AG35-QuecOpen

UART Development Guide

Automotive Module Series

Rev. AG35-Quecopen_UART_ Development_Guide_V1.0

Date: 2019-03-04

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	2018-02-28	Gale GAO	Initial
1.1	2019-03-02	Larry ZHANG	Added UART list of AG35 project and DTSI file configuration Added UART instructions

Contents

Table Index

Figure Index

1 Introduction

This document mainly introduces hardware, software driver layer, software application layer from the perspective of users to help them develop easily and quickly.

This document mainly applies for global market. The automotive module currently supporting this interface includes:

- AG35

2 AG35-QuecOpen UART Description

Six UART that are provided by AG35-QuecOpen module are: debug UART, UART1, UART2, UART3, UART4 and UART5. The UART1 has the same function with UART2, UART3 and UART4, and the four UART that can be used as peripheral communication UART support RTS/CTS on hardware. Among them, UART1 is multiplexed with SPI, UART2 multiplexed with SPI, UART4 and UART5 multiplexed with SDC1. For more details about pin multiplexing, please refer to ***Chapter 4***. The main features of the four UART are described as follows:

- The baud rate supported by debug UART that is used for Linux control and log output is 115200bps.
- The baud rates supported by UART1 are: 4800bps, 9600bps, 19200bps, 38400bps, 57600bps, 115200bps, 230400bps, 460800bps and 921600bps. The default baud rate is 115200bps. UART1 that can be used as common peripheral communication UART supports RTS/CTS flow control.
- The baud rates supported by UART2 are: 4800bps, 9600bps, 19200bps, 38400bps, 57600bps, 115200bps, 230400bps, 460800bps and 921600bps. The default baud rate is 115200bps. UART2 that can be used as common peripheral communication UART supports RTS/CTS flow control.
- The baud rates supported by UART3 are: 4800bps, 9600bps, 19200bps, 38400bps, 57600bps, 115200bps, 230400bps, 460800bps and 921600bps. The default baud rate is 115200bps. UART3 that can be used as common peripheral communication UART supports RTS/CTS flow control.
- The baud rates supported by UART4 are: 4800bps, 9600bps, 19200bps, 38400bps, 57600bps, 115200bps, 230400bps, 460800bps and 921600bps. The default baud rate is 15200bps. UART4 that can be used as common peripheral communication UART supports RTS/CTS flow control.
- The baud rates supported by UART5 are: 4800bps, 9600bps, 19200bps, 38400bps, 57600bps and 115200bps. UART5 does not support RTS and CTS flow control.

2.1. UART Usage Instruction

UART can be divided into High-speed UART and Low-speed UART. The main differences between the two modes are: High-speed UART transmission takes the form of BAM pipe; Low-speed UART transmission takes the form of FIFO. Device node generated by the two modes are also different: the Low-speed UART device name is */dev/ttyHSLx*; the High-speed UART device name is */dev/ttyHSx*. There are no differences on hardware connection, but to make sure the Integrity of receiving and sending pin signal.

Moreover, here are some recommendations of the selection of High-speed UART and Low-speed UART for users. If UART's baud rate is over 115200bps or involves a large data packets, it is recommended to use High-speed UART and hardware flow control transmission such as Bluetooth, etc. If UART's baud rate is less than or equal to 115200bps or involves a small data packets, Low-speed UART can be adopted such as debug UART, etc.

3 Recommendations for Hardware Circuit Design

Note: CTS and RTS on 4G module of QuecOpen have been reversed, so user should connect the CTS of the MCU to the CTS on 4G module, the RTS of the MCU to the RTS on 4G module when connecting.

3.1. UART Design Circuit with Level Shifting

3.1.1. TXS0108EPWR Level Shifting Chip made by TI Company Being Recommended

The UART level of AG35-QuecOpen module is 1.8V. If the user's host system level is 3.3V, it is necessary to add Level shifter when connecting module to host UART. It is recommended to use the TXS0108EPWR Level Shifting Chip made by TI Company. The following figure shows the reference circuit design with level shifting chip.

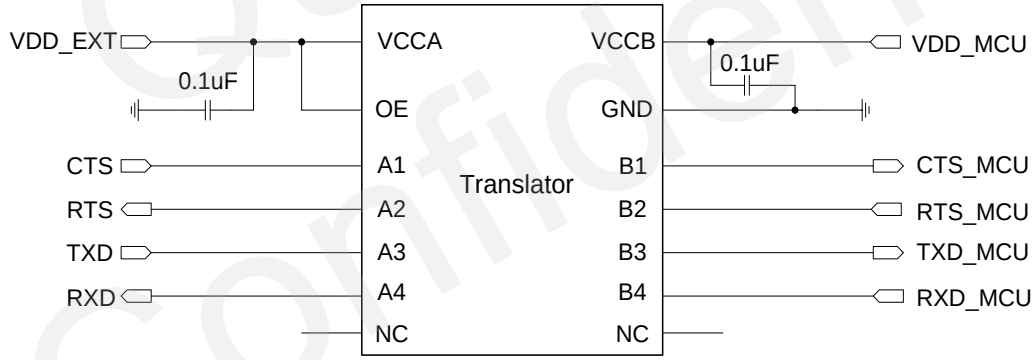


Figure 1: TXS0108EPWR Level Shifting Chip Circuit

For more details, please visit http://www.ti.com.

