AG35-QuecOpen

I2C Development Guide

Automotive Module Series

Rev. AG35-QuecOpen_I2C_Development_Guide_V1.0

Date: 2019-05-09

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	2019-05-09	Larry ZHANG	Added AG35 project

Contents

Figure Index

1 Introduction

This document introduces the hardware, software drive layer and software application layer from the perspective of user development, which would help customers develop easily and quickly.

This document mainly applies to global market. The Automotive Module Series module currently supporting this interface includes:

- AG35

2 Description of AG35-QuecOpen I2C

The module AG35-QuecOpen provides two I2C interfaces, and only the module can be served as master device among applications that related to I2C interfaces.

Clock mode supported: Standard (100 kHz), Fast (400 kHz). Default Clock is 400 kHz.

Support 7-bit device addressing, which means one I2C-BUS could mount up to 2^7-1=127 slave devices. For the I2C device with mass of data, one I2C-BUS to one device is recommended. Some of commonly used slave devices: Codec, Sensor and etc.

The maximum length of single transmission is 2^16–1 bit.

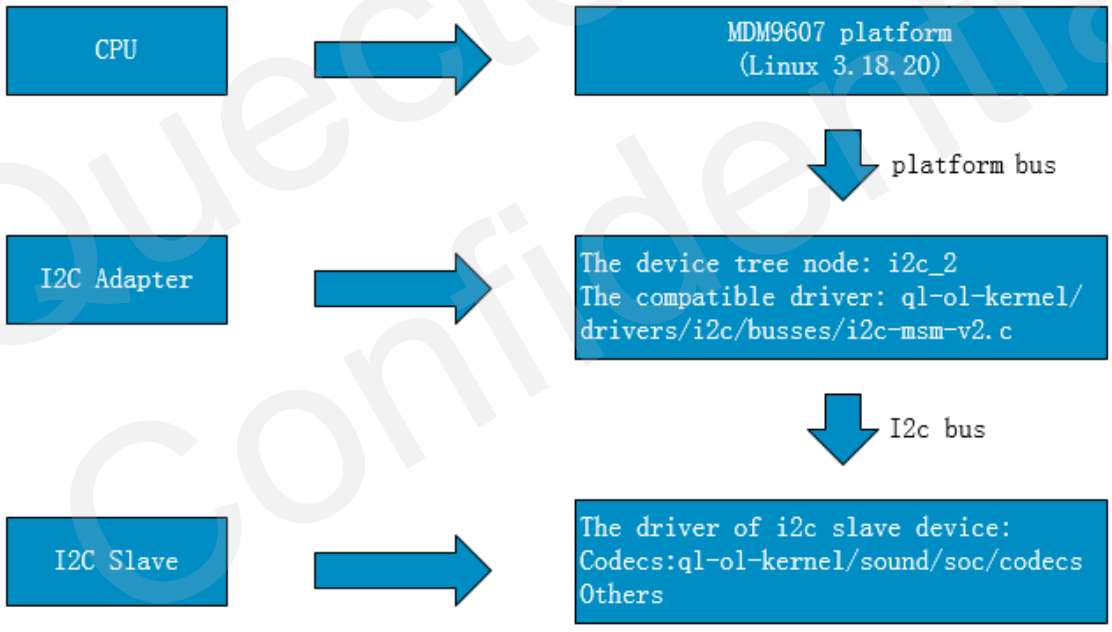

Figure 1: I2C Driver Architecture

3 Recommended Hardware Circuit Design

3.1. Referenced Design for PCM with External Codec Chip and I2C Interface

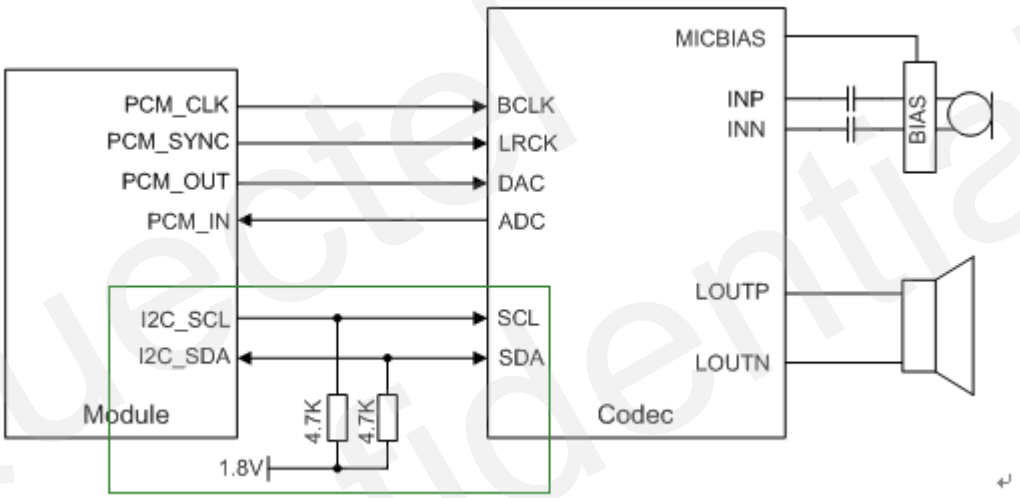

Figure 2: Circuit Diagram of PCM with External Codec Chip and I2C Interface

4 Description of Use of I2C Pin and Device Tree Configuration

4.1. Use of I2C Pin

In the Table 1, non-default multiplexing function will only take effect after the software has been configured. Please configure the software based on corresponding chapter.

I2C_SCL: Pulling up with external 1.8V is needed, or impend without use;
I2C_SDA: Pulling up with external 1.8V is needed, or impend without use;

I2C interface can be multiplexed as GPIO, default I2C.

For more details about the use of pin, please refer to: *Quectel_AG35_QuecOpen_GPIO_Assignment_Speadsheet*

Table 1：Definition of I2C Channel Pin

Pin Name	Pin No.	I/O	Function Description	
			Multiplexing Function 1 (Default)	Multiplexing Function 2
I2C_SCL	43	DO	I2C_SCL_BLSP4	GPIO_19
I2C_SDA	42	IO	I2C_SDA_BLSP4	GPIO_18
I2C2_SCL	74	DI	I2C_SCL_BLSP2	GPIO_7
I2C2_SDA	73	IO	I2C_SDA_BLSP2	GPIO_6

4.2. Methods to Configure I2C Device Tree

4.2.1. I2C Controller Configuration

I2C architecture under Linux consists 3 parts as following:

I2C-Core: It provides I2C-BUS driver and device driver registration and logout methods and I2C communication methods. I2C-Core also offers codes that have no relation to specific controller and upper layer codes of detective device and detecting device address.

