MT6853 SCP Development Guide

Doc No:	CS6853-BD9D-PGD-V1.7EN
Version:	V1.7
Release date:	2021-5-25
Classification:	Confidential B

Document Revision History

Revision	Date	Author	Description
1.0	2020-04-28	Gaia Chang	Initial version
1.1	2020-05-05	Gaia Chang	Add notify chain warning
1.2	2020-05-12	Gaia Chang	Malloc fail and scp_region_info debug
1.3	2020-05-18	Gaia Chang	Fix chap 8.6 Core Dump typos
1.4	2020-06-30	Gaia Chang	scp_region_info supplement
1.5	2020-08-05	Gaia Chang	Add 10.1 How to Enlarge DRAM region code
1.6	2020-08-06	Gaia Chang	Add Access DRAM limit and API
2.01	2021-05-25	Jerry Tseng	Add debug tool information. Modify some clerical error Add PBFR chapter

Table of Contents

Document Revision History **2**
Table of Contents **3**
1 System Overview **5**
1.1 Hardware Architecture 5
1.2 Software Architecture 7
1.3 Summary 7
2 Source Tree **8**
2.1 Little Kernel Bootloader 8
2.2 Linux Kernel Driver 8
2.3 FreeRTOS Tree 8
3 Build System **9**
3.1 Configuration Files 9
3.2 Build Commands 10
3.3 Image Layout 10
4 Boot Sequence **11**
5 Inter Processor Interrupt (IPI) **12**
5.1 Usage on the SCP Side 12
5.2 Usage on the Kernel Side 14
6 DRAM Access Procedure **16**
6.1 Reserving Memory in Linux Kernel 17
6.2 Get Reserved Memory by ID 18
6.3 Remap DRAM Address from AP View to SCP View 18
6.4 Request System Bus and DRAM 19
6.5 Access DRAM 19
7 Drivers Guide **20**
7.1 Driver Initiation 20
7.2 Add a New Driver 20
7.3 Interrupt 20
7.4 Locks 24
7.5 DMA 25
7.6 Hardware Semaphore 26
7.7 GPIO & EINT 27
8 Debug Methods **30**
8.1 PRINTF_* Usage 30
8.2 Mobile log 30
8.3 UART 31
8.4 ADB Logcat 32
8.5 Exception Log Analysis 33
8.6 Core Dump 36

9 Appendix 41
9.1 Common Issues and How to Fix 41
9.2 Code Size Limitation 42
9.3 Scp_Region_Info Structure 44
9.4 SCP Recovery 45
9.5 SCP Performance Budget Center (PBFR) 48
10 Q&A List from Users 51
10.1 How to Enlarge DRAM region code 51
10.2 How to Share Information by DRAM between AP and SCP. 51
Exhibit 1 Terms and Conditions 54

Lists of Figures and Tables

Figure 1-1. MTK 5G SCP Architecture compare with MT6765 SCP Architecture 6
Figure 1-2. MTK 5G SCP Software Architecture 7
Figure 4-1. SCP boot up flow overview 11
Figure 5-1. Tinysys common IPI architecture 12
Figure 8-1. MTK Logger 30
Figure 8-2. UART setting 31
Figure 8-3. Disable SCP Mobile log 32
Figure 8-4. ADB logcat output 33
Figure 9-3. SCP recovery behavior 46
Figure 10-1. Sharing information by DRAM between AP and SCP 51
Figure 10-2. Define size of the feature ID 52
Figure 10-3. Get Physical address and size of the reserved by feature ID 52
Figure 10-4. Use MTK API to transfer address from AP view 52

Table 1-1. Hardware specification 5
Table 1-2. Architecture Specification 7
Table 6-1. The standard DRAM access flow 16
Table 5-1. The default remapping rule table 18
Table 7-1. The EINT and GPIO mapping ball name 27
Table 7-2. The GPIO control register table 28
Table 8-1. PRINTF use scenario 30
Table 8-2. UART Pin Name 31

1 System Overview

SCP (Sensor-hub Control Processor) is a sub-system which is designed to perform always-on tasks even when system is in low power states, such as Voice Wakeup, Sensor Hub. We will describe H/W and S/W architecture of SCP in this chapter.

1.1 Hardware Architecture

SCP consists of dedicate processor(s), SRAM, DMA and peripherals, e.g. I2C, GPIO. We introduced in-house DSP and new architecture since MT6853/6885, so that always-on applications could run with lower power and gain better performance.

- Processor: in-house MDSP RV55 with RISC-V architecture
  - Single precise floating point
  - Compressed instruction
  - DSP ISA for voice acceleration
  - Individual I cache 32K, D cache 64K
  - OPPs from 250 to 624Mhz
- Memory: 768 KB TCM
- DMA:
- Peripherals:
  - I2C x 1
  - I3C x 1
  - SPI x 3
  - UART x 2
- Details

Table 1-1. Hardware specification

		MT6853
Core		RV55
Cache	L1$	I$/ D$: 32KB/64KB
	L2$	NA
TCM	L1TCM	NA
	L2TCM	768KB
Peripherals		▪ I2C x 1 ▪ SPI x 3 ▪ I3C x 1 ▪ UART x 2
DMA		4 channels (0 reserved for i2c)
VoW I/F		2-mic
Operating Frequency		250M Hz@0.55V
Performance (CoreMark@Vmin)		1,170
Power Efficiency (CoreMark/mW)		221