3.1.2. Other Level Shifting Circuits: Reference Circuit of Triode Level Shifting

Another level shifting circuit is shown in the following figure. The input and output circuit design in the following dotted line can refer to the part of solid line, but please pay attention to the connection direction. The triode level shifting circuit in the following figure is not applicable to applications with a baud rate exceeding 460Kbps.

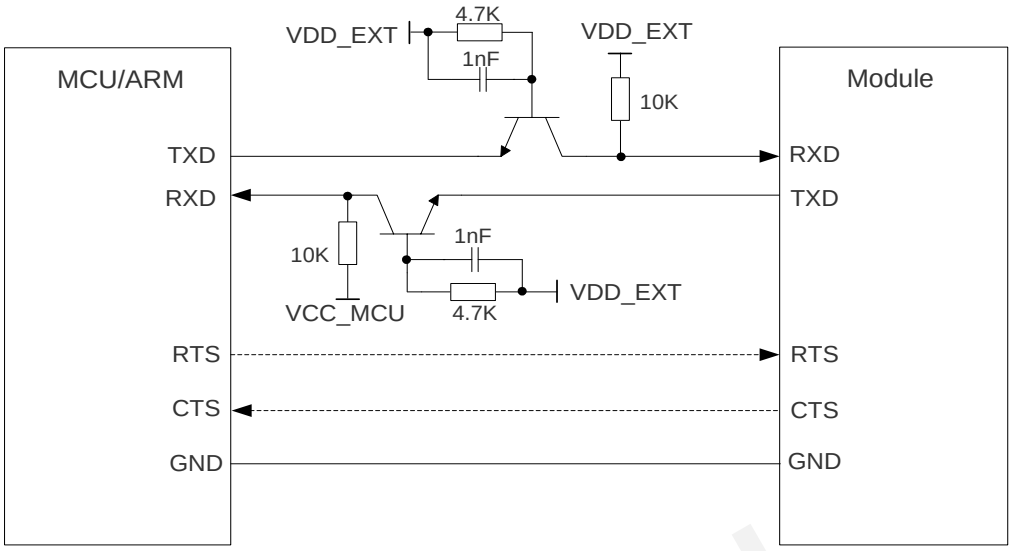


Figure 2: Triode Level Shifting Reference Circuit

3.2. Reference Circuit of Non-Level Shifting Chip

Module's UART level is 1.8V, if the user's host system level is also 1.8V, please connect directly:

Figure 3: 1.8V Direct Connection Circuit

4 Software Adaptation for Driver Layer and Device Tree

4.1. UART Pin Usage

4.1.1. NOTICES

- Non-default multiplexing functions are valid with the condition of software being configured. For the software configuration, please refer to the corresponding function chapter.
- In AG35-QuecOpen, the main UART is only used as common peripheral communication UART and no longer has the AT function.
- For the specific pin usage, please refer to *Quectel_AG35 _QuecOpen_GPIO_Assignment_Spe adsheet.*

4.1.2. Pin Definition of UART1

Table 1: Pin Definition of AG35 UART1 (Software DTSI Configuration Using blsp1_UART3 Fields)

Pin Name	Pin Number	I/O	Functional Description		
			Multiplexing Function 1 (Default)	Multiplexing Function 2	Multiplexing Function 3
UART1_CTS	56	DO	UART_CTS_BLSP3	GPIO_3	SPI_CLK_BLSP3
UART1_RTS	57	DI	UART_RTS_BLSP3	GPIO_2	SPI_CS_N_BLSP3
UART1_TXD	60	DO	UART_TXD_BLSP3	GPIO_0	SPI_MOSI_BLSP3
UART1_RXD	58	DI	UART_RXD_BLSP3	GPIO_1	SPI_MISO_BLSP3

4.1.3. Pin Definition of Debug UART

Table 2: Pin Definition of AG35 Debug UART (Software DTSI Configuration Using blsp1_UART2 Fields)

Pin Name	Pin Number	I/O	Description	Remarks
DBG_TXD	71	DO	Modules send data	1.8V Power Domain
DBG_RXD	72	DI	Modules receive data	1.8V Power Domain

4.1.4. Pin Definition of UART2 (RTS/CTS Multiplexing with SPI)

Table 3: Pin Definition of AG35 UART2 (Software DTSI Configuration Using blsp1_UART5 Fields)