I2C-BUS (Controller) Driver: It enables the control of I2C hardware. The controller is controlled by CPU, and it can also be integrated into CPU.

I2C Device Driver: That is client's I2C slave device driver, it enables the control of I2C hardware. The device usually been mounted on the I2C controller that controlled by CPU, and exchange data with CPU through I2C controller.

Among above 3 parts, customers only need to concern and modify the device driver.

I2C-BUS Driver: That is I2C controller. I2C-MSM-V2 controller is used on MDM9628 platform; QuecOpen has completed all hardware parameter configuration, such as the compatible driver, pin selection, register address, CLK, interrupt number, parameters of DMA Engine API, and pin configuration when the system is sleeping or working. There is no need of customer's concern and modifying.

I2C1 configuration as following (This BUS communicates with Codec by default):

```
        i2c_4: i2c@78b8000 { /* BLSP1 QUP4 */
                compatible = "qcom,i2c-msm-v2";
                #address-cells = <1>;
                #size-cells = <0>;
                reg-names = "qup_phys_addr";
                reg = <0x78b8000 0x600>;
                interrupt-names = "qup_irq";
                interrupts = <0 98 0>;
                qcom,clk-freq-out = <400000>;
                qcom,clk-freq-in  = <19200000>;
                clock-names = "iface_clk", "core_clk";
                clocks = <&clock_gcc clk_gcc_blsp1_ahb_clk>,
                        <&clock_gcc clk_gcc_blsp1_qup4_i2c_apps_clk>;

                pinctrl-names = "i2c_active", "i2c_sleep";
                pinctrl-0 = <&i2c_4_active>;
                pinctrl-1 = <&i2c_4_sleep>;
                qcom,noise-rjct-scl = <0>;
                qcom,noise-rjct-sda = <0>;
                qcom,master-id = <86>;
                dmas = <&dma_blsp1 18 64 0x20000020 0x20>,
                        <&dma_blsp1 19 32 0x20000020 0x20>;
                dma-names = "tx", "rx";
                status = "disabled";

                // achang-20180607, for update codec driver. (start)
                rt5616_codec@1b{
                        compatible = "realtek,rt5616";
                        reg = <0x1b>;
                };

                nau8810_codec@1a{
                        compatible = "nuvoton,nau8810";
                        reg = <0x1a>;
                };

                tlv320aic3x_codec@18{
```

Please note, unless customers do not use I2C controller on MDM9628 platform, it can be used as GPIO, which means that customers can disable I2C controller through the following method.