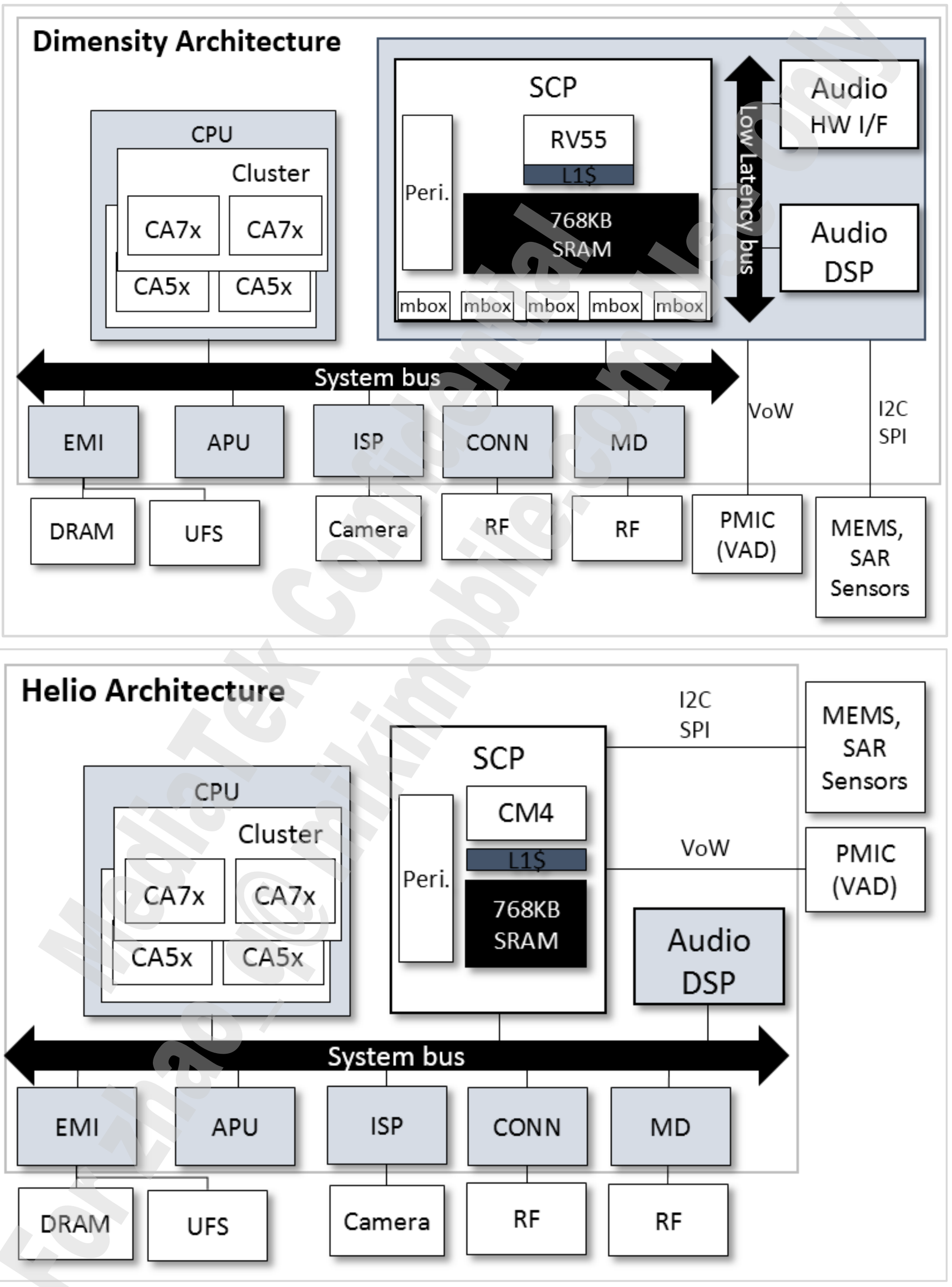


Figure 1-1. MTK 5G SCP Architecture compare with MT6765 SCP Architecture

1.2 Software Architecture

SCP SW is based on AWS freeRTOS v10.1.0.1 which is a real time kernel in MIT v2 license supports multi-task, mutex, semaphore and software timer.
The SW package also includes additional middleware such as audio and sensor hub, but we won't discuss about details in this document.

Let's zoom out a little bit to get a whole picture. SCP communicates with Linux kernel via IPI (Inter Processor Interrupt) based on OpenAMP rpmsg/remoterproc framework. Similarly, we provide other mechanisms to enable co-work between AP and SCP:

- IPI – Chapter 5
- HW semaphore – Chapter 7.6
- Logger – Chapter 8.2
- Recovery – Chapter 9.3