Pin Name	Pin Number	I/O	Functional Description		
			Multiplexing Function 1 (Default)	Multiplexing Function 2	Multiplexing Function 3
UART2_CTS	164	DO	UART_CTS_BLSP5	GPIO_11	SPI_CLK_BLSP5
UART2_RTS	166	DI	UART_RTS_BLSP5	GPIO_10	SPI_CS_N_BLSP5
UART2_TXD	163	DO	UART_TXD_BLSP5	GPIO_8	SPI_MOSI_BLSP5
UART2_RXD	165	DI	UART_RXD_BLSP5	GPIO_9	SPI_MISO_BLSP5

4.1.5. Pin Definition of UART3 (Multiplexing with SPI)

Table 4: Pin Definition of AG35 UART3 (Software DTSI Configuration Using blsp1_UART6 Fields)

Pin Name	Pin Number	I/O	Functional Description			
			Multiplexing Function 1 (Default)	Multiplexing Function 2	Multiplexing Function 3	Multiplexing Function 4
SPI_CS_N	79	DO	SPI_CS_N_BLSP6	GPIO_22	UART_RTS_BLSP6	PCM_1_DOUT
SPI_MOSI	77	DO	SPI_MOSI_BLSP6	GPIO_20	UART_TXD_BLSP6	PCM_1_SYNC
SPI_MISO	78	DI	SPI_MISO_BLSP6	GPIO_21	UART_RXD_BLSP6	PCM_1_DIN
SPI_CLK	80	DO	SPI_CLK_BLSP6	GPIO_23	UART_CTS_BLSP6	PCM_1_CLK

4.1.6. Pin Definition of UART4 (Multiplexing with SDC1)

Table 5: Pin Definition of AG35 UART4 (Software DTSI Configuration Using blsp1_UART1 Fields)

Pin Name	Pin Number	I/O	Functional Description		
			Multiplexing Function 1 (Default)	Multiplexing Function 2	Multiplexing Function 3
SDC1_DATA0	20	IO	SDC1_DATA0	GPIO_15	UART_CTS_BLSP1
SDC1_DATA1	21	IO	SDC1_DATA1	GPIO_14	UART_RTS_BLSP1
SDC1_DATA2	22	IO	SDC1_DATA2	GPIO_13	UART_RXD_BLSP1
SDC1_DATA3	23	IO	SDC1_DATA3	GPIO_12	UART_TXD_BLSP1

4.1.7. Pin Definition of UART5 (Multiplexing with SDC1)

Table 6: Pin Definition of AG35 UART5 (Software DTSI Configuration Using blsp1_UART4 Fields)

Pin Name	Pin Number	I/O	Functional Description		
			Multiplexing Function 1 (Default)	Multiplexing Function 2	Multiplexing Function 3
SDC1_CMD	18	IO	SDC1_CMD	GPIO_17	UART_RXD_BLSP4
SDC1_CLK	19	DO	SDC1_CLK	GPIO_16	UART_TXD_BLSP4

4.1.8. UART Logic Level

Table 7: UART Logic Level

Parameter	Minimum	Maximum	Unit
V_{IL}	-0.3	0.6	V
V_{IH}	1.2	2.0	V
V_{OL}	0	0.45	V
V_{OH}	1.35	1.8	V

4.2. UART Device Tree Configuration Method

4.2.1. Configuration Instructions

Quectel has completed a series of UART configuration such as compatible drivers, pin selection, register address, UART interrupt number, CLK, and system sleeping and working configuration, etc., so users just need to simply open or close as follows.

4.2.2. UART1 Configuration

UART1 supports hardware flow control by default and is shown as the */dev/ttyHS0* device node inside the module.

```
root@mdm9607-perf:~# ls /dev/tty*
/dev/tty      /dev/tty19    /dev/tty3     /dev/tty40    /dev/tty51    /dev/tty62
/dev/tty0     /dev/tty2     /dev/tty30    /dev/tty41    /dev/tty52    /dev/tty63
/dev/tty1     /dev/tty20    /dev/tty31    /dev/tty42    /dev/tty53    /dev/tty7
/dev/tty10    /dev/tty21    /dev/tty32    /dev/tty43    /dev/tty54    /dev/tty8
/dev/tty11    /dev/tty22    /dev/tty33    /dev/tty44    /dev/tty55    /dev/tty9
/dev/tty12    /dev/tty23    /dev/tty34    /dev/tty45    /dev/tty56    /dev/ttyGS0
/dev/tty13    /dev/tty24    /dev/tty35    /dev/tty46    /dev/tty57    /dev/ttyHS0
/dev/tty14    /dev/tty25    /dev/tty36    /dev/tty47    /dev/tty58    /dev/ttyHSL0
/dev/tty15    /dev/tty26    /dev/tty37    /dev/tty48    /dev/tty59
/dev/tty16    /dev/tty27    /dev/tty38    /dev/tty49    /dev/tty6
/dev/tty17    /dev/tty28    /dev/tty39    /dev/tty5     /dev/tty60
/dev/tty18    /dev/tty29    /dev/tty4     /dev/tty50    /dev/tty61
```