Disable controller device node:

```
diff --git a/mdm9607-mtp.dtsi b/mdm9607-mtp.dtsi
index b51daf0..04593cd 100644
--- a/mdm9607-mtp.dtsi
+++ b/mdm9607-mtp.dtsi
@@ -74,7 +74,7 @@
 };

 &i2c_4 {
-       status = "ok";
+       status = "disabled";
 };

 &i2c_2 {
```

I2C2 configuration as following:

```
i2c_2: i2c@78b6000 { /* BLSP1 QUP4 */
        compatible = "qcom,i2c-msm-v2";
        #address-cells = <1>;
        #size-cells = <0>;
        reg-names = "qup_phys_addr";
        reg = <0x78b6000 0x600>;
        interrupt-names = "qup_irq";
        interrupts = <0 96 0>;
        qcom,clk-freq-out = <400000>;
        qcom,clk-freq-in  = <19200000>;
        clock-names = "iface_clk", "core_clk";
        clocks = <&clock_gcc clk_gcc_blsp1_ahb_clk>,
                 <&clock_gcc clk_gcc_blsp1_qup2_i2c_apps_clk>;

        pinctrl-names = "i2c_active", "i2c_sleep";
        pinctrl-0 = <&i2c_2_active>;
        pinctrl-1 = <&i2c_2_sleep>;
        qcom,noise-rjct-scl = <0>;
        qcom,noise-rjct-sda = <0>;
        qcom,master-id = <86>;
        dmas = <&dma_blsp1 14 64 0x20000020 0x20>,
                 <&dma_blsp1 15 32 0x20000020 0x20>;
        dma-names = "tx", "rx";
```

Please note, unless customers do not use I2C controller on MDM9628 platform, it can be used as GPIO, which means that customers can disable I2C controller through the following method.

Disable controller device node:

```
--- a/ql-ol-kernel/arch/arm/boot/dts/qcom/mdm9607-mtp.dtsi
+++ b/ql-ol-kernel/arch/arm/boot/dts/qcom/mdm9607-mtp.dtsi
@@ -38,7 +38,7 @@

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

4.2.2. I2C Slave Device Configuration

By default, several Codec slave devices have been mounted below I2C controller node of device tree *mdm9628.dtsi,* which have defined compatible driver and slave device address;

```
rt5616_codec@1b{
        compatible = "realtek,rt5616";
        reg = <0x1b>;
};

nau8810_codec@1a{
        compatible = "nuvoton,nau8810";
        reg = <0x1a>;
};

tlv320aic3x_codec@18{
        compatible = "ti,tlv320aic3104";
        reg = <0x18>;
        ai3x-ocmv = <0>;
        ai3x-micbias-vg = <2>;
};
```

Please contact slave device supplier for drive and configuration guide if customers want to add new I2C device.

5 QuecOpen Application Layer API

5.1. User Programming Specification

SDK in QuecOpen project provides complete user programming interfaces;
Please refer to: *ql-ol-sdk/ql-ol-extsdk/*

```
ol@ql-Ubuntu:~/SDK/changan/ql-ol-sdk/ql-ol-extsdk$ ls
docs  example  include  lib  target  tools
ol@ql-Ubuntu:~/SDK/changan/ql-ol-sdk/ql-ol-extsdk$
```

Lib Directory as shown on the above screenshot contains API Interface lib provided by Quectel;
Include Directory is header file of all APIs; Example Directory offers API using example classified by function.

This document only introduces interfaces and examples related to I2C.

I2C Application programming needs to depend on library *libql_peripheral.a*
Header files: *ql_i2c.h*

5.2. Introduction of I2C API

When the I2C controller is in working order and I2C slave device such as Codec has been mounted to I2C-BUS, communication between Codec CPU can be made through following interface directly:

```
int QI_I2C_Init (char *dev_name);
```
Initiate the I2C device.

Parameters:
dev_name: Device name, such as */dev/i2c-4 for AG35 codec device*

Return value: File descriptor, error return value -1;

```
int QI_I2C_Read (int fd, unsigned short slaveAddr,
                 unsigned char ofstAddr,
```