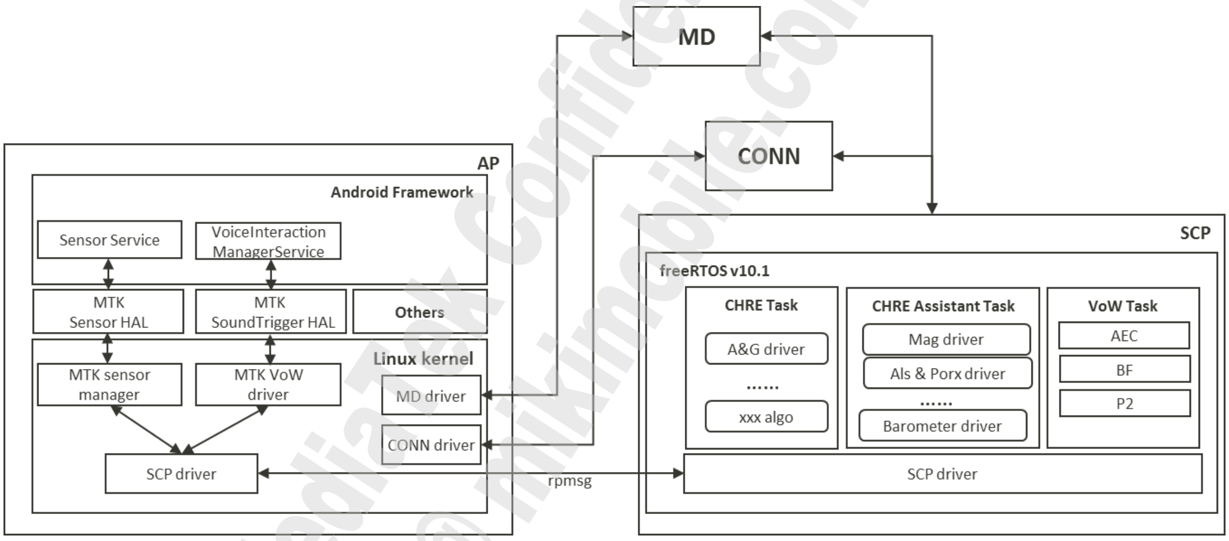


Figure 1-2. MTK 5G SCP Software Architecture

1.3 Summary

Table 1-2. Architecture Specification

Item	Value
Platform	MT6853
Project	$PROJECT
Linux version	4.14
FreeRTOS version	10.1.0.1
ISA	RV55
L2 TCM	768KB

2 Source Tree

SCP source tree includes bootloader (i.e. LK), Linux kernel and FreeRTOS. List as below:

2.1 Little Kernel Bootloader

- vendor/mediatek/proprietary/bootable/bootloader/lk/platform/MT6853MT6853/mt_scp.c

2.2 Linux Kernel Driver

- SCP driver path
  - kernel-4.14/drivers/misc/mediatek/scp
- SCP DTS path
  - kernel-4.14/arch/arm64/boot/dts/mediatek/MT6853.dts

2.3 FreeRTOS Tree

- RTOS kernel
  - vendor/mediatek/proprietary/tinysys/kernel/FreeRTOS_v10.1.0.1
- Platform and peripheral drivers
  - vendor/mediatek/proprietary/tinysys/scp
  - vendor/mediatek/proprietary/tinysys/common
- Libraries
  - vendor/mediatek/proprietary/tinysys/scp/middleware
- Toolchain
  - prebuilts/clang/md32rv/linux-x86

3 Build System

Before digging into programming, developers might want to know how to compile with basic SCP configuration. This chapter will show where/how configurations are arranged, and how to produce SCP image by commands.

3.1 Configuration Files

3.1.1 Little Kernel Bootloader

- vendor/mediatek/proprietary/bootable/bootloader/lk/project/MT6853.mk

3.1.2 Linux Kernel

- kernel-4.14/arch/arm64/configs/$PROJECT_debug_defconfig
- kernel-4.14/arch/arm64/configs/$PROJECT_defconfig
    - Enable/disable SCP driver: CONFIG_MTK_TINYSYS_SCP_SUPPORT
    - Features switch, such as Voice Wakeup and Sensor Hub
    - Enable/disable SCP driver: MTK_TINYSYS_SCP_SUPPORT

3.1.3 FreeRTOS

- vendor/mediatek/proprietary/tinysys/scp/project/RV55_A/MT6853/platform/platform.mk
    - **Default** configurations of the platform
    - Extra CFLAGS
    - Extra LDFLAGS
    - Driver/middleware C objects and include path
- vendor/mediatek/proprietary/tinysys/scp/project/RV55_A/MT6853/project.mk
    - Project-specific configuration
    - **Customize** project by overriding options in platform.mk

3.2 Build Commands

There are several ways to build SCP images. We just list 2 of them in ALPS SW packages.

3.2.1 Standalone

This method can build SCP firmware quickly, especially useful when development.

- The output path: ./tinysys_out
- SYNOPSIS: *PROJECT=XXX*$PROJECT *BUILD_TYPE=[release, debug] make*
- Ex:

```
$ cd vendor/mediatek/proprietary/tinysys/scp
$ PROJECT=k6853v1_64$PROJECT BUILD_TYPE=debug make -j24
```

3.2.2 With AOSP Hierarchy

- The output path: out/target/product/$PROJECT/obj/TINYSYS_OBJ/tinysys-scp_intermediates/RV55_A/scp
- Ex:

```
$ make tinysys-scp -j24
$ mmm vendor/mediatek/proprietary/tinysys/scp -j24
$ cd vendor/mediatek/proprietary/tinysys/scp && mm -j24
$ mmm vendor/mediatek/proprietary/tinysys/scp:tinysys-scp -j24
```

3.3 Image Layout

- Partition in EMMC/UFS: scp1/scp2
  - scp1: main and active partition
  - scp2: backup for AB system
- Image: scp.img
  - Consists of:
    1. tinysys-scp-RV55_A.bin: firmware/data located in SRAM
    2. tinysys-scp-RV55_A.elf: elf with symbol, for debug purpose
    3. tinysys-scp-RV55_A_DRAM.bin: firmware/data located in DRAM

4 Boot Sequence

After the SCP image is ready, we need to know how the image are loaded into SRAM/DRAM and how SCP is initiated to run.