Generally Speaking, it is recommended to treat the 4 pins that is used by UART1 as UART (by default) or modify the 4 pins as GPIO. Here is how to modify UART1 as GPIO.

```
--- a/ql-ol-kernel/arch/arm/boot/dts/qcom/mdm9607-mtp.dtsi
+++ b/ql-ol-kernel/arch/arm/boot/dts/qcom/mdm9607-mtp.dtsi
@@ -52,7 +52,7 @@
 };

 &blsp1_uart3 {
-       status = "ok";
+       status = "disabled";
 };
```

Compile the kernel and download it. For more details, please refer to the compilation method in *KBA_QuecOpen_EC2X&AG35_Quick_Start*.

Download new firmware, and if /dev/ttyHS0 disappeared as the following figure, which means UART1 has been closed and the 4 pins can be used as GPIO.

```
root@mdm9607-perf:~# ls /dev/tty*
/dev/tty      /dev/tty19    /dev/tty3     /dev/tty40    /dev/tty51    /dev/tty62
/dev/tty0     /dev/tty2     /dev/tty30    /dev/tty41    /dev/tty52    /dev/tty63
/dev/tty1     /dev/tty20    /dev/tty31    /dev/tty42    /dev/tty53    /dev/tty7
/dev/tty10    /dev/tty21    /dev/tty32    /dev/tty43    /dev/tty54    /dev/tty8
/dev/tty11    /dev/tty22    /dev/tty33    /dev/tty44    /dev/tty55    /dev/tty9
/dev/tty12    /dev/tty23    /dev/tty34    /dev/tty45    /dev/tty56    /dev/ttyGS0
/dev/tty13    /dev/tty24    /dev/tty35    /dev/tty46    /dev/tty57    /dev/ttyHSL0
/dev/tty14    /dev/tty25    /dev/tty36    /dev/tty47    /dev/tty58
/dev/tty15    /dev/tty26    /dev/tty37    /dev/tty48    /dev/tty59
/dev/tty16    /dev/tty27    /dev/tty38    /dev/tty49    /dev/tty6
/dev/tty17    /dev/tty28    /dev/tty39    /dev/tty5     /dev/tty60
/dev/tty18    /dev/tty29    /dev/tty4     /dev/tty50    /dev/tty61
```

Pay attention to whether the following configuration is disabled if users treat UART1 as common GPIO:

```
--- a/mdm9607-mtp.dtsi
+++ b/mdm9607-mtp.dtsi
@@ -101,7 +101,7 @@

 /* Add spi3 node ---> gpio0,1,2,3, gale 2018-3-14 */
 &spi_3 {
-        status = "disabled";
+        status = "disabled";
 };
```

4.2.3. Debug UART Configuration

Debug UART is used for Linux debugging and log output of the module by default, it is not recommended to make any changes.

Debug UART does not enable hardware flow control and is shown as the */dev/ttyHSL0* device node inside the module.

```
root@mdm9607-perf:~# ls /dev/tty*
/dev/tty      /dev/tty19    /dev/tty3     /dev/tty40    /dev/tty51    /dev/tty62
/dev/tty0     /dev/tty2     /dev/tty30    /dev/tty41    /dev/tty52    /dev/tty63
/dev/tty1     /dev/tty20    /dev/tty31    /dev/tty42    /dev/tty53    /dev/tty7
/dev/tty10    /dev/tty21    /dev/tty32    /dev/tty43    /dev/tty54    /dev/tty8
/dev/tty11    /dev/tty22    /dev/tty33    /dev/tty44    /dev/tty55    /dev/tty9
/dev/tty12    /dev/tty23    /dev/tty34    /dev/tty45    /dev/tty56    /dev/ttyGS0
/dev/tty13    /dev/tty24    /dev/tty35    /dev/tty46    /dev/tty57    /dev/ttyHS0
/dev/tty14    /dev/tty25    /dev/tty36    /dev/tty47    /dev/tty58    /dev/ttyHSL0
/dev/tty15    /dev/tty26    /dev/tty37    /dev/tty48    /dev/tty59
/dev/tty16    /dev/tty27    /dev/tty38    /dev/tty49    /dev/tty6
/dev/tty17    /dev/tty28    /dev/tty39    /dev/tty5     /dev/tty60
/dev/tty18    /dev/tty29    /dev/tty4     /dev/tty50    /dev/tty61
```

The corresponding debug UART in device tree is:

File path: *ql-ol-kernel/arch/arm/boot/dts/qcom/mdm9607-mtp.dtsi*

```
&blsp1_uart2 {
        status = "ok";
        pinctrl-names = "default","sleep";
        pinctrl-0 = <&blsp1_uart2_active>;
        pinctrl-1 = <&blsp1_uart2_sleep>;
};
```