```
                                unsigned char* ptrBuff,
                                unsigned short length);
```

Read specific length of bytes from a certain offset address of I2C device, error return value -1;

Parameters:
Fd: Device file descriptor
slaveAddr: Device address, 0x18(codec3104), 0x1A(codec8814), 0x1B(codec5616)
ofstAddr: Offset address (Note: Codec5616, one register has 2 bytes.)
ptrBuff: Data read by pointer pointed.
length: Length of read

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

Write specific length of bytes to a certain offset address of I2C device, error return value -1;

Parameters:
Fd: Device file descriptor
slaveAddr: Device address, 0x18(codec3104), 0x1A(codec8814), 0x1B(codec5616)
ofstAddr: Offset address (Note: Codec5616, one register has 2 bytes.)
ptrBuff: Pointer of data to be written
length: Length of writing

```
int QI_I2C_Deinit (int fd);
```
Disable I2C device

Please refer to: *ql-ol-extsdk/example/i2c*

6 I2C Function Test and Verification

6.1. Introduction and Compilation of Example

In *ql-ol-extsdk/example/i2c,* writing one-byte data to a register address of slave device that is located in some address on the specific I2C-BUS, and then reread it.

```
#define I2C_DEV         "/dev/i2c-4"    //i2c-4 on AG35
#define I2C_SLAVE_ADDR  0x18    //codec 3104
#define WHO_AM_I        0x02
#define WHO_AM_I_VALUE  0x12
```

Enter into directory: *ql-ol-sdk/ql-ol-extsdk/example/i2c*, generate executable program: *example_i2c* with *Make*. The premise of compilation is that initialization of cross compiling environment has been completed.
source ql-ol-crosstool/ql-ol-crosstool-env-init

```
ol@ql-Ubuntu:~/SDK/changan/ql-ol-sdk/ql-ol-extsdk/example/i2c$ make
arm-oe-linux-gnueabi-gcc  -march=armv7-a -mfloat-abi=softfp -mfpu=neon --sysroo
  -O2 -fexpensive-optimizations -frename-registers -fomit-frame-pointer -ftree-
quectel-features-config.h -fstack-protector-strong -pie -fpie -Wa,--noexecstack
./../lib/interface/inc -I/home/ol/SDK/changan/ql-ol-sdk/ql-ol-crosstool/sysroot
stool/sysroots/armv7a-vfp-neon-oe-linux-gnueabi/usr/include -I/home/ol/SDK/chan
I/home/ol/SDK/changan/ql-ol-sdk/ql-ol-crosstool/sysroots/armv7a-vfp-neon-oe-lin
rmv7a-vfp-neon-oe-linux-gnueabi/usr/include/qmi -I/home/ol/SDK/changan/ql-ol-sd
  -I./ -I./inc -I../../include -I/home/ol/SDK/changan/ql-ol-sdk/ql-ol-extsdk/ex
s/armv7a-vfp-neon-oe-linux-gnueabi/usr/include -I/home/ol/SDK/changan/ql-ol-sdk
gan/ql-ol-sdk/ql-ol-crosstool/sysroots/armv7a-vfp-neon-oe-linux-gnueabi/usr/inc
ux-gnueabi/usr/include/dsutils -I/home/ol/SDK/changan/ql-ol-sdk/ql-ol-crosstool
k/ql-ol-crosstool/sysroots/armv7a-vfp-neon-oe-linux-gnueabi/usr/include/qmi-fra
hread example_i2c.c
arm-oe-linux-gnueabi-gcc  -march=armv7-a -mfloat-abi=softfp -mfpu=neon --sysroo
 -L./ -L/home/ol/SDK/changan/ql-ol-sdk/ql-ol-extsdk/example/i2c/../../lib -lrt
rt -lpthread /home/ol/SDK/changan/ql-ol-sdk/ql-ol-extsdk/example/i2c/../../lib/
ol@ql-Ubuntu:~/SDK/changan/ql-ol-sdk/ql-ol-extsdk/example/i2c$ ls
example_i2c  example_i2c.c  example_i2c.o  Makefile
ol@ql-Ubuntu:~/SDK/changan/ql-ol-sdk/ql-ol-extsdk/example/i2c$
```

6.2. Function Testing

Take Codec3104 for an example:

Compile and upload *example_i2c* to the module.
Execute *adb* push *<example_i2c on the host computer path> <module inside path, such as/usrdata>* to do uploading or use serial port protocol.

If OPEN_EVB is in use, please connect I2C pin header of J0201 with jumper cap:
Screen printing GPIO_6 (corresponding CPU pin is GPIO_18) connect to I2C_SDA
Screen printing GPIO_7 (corresponding CPU pin is GPIP_19) connect to I2C_SCL
To intercommunicate hardware access.

Insert codec3104 on OPEN_EVB;

Execute *example_i2c* as following screenshot:

```
root@mdm9607-perf:~# ./example_i2c
< Ql_I2C_Init=3 >
address=0x18, offset=2, byte_num=1, value[1]=18, value[2]=0
< write i2c value=0x12, iRet=1 >
< read i2c iRet=2, value=0x012 >
```

7 I2C Driver Debugging Method

All above context is enough for customers to make device work properly. But there are always some unexpected problems, whether it is user's inappropriate operation or code problems, then debugging method is needed to troubleshot.

7.1. General Debugging Method

For the SDK in QuecOpen project, the default message level provided by Kernel Log is 4 (Kern_WARNING), which means that default message is 4 when Kernel calls *printk()* without appointing message level.

The default print level of console is 7 (KERN_DEBUG), which means that even though the Kernel Log that small than 7 still can be executed by Kernel Code, the log will be saved below Kernel *log_buffer* then. And the buffer log will be outputted to standard output when customers use *dmesg.*

If customers want to open Debug Log that already compiled into Kernel, please modify directly through entering *dmesg –n 8* in command line, or modify the code by default as below.

```
--- a/ql-ol-kernel/include/linux/printk.h
+++ b/ql-ol-kernel/include/linux/printk.h
@@ -40,7 +40,7 @@ static inline const char *printk_skip_level(const char *buffer
 #define CONSOLE_LOGLEVEL_SILENT  0 /* Mum's the word */
 #define CONSOLE_LOGLEVEL_MIN    1 /* Minimum loglevel we let people use */
 #define CONSOLE_LOGLEVEL_QUIET  4 /* Shhh ..., when booted with "quiet" */
-#define CONSOLE_LOGLEVEL_DEFAULT 7 /* anything MORE serious than KERN_DEBUG */
+#define CONSOLE_LOGLEVEL_DEFAULT 8 /* anything MORE serious than KERN_DEBUG */
 #define CONSOLE_LOGLEVEL_DEBUG 10 /* issue debug messages */
 #define CONSOLE_LOGLEVEL_MOTORMOUTH 15 /* You can't shut this one up */
```

However, in many other driver modules, they will define their own DEBUG compiling macro, the code of *printk(KERN_DEBUG)* will not be compiled if the macro hasn't been opened. Select the debug options below:
Make Kernel_Menuconfig, select 3 I2C debug options as below screenshot shows, then compile and download them.

```
<*> I2C support
[*]   Enable compatibility bits for old user-space
<*>   I2C device interface
< >   I2C bus multiplexing support
[*]   Autoselect pertinent helper modules
      I2C Hardware Bus support  --->
< >   I2C/SMBus Test Stub
[*]   I2C Core debugging messages
[*]   I2C Algorithm debugging messages
[ ]   I2C Bus debugging messages
```

At this time, more debugging messages come out:

```
root@mdm9607-perf:/sys/kernel/debug/tracing/options# /home/root/example_i2c
< Ql_I2C_Init=3 >
address=0x18, offset=2, byte_num=1, value[1]=18, value[2]=0
< write i2c value=0x12, iRet=1 >
< read i2c iRet=2, value=0x012 >
root@mdm9607-perf:/sys/kernel/debug/tracing/options# dmesg -c
[ 2217.891271] i2c i2c-2: ioctl, cmd=0x707, arg=0xbee7cc34
[ 2217.891407] i2c i2c-2: master_xfer[0] W, addr=0x18, len=2
[ 2217.891511] i2c-msm-v2 78b6000.i2c: #2775 pm_runtime: resuming...
[ 2217.891572] i2c-msm-v2 78b6000.i2c: #2690 resuming...
[ 2217.892016] i2c-msm-v2 78b6000.i2c: xfer() mode:0 msg_cnt:1 rx_cbt:0 tx_cnt:2
[ 2217.892088] i2c-msm-v2 78b6000.i2c: #708 Starting FIFO transfer
[ 2217.892150] i2c-msm-v2 78b6000.i2c: QUP state after programming for next transfers
[ 2217.892207] i2c-msm-v2 78b6000.i2c: tag.val:0x2833081 tag.len:4 (null)
[ 2217.892260] i2c-msm-v2 78b6000.i2c: #482 OUT-FIFO:0x02833081
[ 2217.892316] i2c-msm-v2 78b6000.i2c: data: 0x2 0x12 0x35 0xcf
[ 2217.892370] i2c-msm-v2 78b6000.i2c: #482 OUT-FIFO:0x00001202
[ 2217.893014] i2c-msm-v2 78b6000.i2c: NONE: msgs(n:1 cur:0 tx) bc(rx:0 tx:2) mode:FIFO slv_addr:0x18 MSTR_STS:0
[ 2217.893470] i2c i2c-2: ioctl, cmd=0x707, arg=0xbee7cc30
[ 2217.893538] i2c i2c-2: master_xfer[0] W, addr=0x18, len=1
[ 2217.893590] i2c i2c-2: master_xfer[1] R, addr=0x18, len=1
[ 2217.893776] i2c-msm-v2 78b6000.i2c: xfer() mode:0 msg_cnt:2 rx_cbt:1 tx_cnt:1
[ 2217.893831] i2c-msm-v2 78b6000.i2c: #708 Starting FIFO transfer
[ 2217.893892] i2c-msm-v2 78b6000.i2c: QUP state after programming for next transfers
[ 2217.893948] i2c-msm-v2 78b6000.i2c: tag.val:0x1823081 tag.len:4 (null)
[ 2217.894001] i2c-msm-v2 78b6000.i2c: #482 OUT-FIFO:0x01823081
[ 2217.894056] i2c-msm-v2 78b6000.i2c: data: 0x2 0x83 0x35 0xcf
[ 2217.894109] i2c-msm-v2 78b6000.i2c: tag.val:0x1873181 tag.len:4 (null)
[ 2217.894161] i2c-msm-v2 78b6000.i2c: #482 OUT-FIFO:0x87318102
[ 2217.894214] i2c-msm-v2 78b6000.i2c: #482 OUT-FIFO:0x00000001
[ 2217.894660] i2c-msm-v2 78b6000.i2c: NONE: msgs(n:2 cur:0 tx) bc(rx:1 tx:1) mode:FIFO slv_addr:0x18 MSTR_STS:0
[ 2217.894789] i2c-msm-v2 78b6000.i2c: #490 IN-FIFO :0x00120187
[ 2217.894844] i2c-msm-v2 78b6000.i2c: (null)
[ 2218.140356] i2c-msm-v2 78b6000.i2c: #2763 pm_runtime: suspending...
[ 2218.140483] i2c-msm-v2 78b6000.i2c: #2665 suspending...
root@mdm9607-perf:/sys/kernel/debug/tracing/options#
```

7.2. Debugging with Kernel Tracer

QuecOpen SDK enable Kernel Debugfs and Kernel Tracer of Kernel Hacking by default. The Kernel Tracer has powerful function, and usually be used for Kernel debugging.

```
root@mdm9607-perf:/sys/kernel/debug/tracing# ls
README                  instances             trace
available_events        options               trace_clock
available_tracers       per_cpu               trace_marker
buffer_size_kb          printk_formats        trace_options
buffer_total_size_kb    saved_cmdlines        trace_pipe
current_tracer          saved_cmdlines_size   tracing_cpumask
events                  saved_tgids           tracing_on
free_buffer             set_event             tracing_thresh
root@mdm9607-perf:/sys/kernel/debug/tracing#
```

Turn on Kernel stack trace:

`echo 1 > options/stacktrace`

Open log outputted by *printk:*

`echo 1 > events/printk/enable`

Begin I2C event debugging

`echo 1 > events/i2c/enable`

Initiate a I2C visit

Cat trace command could show I2C Kernel API call stack

```
 => ret_fast_syscall
    example_i2c-1699  [000] d..2  3358.444475: console: [ 3358.444460] i2c-msm-v2 78b6000.i2c: xfer() mode

    example_i2c-1699  [000] d..2  3358.444515: <stack trace>
 => vprintk_emit
 => dev_vprintk_emit
 => dev_printk_emit
 => __dev_printk
 => dev_printk
 => i2c_msm_frmwrk_xfer
 => __i2c_transfer
 => i2c_transfer
 => i2cdev_ioctl_rdrw
 => i2cdev_ioctl
 => do_vfs_ioctl
 => SyS_ioctl
 => ret_fast_syscall
    example_i2c-1699  [000] d..2  3358.444543: console: [ 3358.444533] i2c-msm-v2 78b6000.i2c: #708 Startin

    example_i2c-1699  [000] d..2  3358.444572: <stack trace>
 => vprintk_emit
 => dev_vprintk_emit
 => dev_printk_emit
 => __dev_printk
 => _dev_info
```