The whole flow is completed by LK booloader/Linux kernel/SCP firmware together, as below:

- LK Bootloader (vendor/mediatek/proprietary/bootable/bootloader/lk/ platform/MT6853/**mt_scp.c**)
  - Allocate permanent DRAM memory for SCP image
  - Load/verify SCP image
  - Setting EMI MPU (AP read-only)
- Kernel
  - Initial setting (mbox/ipi/logger/…)
  - Kick SCP
- SCP
  - Loader (vendor/mediatek/proprietary/tinysys/scp/project/RV55_A/MT6853/platform/***boot.S***)
    - Load SCP image to SRAM
    - Jump to FreeRTOS
  - FreeRTOS (vendor/mediatek/proprietary/tinysys/scp/project/RV55 _A/MT6853/platform/src/***main.c***)
    - Driver init
    - Setup MPU

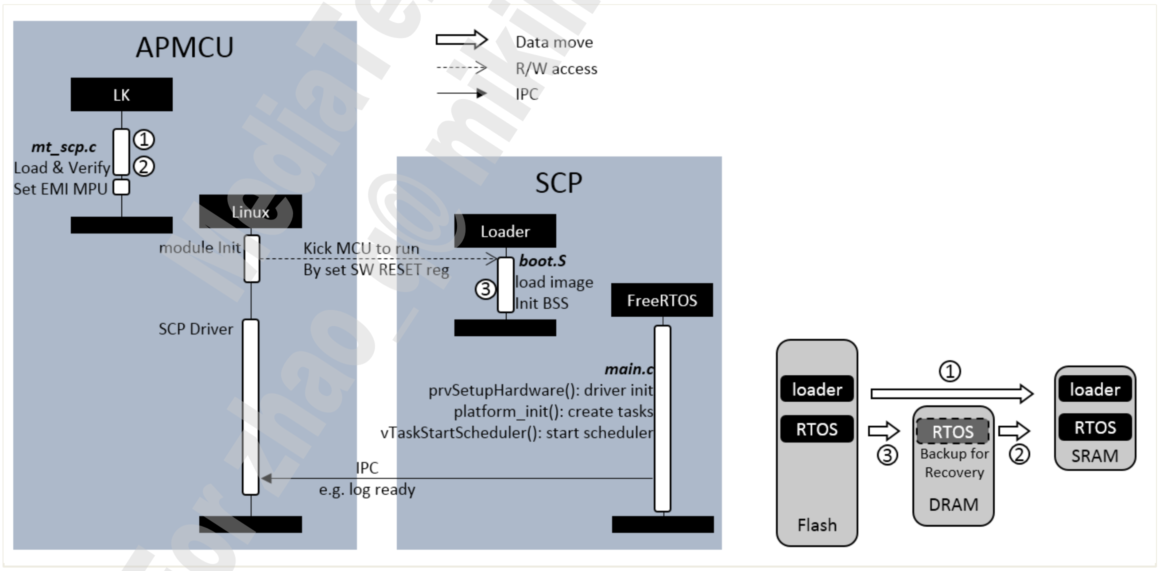


Figure 4-1. SCP boot up flow overview

5 Inter Processor Interrupt (IPI)

IPI is a mechanism to pass messages between drivers in Linux and FreeRTOS. It consists of:

1. A piece share memory: to exchange data
2. A set of interrupts: to notify each other

Figure 5-1 shows the Tinysys IPI architecture. The SW architecture consists of 4 layers

1. Synchronization layer: public API for communication between AP and Tinysys
2. Rpmsg layer: blocking/non-blocking send functions
3. Queue layer: queue operation and management functions
4. Physical layer: physical hardware operations