4.2.4. UART2 Configuration

It is recommended to use UART2 and Bluetooth communication if the product definition has BT function and communicates by UART. The UART is open by default and the following devices will appear after it is opened:

```
~ # ls /dev/tty*
/dev/tty      /dev/tty19    /dev/tty3     /dev/tty40    /dev/tty51    /dev/tty62
/dev/tty0     /dev/tty2     /dev/tty30    /dev/tty41    /dev/tty52    /dev/tty63
/dev/tty1     /dev/tty20    /dev/tty31    /dev/tty42    /dev/tty53    /dev/tty7
/dev/tty10    /dev/tty21    /dev/tty32    /dev/tty43    /dev/tty54    /dev/tty8
/dev/tty11    /dev/tty22    /dev/tty33    /dev/tty44    /dev/tty55    /dev/tty9
/dev/tty12    /dev/tty23    /dev/tty34    /dev/tty45    /dev/tty56    /dev/ttyGS0
/dev/tty13    /dev/tty24    /dev/tty35    /dev/tty46    /dev/tty57    /dev/ttyHS0
/dev/tty14    /dev/tty25    /dev/tty36    /dev/tty47    /dev/tty58    /dev/ttyHS1
/dev/tty15    /dev/tty26    /dev/tty37    /dev/tty48    /dev/tty59    /dev/ttyHSL0
/dev/tty16    /dev/tty27    /dev/tty38    /dev/tty49    /dev/tty6
/dev/tty17    /dev/tty28    /dev/tty39    /dev/tty5     /dev/tty60
/dev/tty18    /dev/tty29    /dev/tty4     /dev/tty50    /dev/tty61
~ #
```

If there is no corresponding device above and needs to be opened, please check whether the following code is open:

```
--- a/mdm9607-mtp.dtsi
+++ b/mdm9607-mtp.dtsi
@@ -57,7 +57,7 @@

 /* larryzhang-20190225:The default is high speed for BLSP1_UART5(gpio8,9,10,11)*/
  &blsp1_uart5 {
-       status = "disabled";
+       status = "ok";
  };

 /* UART6: enable gpio20, 21 uart6 by gale */
@@ -106,7 +106,7 @@

 /* Add spi5 node ---> gpio8,9,10,11, gale 2018-3-14 */
 &spi_5 {
-        status = "disabled";
+        status = "disabled";
 };

 //add by cullen
```

Compile the kernel and download it. For more details, please refer to the compilation method in *KBA_QuecOpen_EC2X&AG35_Quick_Start*.
Download new firmware, if additional / dev / ttyHS1 device nodes are found, then UART2 can work.

4.2.5. UART3 Configuration

UART3 is not enabled by default and UART device node are not displayed inside the module. PIN77, PIN78, PIN79 and PIN80 are configured as SPI interfaces.

```
~ #
~ # ls /dev/tty*
/dev/tty      /dev/tty19    /dev/tty3     /dev/tty40    /dev/tty51    /dev/tty62
/dev/tty0     /dev/tty2     /dev/tty30    /dev/tty41    /dev/tty52    /dev/tty63
/dev/tty1     /dev/tty20    /dev/tty31    /dev/tty42    /dev/tty53    /dev/tty7
/dev/tty10    /dev/tty21    /dev/tty32    /dev/tty43    /dev/tty54    /dev/tty8
/dev/tty11    /dev/tty22    /dev/tty33    /dev/tty44    /dev/tty55    /dev/tty9
/dev/tty12    /dev/tty23    /dev/tty34    /dev/tty45    /dev/tty56    /dev/ttyGS0
/dev/tty13    /dev/tty24    /dev/tty35    /dev/tty46    /dev/tty57    /dev/ttyHS0
/dev/tty14    /dev/tty25    /dev/tty36    /dev/tty47    /dev/tty58    /dev/ttyHS1
/dev/tty15    /dev/tty26    /dev/tty37    /dev/tty48    /dev/tty59    /dev/ttyHSL0
/dev/tty16    /dev/tty27    /dev/tty38    /dev/tty49    /dev/tty6
/dev/tty17    /dev/tty28    /dev/tty39    /dev/tty5     /dev/tty60
/dev/tty18    /dev/tty29    /dev/tty4     /dev/tty50    /dev/tty61
~ #
```

If users need to add additional UART, open this UART as follows:

```
--- a/mdm9607-mtp.dtsi
+++ b/mdm9607-mtp.dtsi
@@ -62,7 +62,7 @@

 /* UART6: enable gpio20, 21 uart6 by gale */
 &blsp1_uart6 {
-        status = "disabled";
+        status = "ok";
         pinctrl-names = "default","sleep";
         pinctrl-0 = <&blsp1_uart6_active>;
         pinctrl-1 = <&blsp1_uart6_sleep>;
@@ -111,7 +111,7 @@

 //add by cullen
 &spi_6 {
-        status = "ok";
+        status = "disabled";
 };
```

Compile the kernel and download it. For more details, please refer to the compilation method in *KBA_QuecOpen_EC2X&AG35_Quick_Start*.

