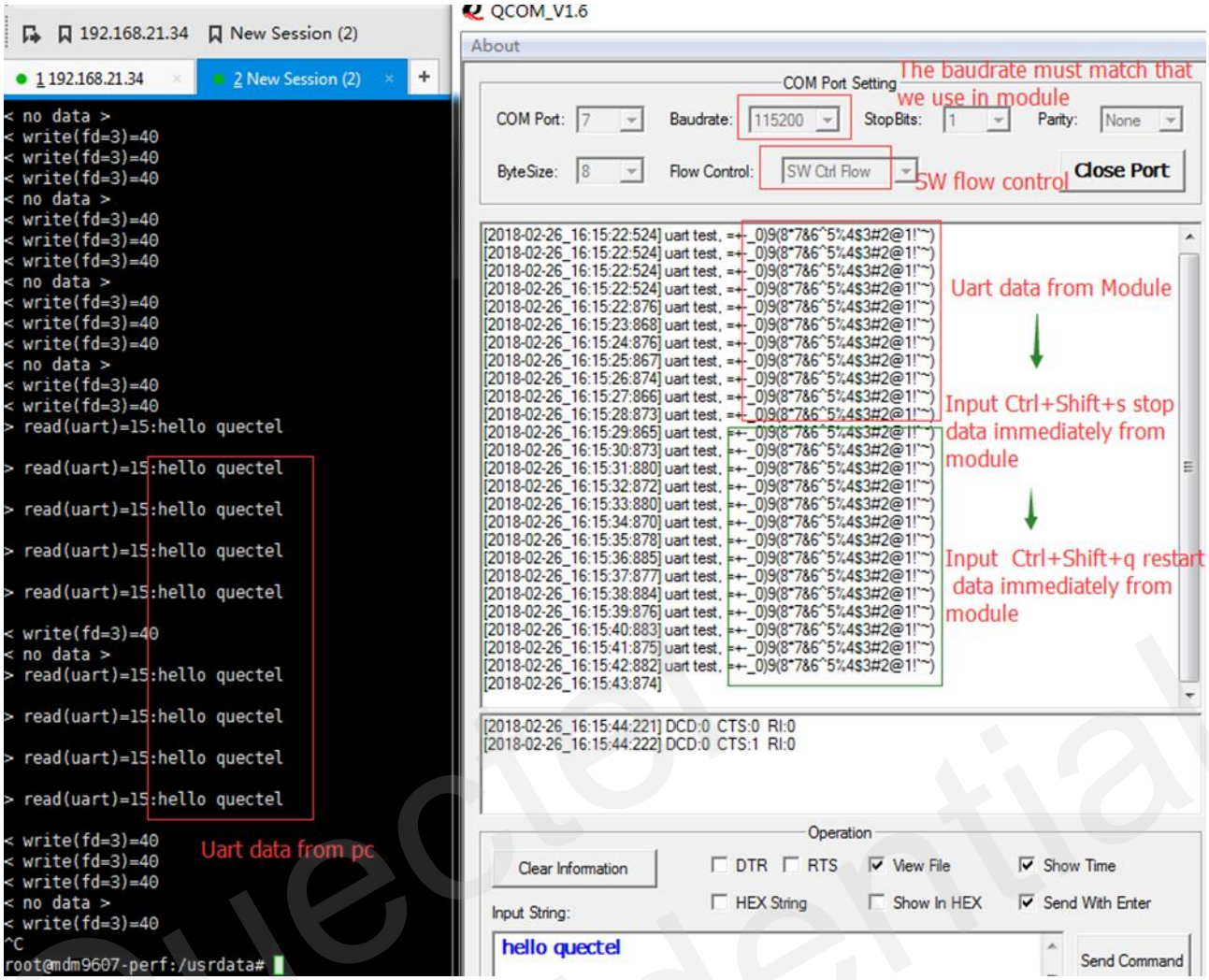
```
fd_uart = Ql_UART_Open(QL_UART1_DEV, baudRate, FC_XONXOFF);
printf("< open(\"%s\", %d)=%d\n", QL_UART1_DEV, baudRate, fd_uart);
```

1. Compile and upload example_uart to the module via adb push<the path of example_uart in the host>, < internal path of the module, such as /usrdata>
   Or
   UART protocol, RZ.

2. Execute example_uart 115200, and open the corresponding COM port with corresponding baud rate and hardware flow control on the host as follows.

```
root@mdm9607-perf:/usrdata# ./example_uart 115200
< OpenLinux: UART example >
< open("/dev/ttyHS0", 115200)=3
```

Type Ctrl+Shift+S on the keyboard in the UART software of the host, or send hexadecimal 0x13 when enabling software flow control to transfer data, the module side will stop data transmission immediately. Resume data transmission by Ctrl+Shift+Q or sending 0x11 to verify that the software flow control is normal.



9 Appendix A References

Table 1: Terms and Abbreviations

Abbreviation	Description
UART	Universal Asynchronous Receiver/Transmitter
LTE	Long Term Evolution
ADB	Android Debug Bridge
GPIO	General-purpose Input/Output
RTS	Request to Send
CTS	Clear to Send
COM	Component Object Model
RZ	Receive Z-Modem
RX	Receive
ASCII	American Standard Code for Information Interchange