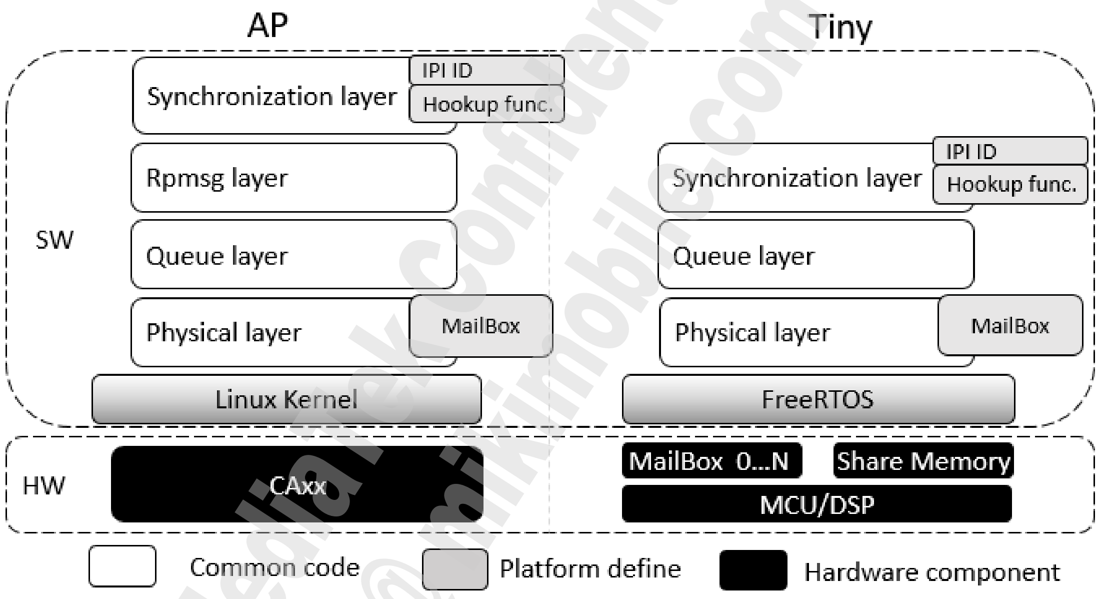


Figure 5-1. Tinysys common IPI architecture

The MailBox are only predefined. Developers use the flowing API to send IPI to Tinysys, including IPI ID and register the call back function (ipi_cb).

5.1 Usage on the SCP Side

i. Add a new IPI id to *enum ipi_id* in ipi_legacy_wrap.h
   - **Path**
     vendor/mediatek/proprietary/tinysys/scp/drivers/RV55 _A/MT6853/mbox/ipi_legacy_wrap.h

ii. Register an IPI handler
   - **API**

ipi_status scp_ipi_registration(enum ipi_id id, ipi_handler_t handler, const char *name)

Description

To register IPI handler

Parameters

id: id declared in ipi_legacy_wrap.h

handler: IPI handler, a callback

name: a string to recognize IPI handler

Return values

DONE: complete successfully

ERROR: something wrong in lower layer driver, i.e. mbox

BUSY: the channel is busy. Need to retry.

- **Example**

```
#include “scp_ipi.h”

ipi_status ret;
ret = scp_ipi_registration(IPI_NEW_ID, ipi_cb, “ipi_cb name”);
if (ret != DONE)
      PRINTF_E(“Register IPI failed\n”);
```

```
void ipi_cb(int id, void *data, unsigned int len)
{
      /* data: received message from kernel */
      ....
      return 0;
}
```

iii. Send an IPI

- **API**

ipi_status scp_ipi_send(enum ipi_id, void *buf, uint32_t len, uint32_t wait, enum ipi_id dir)

Description

API for apps to send an IPI to scp

Parameters

id: IPI id declared in ipi_legacy_wrap.h
buf: the message which will be sent to Linux kernel
len: message length(in bytes)
wait: wait(1) or not(0)
dir: direction; default is IPI_SCP2AP

Return values

DONE: complete successfully

ERROR: something wrong in lower layer driver, i.e. mbox

- **Example:**

```
#include "scp_ipi.h"

ipi_status ret;
ret = scp_ipi_send (IPI_NEW_ID, (void *)&msg, msg_size, 0, IPI_SCP2AP);
if (ret != DONE)
      PRINTF_E("Send IPI failed\n");
```

5.2 Usage on the Kernel Side

- Add a new IPI id to *enum ipi_id* in scp_ipi_wrapper.h
- **Path**
  kernel-4.14/drivers/misc/mediatek/scp/MT6853/scp_ipi_wrapper.h
- Register an IPI
- **API**

enum scp_ipi_status scp_ipi_registration(enum ipi_id id, void(*ipi_handler)(int id, void *data, unsigned int len), const char *name)

Description

To register IPI handler

Parameters

id: id declared in ipi_legacy_wrap.h

handler: IPI handler, a callback

name: a string to recognize IPI handler

Return values

DONE: complete successfully

ERROR: something wrong in lower layer driver, i.e. mbox

- Send an IPI
- **API**

enum scp_ipi_status scp_ipi_send(enum ipi_id, void *buf, unsigned int len, unsigned int wait, enum scp_core_id scp_id)

Description

API for apps to send an IPI to scp

Parameters

id: IPI id declared in ipi_legacy_wrap.h
buf: the message which will be sent to Linux kernel
len: message length(in bytes)
wait: wait(1) or not(0)
dir: direction; default is IPI_SCP2AP
scp_id: send to which SCP, choose SCP_A_ID as default

Return values

DONE: complete successfully

ERROR: something wrong in lower layer driver, i.e. mbox

SCP_NOT_READY: scp is not ready. Need to retry.

6 DRAM Access Procedure

Because DRAM and the system bus could be turned off regardless SCP's status, SCP developers **MUST** follow the following procedure to avoid system hang.