Download new firmware, if additional device node of */dev/ttyHSL1* is found in *ls /dev/ttyHS**, then UART3 can work.

In addition, the UART number in the module is determined by the sequence of multiple UART nodes in the device tree made by the UART driver. If the above debug UART is not enabled, the UART3 node name will turn into ttyHSL0.

Note: For general customers, the SDC1 interface is used on WIFI and is not used separately, so here do not introduce how to modify UART4 and UART5. For the special requirement, please contact Quectel FAE to get modification patch.

5 QuecOpen Application Layer API

5.1. User Programming Instructions

A complete user programming interface is provided in QuecOpen project SDK.
Reference path: *ql-ol-sdk/ql-ol-extsdk/*

```
gale@eve-linux02:~/MDM9x07/SDK_FAG0130/ql-ol-sdk/ql-ol-extsdk$ ls
docs  example  include  lib  target  tools
gale@eve-linux02:~/MDM9x07/SDK_FAG0130/ql-ol-sdk/ql-ol-extsdk$
```

API interface library provided by Quectel is included under the directory of *lib* in the above figure. The directory of *include* is the header file for all APIs. The directory of *example* provides function-based API usage example.

Here introduce UART related interfaces and reference examples.
UART application programming depends on library of *libql_peripheral.a.*
Head file: *ql_uart.h*.

5.2. UART API Introduction

Flow control enumeration: select flow control mode.

```
typedef enum {
    FC_NONE = 0,    // None Flow Control
    FC_RTSCTS,      // Hardware Flow Control (rtscts)
    FC_XONXOFF      // Software Flow Control (xon/xoff)
}Enum_FlowCtrl;
```

Parity bit enumeration: support none, odd parity and even parity.

```
typedef enum {
    PB_NONE = 0,    //none parity check
    PB_ODD,         //odd parity check
    PB_EVEN         //even parity check
}Enum_ParityBit;
```

Data bit enumeration: support 5, 6, 7 and 8.

```
typedef enum {
    DB_CS5 = 5,
    DB_CS6 = 6,
    DB_CS7 = 7,
    DB_CS8 = 8
}Enum_DataBit;
```

Stop bit enumeration: support 1bit and 2bit.

```
typedef enum {
    SB_1 = 1,
    SB_2 = 2
}Enum_StopBit;
```

Baud rate enumeration: baud rate supported by the module.

```
typedef enum {
    B_4800      = 4800,
    B_9600      = 9600,
    B_19200     = 19200,
    B_38400     = 38400,
    B_57600     = 57600,
    B_115200    = 115200,
    B_230400    = 230400,
    B_460800    = 460800,
    B_921600    = 921600,
}Enum_BaudRat;
```

UART properties structure:

```
typedef struct {
    Enum_BaudRate       baudrate;
    Enum_DataBit        databit;
    Enum_StopBit        stopbit;
    Enum_ParityBit      parity;
    Enum_FlowCtrl       flowctrl;
}ST_UARTDCB;
```

int Ql_UART_Open(const char port, Enum_BaudRate baudrate, Enum_FlowCtrl flowctrl);*

Open a UART device file with the specified baud rate and flow control method, and the default parity bit is PB_NONE, data bit DB_CS8, and stop bit SB_1. Use the interface of *int Ql_UART_SetDCB(int fd, ST_UARTDCB *dcb)* to modify these properties if neccessary.

Parameter: port: device name, such as */dev/ttyHS0*
baudrate: baud rate, refer to enumeration Enum_BaudRat, such as B_9600, B_115200
flowCtrl: flow control, enumeration Enum_FlowCtrl

Return value: return file descriptor, otherwise return -1.

*int Ql_UART_SetDCB(int fd, ST_UARTDCB *dcb);*
Set UART properties, including flow control, parity bit, data bit, stop bit and baud rate.
Parameter: fd: device file descriptor
dcb: Filled UART properties structure
Return value: return 0, returns -1 when errors happened.
*int Ql_UART_GetDCB(int fd, ST_UARTDCB *dcb);*
Obtain the current UART properties, including flow control, parity bit, data bit, stop bit, and baud rate.
Parameter: fd: device file descriptor
dcb: UART properties structure
Return value: return 0, returns -1 when errors happened.

int Ql_UART_Read(int fd, char buf, unsigned int buf_len);*
Read the content of specified byte length from the UART to buf; return the read byte length
Parameter: fd: device file descriptor
buf: read data pointer
buf_len: read the length
Return value: return the read byte length.

int Ql_UART_Write(int fd, const char buf, unsigned int buf_len);*
Write the specified byte length data from buf to UART; return the written byte length.
Parameter: fd: device file descriptor
buf: write data pointer
buf_len: write the length
Return value: return the written byte length.

int Ql_UART_Close(int fd);
Close device file descriptor.