Table 6-1. The standard DRAM access flow

#	Steps	Note
1	Check/add predefined table in scp_reserve_mblock[]	
2	SCP Linux driver gets reserved physical address by APIs.	phys_addr_t scp_get_reserve_mem_phys(scp_reserve_mem_id_t id) phys_addr_t scp_get_reserve_mem_virt(scp_reserve_mem_id_t id) phys_addr_t scp_get_reserve_mem_size(scp_reserve_mem_id_t id)
3	SCP Linux driver sends the address to SCP via IPI.	scp_ipi_send(uint32 id, void* buf, uint len)
4	SCP accesses DRAM via API.	uint32_t ap_to_scp(uint32_t ap_addr)
5	Enable DRAM before using.	void dvfs_enable_DRAM_resource(scp_reserve_mem_id_t dma_id) void dvfs_disable_DRAM_resource(scp_reserve_mem_id_t dma_id)

6.1 Reserving Memory in Linux Kernel

To reserve a space in DRAM to exchange data between AP and SCP, we need to add entries in scp_reserve_mblock[] first. The definition could be found here:

- **Path**
  - kernel-4.14/drivers/misc/mediatek/scp/MT6853/scp_helper.h
  - kernel-4.14/drivers/misc/mediatek/scp/MT6853/scp_helper.c
- **Example**

```
enum scp_reserve_mem_id_t{
      VOW_MEM_ID,
      SENS_MEM_ID,
...
      SCP_DRV_PARAMS_MEM_ID,
      NUMS_MEM_ID ,
};
static scp_reserve_mblock_t scp_reserve_mblock[] = {
   {
      .num = VOW_MEM_ID,
      .start_phys = 0x0,
      .start_virt = 0x0,
      .size = 0x34E000,/* 211 KB*/
   },
   {
      .num = SENS_MEM_ID,
      .start_phys = 0x0,
      .start_virt = 0x0,
      .size = 0x100000,/*1MB*/
   },
...
   {
      .num = SCP_DRV_PARAMS_MEM_ID,
      .start_phys = 0x0,
      .start_virt = 0x0,
      .size = 0x100,/* 256 bytes*/
   },
};
```

6.2 Get Reserved Memory by ID

Because the AP CPU uses virtual address and SCP uses physical address, both the two types of address are supplied. The following APIs are used to get virtual/physical address and size by given IDs which are declared in 6.1.

- **API**
  - phys_addr_t scp_get_reserve_mem_phys(scp_reserve_mem_id_t id)
  - phys_addr_t scp_get_reserve_mem_virt(scp_reserve_mem_id_t id)
  - phys_addr_t scp_get_reserve_mem_size(scp_reserve_mem_id_t id)
- **Header Path**
  - kernel-4.14/drivers/misc/mediatek/scp/MT6853/scp_helper.h
- **Return Value**
  - The start address of the reserved memory, or
  - 0x0: means no mapping

After getting the physical address, developers **MUST** pass it to SCP via IPI, as mentioned in 5.2.

6.3 Remap DRAM Address from AP View to SCP View

SCP is a 32 bit system and can only access maximum 4G (0xffffffff) address. If SCP applications need to access higher than 4G, we need to use the following APIs to remap address to SCP memory window. Here is the mapping table which shows the relation between AP and SCP.

Table 5-2. The default remapping rule table

Name	SCP/DMA		Size	AP side	
External Memory	0x1000_0000	0x1FFF_FFFF	256MB	0x6000_0000	0x6FFF_FFFF
External Memory	0x2000_0000	0x2FFF_FFFF	256MB	0x7000_0000	0x7FFF_FFFF
External Memory	0x5000_0000	0x5FFF_FFFF	256MB	0x0000_0000	0x0FFF_FFFF
External Memory	0x6000_0000	0x6FFF_FFFF	256MB	0x1000_0000	0x1FFF_FFFF
External Memory	0x9000_0000	0x9FFF_FFFF	256MB	0x8000_0000	0x8FFF_FFFF
External Memory	0xA000_0000	0xAFFF_FFFF	256MB	0x9000_0000	0x9FFF_FFFF
External Memory	0xD000_0000	0xDFFF_FFFF	256MB	0x2000_0000	0x2FFF_FFFF
External Memory	0xE000_0000	0xEFFF_FFFF	256MB	0x3000_0000	0x3FFF_FFFF
External Memory	0xF000_0000	0xFFFF_FFFF	256MB	0x5000_0000	0x5FFF_FFFF

- **API**
  - uint32_t ap_to_scp(uint32_t ap_addr);
  - uint32_t scp_to_ap(uint32_t scp_addr);
- **Header Path**
  - vendor/mediatek/proprietary/tinysys/scp/drivers/common/dma/inc/dma_api.h
- **Return Value**
  - The mapped address, or
  - 0x0: means no mapping

6.4 Request System Bus and DRAM

Because the system bus and DRAM will enter sleep mode when no data is being transmitted or system suspends, the following APIs **MUST** be invoke to make sure DRAM is ready to access.

- **API for task context**
  - void dvfs_enable_DRAM_resource(scp_reserve_mem_id_t dma_id): before DRAM access
  - void dvfs_disable_DRAM_resource(scp_reserve_mem_id_t dma_id): after DRAM access

Note
It will take 5ms to wake up DRAM when 26M clock is gated.

- **API for ISR context**
  - void dvfs_enable_DRAM_resource_from_isr(scp_reserve_mem_id_t dma_id)
  - void dvfs_disable_DRAM_resource_from_isr(scp_reserve_mem_id_t dma_id)
- **Header Path**
  - vendor/mediatek/proprietary/tinysys/common/drivers/dma/v3/inc/dma.h
  - vendor/mediatek/proprietary/tinysys/scp/drivers/RV55_A/MT6853/dvfs/inc/dvfs.h

6.5 Access DRAM

MT6853 seems DRAM write as IO write, please wrapper any DRAM access with Memory IO write API
API:
32bit write → mem_write32(addr, data)
16bit write → mem_write16(addr, data)
8bit write → mem_write8(addr, data)

Example
*(unsigned int *) reserve_dram_addr1 = ap_to_scp(**dram_address_from_AP**);
*(unsigned int *) reserve_dram_addr1 = 0x1234;   //please do not write directly
→
*(unsigned int *) reserve_dram_addr1 = ap_to_scp(dram_address_from_AP);
mem_write32((unsigned int *) reserve_dram_addr1, 0x1234);

▪

7 Drivers Guide

7.1 Driver Initiation

Due to the multi-thread is not enabled when driver init, developers **MUST**

- put drivers init function in platform_init().
- NEVER use block functions in driver init function, because it will block forever. For example:
  - vTaskDelay
  - HW semaphore
  - Busy loop, e.g. polling registers

7.2 Add a New Driver

Here are steps to add a new driver to the source tree:

1. Put the driver body in the appropriate folder (choose one of following folders)
   - **Path**
     - vendor/mediatek/proprietary/tinysys/common /* common drivers for tinysys */
     - vendor/mediatek/proprietary/tinysys/scp/drivers/common /* common drivers for scp */
     - vendor/mediatek/proprietary/tinysys/scp/drivers/RV55_A/MT6853/drivers /* platform drivers */
2. add a new compiler option
   - **Path**
     - vendor/mediatek/proprietary/tinysys/scp/project/MT6853/platform/platform.mk

   - **Example**:DMA driver

```
CFG_DMA_SUPPORT = yes
...
ifeq ($(CFG_DMA_SUPPORT),yes)
   INCLUDES += $(COMMON_DIR)/drivers/dma/v3/inc
   INCLUDES += $(SCP_DRIVERS_DIR)/common/dma/inc
   INCLUDES += $(DRIVERS_PLATFORM_DIR)/dma
   C_FILES  += $(COMMON_DIR)/drivers/dma/v3/dma.c
   C_FILES  += $(SCP_DRIVERS_DIR)/common/dma/dma_api.c
endif
```

7.3 Interrupt

MT6853 SCP supports 15 priority levels of interrupt, and the lower level number with the higher priority. If more than one IRQ happen at the same time, CPU will serve the one with highest priority first.

NEVER set interrupt priorities higher than 2. Level 0 is for the "watch dog" or "system fail", level 1 is for "sleep control" interrupt.
We will describe how to use interrupt in SCP in this chapter.

- **Path**
  vendor/mediatek/proprietary/tinysys/common/drivers/irq/v3/inc/irq.h

7.3.1 IRQ Registration

Before an IRQ is served, the driver developer have to make an association between the IRQ ID and the corresponding handler. The IRQ is defined according the structure of INTC_IRQ, and the driver developer following API to register handlers with IRQ IDs.

- **Structure**

```
struct INTC_IRQ
{
   uint8_t id;
   uint8_t group;
   uint8_t pol;
}
```

id: the irq number
group: from INTC_GRP_0 (the highest priority) to INTC_GRP_14 (the lowest priority)
pol: the polarity with INTC_POL_HIGH or INTC_POL_LOW

- **API**

```
int intc_irq_request(struct INTC_IRQ *irq, irq_handler_t handler, void *userdata)
```

Description
Request an irq and register the handler.
Parameters
irq: the irq structure which is declared at intc.h
handler: irq handler
userdata: it will deliver to irq handler as a parameter
Return values
0: success
-1: request fail

- **Example**

```
struct INTC_IRQ INTC_IRQ_SYSTICK = {0, INTC_GRP_8, INTC_POL_HIGH};

#include “irq.h”

int ret;
ret = intc_irq_request(&INTC_IRQ_SYSTICK, test_ist, NULL);
if (ret != 0)
      PRINTF_E(“Register irq failed\n”);
```

7.3.2 Enable IRQ

After IRQ handler is registered, SCP is ready to service. The next step is to enable a specific IRQ with the following API.

- **API**

int intc_irq_enable(struct INTC_IRQ *irq)

Description

Enable specified irq served.

Parameters

irq: the irq structure which is declared at intc.h

Return values

0: success

-1: request fail

- **Example:**

```
#include “irq.h”

int ret;
ret = intc_irq_enable(&INTC_IRQ_SYSTICK);
if (ret != 0)
      PRINTF_E(“enable irq failed\n”);
```

7.3.3 Disable IRQ

Opposite, when IRQs are not used temporarily, driver developers have to call following API to make SCP stop service the IRQs.

- **API**

int intc_irq_disable(struct INTC_IRQ *irq)

Description

Disable specified irq served.

Parameters

irq: the irq structure which is declared at intc.h

Return values

0: success

-1: request fail

- **Example**

```
#include “irq.h”

int ret;
ret = intc_irq_dsiable(&INTC_IRQ_SYSTICK);
if (ret != 0)
      PRINTF_E(“disable irq failed\n”);
```

7.3.4 Wakeup Source Registration

When SCP is in the sleep state, an IRQ will not be serviced unless it is set as wakeup source. The following API is used to register an IRQ as a wakeup source.

- **API**

int intc_irq_wakeup_set(struct INTC_IRQ *irq, unsigned int wake_src)

Description

Set wakeup source for specified irq.