Advanced UART programming:
The above interfaces can fully meet the needs of customers.
If the customer is an advanced Linux user and is very proficient in the characteristics of the UART, users can set the UART with the following interfaces.
int Ql_UART_IoCtl(int fd, unsigned int cmd, void pValue);*
Control UART device properties
Parameter: fd: device file descriptor
cmd: UART ioctl request, such as TCGETS, TCSETS, TIOCMGET, TIOCMSET, etc.
pValue: UART device properties pointer
Return value: return 0 if succeed, otherwise return -1.

Users can monitor the device file descriptor of uart_fd to implement asynchronous notification to read UART data.
Please refer to: *ql-ol-extsdk/example/uart.*

6 Verification for UART Function Test

6.1. 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.

```
3
4 #define QL_UART1_DEV  "/dev/ttyHS0"
5
6 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-
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
```

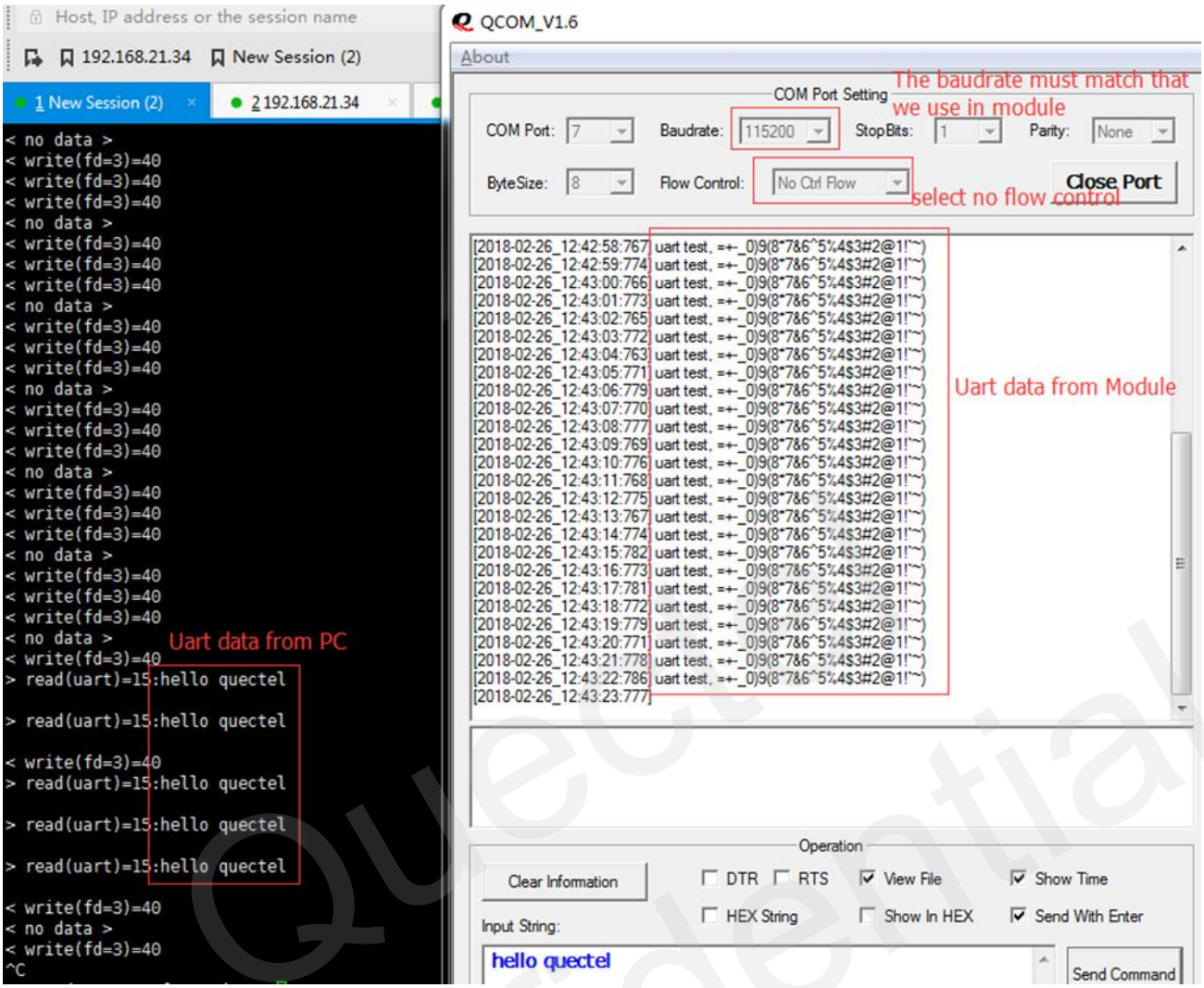
6.2. Function Test

6.2.1. 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. If users need to use OPEN_EVB, it is necessary to connect the J0202's main UART pin header with a jumper cap.
   GPIO_0 connects to MAIN_TXD;
   GPIO_1 connects to MAIN_RXD;
   Then connect to the hardware path.
3. 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
```



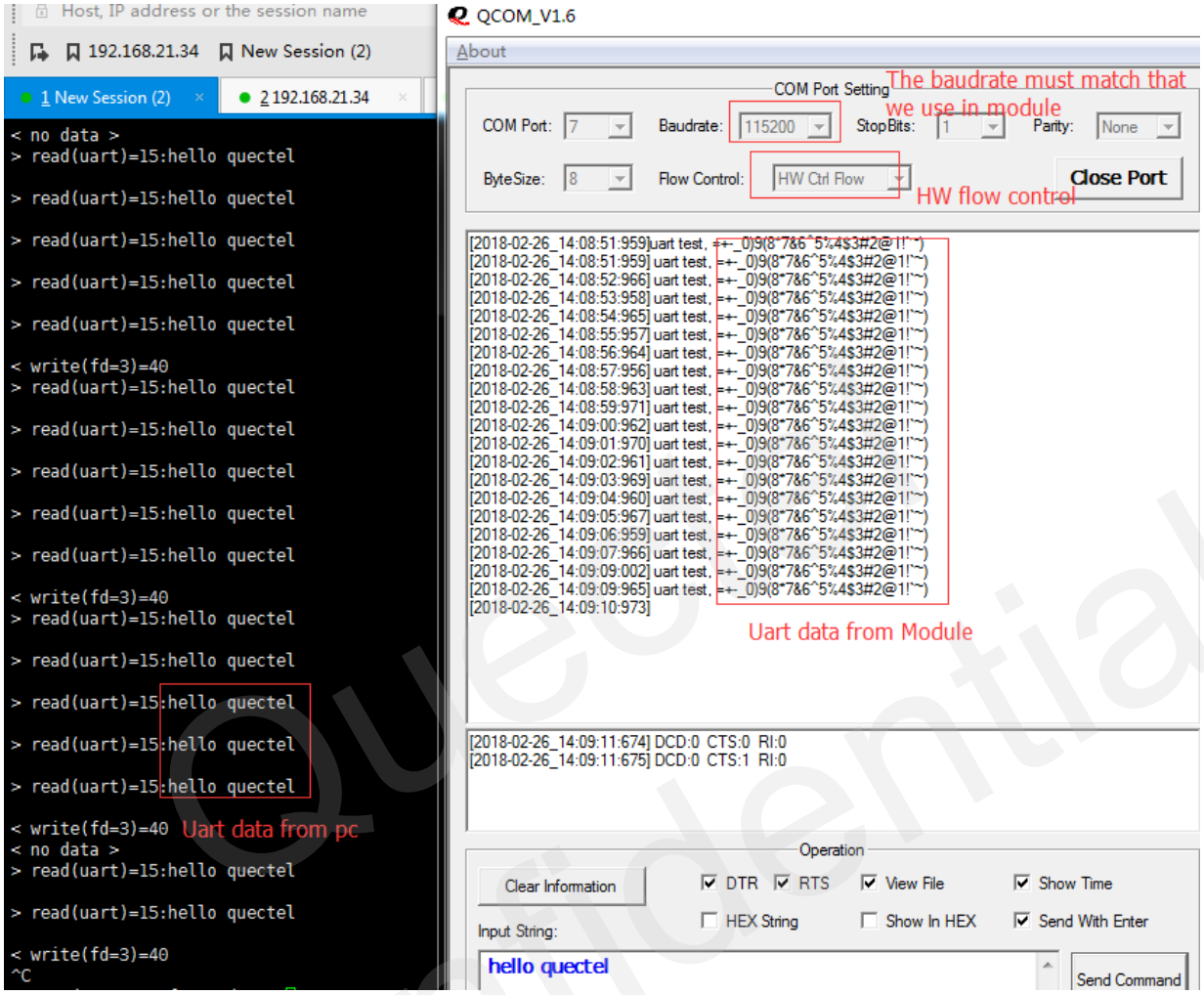
6.2.2. Enabling Hardware Flow Control

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. If users need to use OPEN_EVB, it is necessary to connect the J0202's main UART pin header with a jumper cap.
   GPIO_0 connects to MAIN_TXD;
   GPIO_1 connects to MAIN_RXD;
   GPIO_2 connects to MAIN_RTS;
   GPIO_3 connects to MAIN_CTS;
   Then connect to the hardware path.
3. 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
```



6.2.3. Enabling Software Flow Control

6.2.3.1. 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

6.2.3.2. 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. If users need to use OPEN_EVB, it is necessary to connect the J0202's main UART pin header with a jumper cap.
   GPIO_0 connects to MAIN_TXD;
   GPIO_1 connects to MAIN_RXD;
   Then connect to the hardware path.
3. 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.