Parameters

irq: the irq structure which is declared at intc.h

wake_src: 1 for wakeup source, 0 for non-wakeup source

Return values

0: success

-1: request fail

- **Example**

```
#include “irq.h”

int ret;
ret = intc_irq_wakeup_set(&INTC_IRQ_SYSTICK, 1);
if (ret != 0)
      PRINTF_E(“irq wakeup source setup failed\n”);
```

7.4 Locks

SCP provide spin_lock mechanism for dual core synchronize, the user only implement single core do not use spin_lock API. User enable support with set CFG_ATOMIC_PLAT_SUPPORT = yes

WARNING

1. ISR execution time must be **as SHORT as possible** and stack **depth as LESS as possible**
2. **NEVER** use block APIs, such as
   - HW semaphore API
   - Wait for IPI
   - Any APIs with polling external devices
3. **NEVER** use FreeRTOS API in ISR without FromISR postfix
   - It's a rewritten version with block-free API and quick implementation
4. **MUST** use portYIELD_FROM_ISR() if there is a high priority task waken in ISR

- **Path**
  vendor/mediatek/proprietary/tinysys/scp/project/RV55_A/MT6853/platform/platform.mk

- **API**

spinlock_t SYNC_SECTION lock;

void spinlock_lock(spinlock_t * lock)
void spinlock_unlock(spinlock_t * lock)

description

spin lock for Sync dual core.

parameters

lock: variable defined by SYNC_SECTION

Return values

NA.

- **Header Path**
  vendor/mediatek/proprietary/tinysys/scp/project/RV55_A/MT6853/platform/inc/mtk_atomic.h

WARNING

1. **NEVER** keep spin lock more than 1ms due to preempt is disabled during spin lock and unlock.

7.5 DMA

Direct Memory Access (DMA) is a kind of hardware that supports copy data from/to specified source/destination without involving CPU.
SCP DMA supports:

1. burst AXI mode to speed up memory translations
2. 8 channels, i.e. the engine could operate up to 8 transactions simultaneously.

- **API**

DMA_RESULT scp_dma_transaction(uint32_t dst_addr, uint32_t src_addr, uint32_t len, int8_t scp_dma_id, int32_t ch)
DMA_RESULT scp_dma_transaction_dram(uint32_t dst_addr, uint32_t src_addr, uint32_t len, int8_t scp_dma_id, int32_t ch)

description

Copy data from src_addr to dst_addr by specified DMA channel

parameters

dst_addr: destination address
src_addr: source address
len: length in bytes to copy
scp_dma_id: DMA ID
ch: channel ID

Return values

DMA_RESULT_DONE (=0) means success start
DMA_RESULT_NO_FREE_CH (=-1) means DMA hardware busy

- **Header Path**
  - vendor/mediatek/proprietary/tinysys/common/drivers/dma/v3/inc/dma.h
  - vendor/mediatek/proprietary/tinysys/scp/drivers/common/dma/dma_api.h
  - vendor/mediatek/proprietary/tinysys/scp/project/rv55_a/mt6853/mt_dma.h
- **scp_dma_id**
  - It's the identity of DMA channel in mt_dma.h. It's recommended to use different dma_id with which debugging channel full issues will be easier.
- **Limitations**
  - The maximum data size per transaction is 262140 bytes
  - Use 4byte-aligned address to get the best performance
- **Examples**:

```
ret = scp_dma_transaction(dst_buf + dst_w_pos, src_buf + src_r_pos, src_len, LOGGER_DMA_ID,
NO_RESERVED);

if (ret != DMA_RESULT_DONE) {

        PRINTF_E("log dma trans fail%u\n", ret);

        return 0;

}
```

7.6 Hardware Semaphore

Hardware semaphore is a special hardware which provides mutex-like flow control between Linux driver and FreeRTOS. There are 16 sets in SCP.

The following APIs make Hardware semaphore easy to use. They work on both SCP and Linux driver. Just include the right header and make sure the flags are the same.

- **API**

int semaphore_get(unsigned int flags)
int semaphore_release(unsigned int flags);

description

Semaphore between AP and SCP.

parameters

flag: 0 ~15 for 16 sets in SCP

Return values

0: get semaphore fail

1: get semaphore success

- **Header**
  - kernel-4.14/drivers/misc/mediatek/scp/MT6853/scp_helper.h
  - vendor/mediatek/proprietary/tinysys/common/drivers/sem/v1/inc/sem.h
- **Return**
  - 1: success get semaphore
  - 0: fail to get semaphore

- **Examples**:

```
int get_ semaphore;

while(1) {
  get_semaphore = semaphore_get(4)
  if (get_semaphore)
    break;
}
```

7.7 GPIO & EINT

SCP also provides GPIO an external interrupts so that external components such as gyro sensors could send events to SCP.

- **GPIOs**
  - Function set to: TP_GPIO?_AO, support TP_GPIO0_AO~TP_GPIO15_AO
- **EINTs**
  - Function set to Aux Func.0(GPIO), support EINT0~15

Table 7-1. The EINT and GPIO mapping ball name

Ball name	GPIO Reset Default Mode	EINT	Aux Func.0	Aux Func.4	Aux Func.6
EINT0	0	EINT0	B:GPIO0		B0:TP_GPIO0_AO
EINT1	0	EINT1	B:GPIO1		B0:TP_GPIO1_AO
EINT2	0	EINT2	B:GPIO2		B0:TP_GPIO2_AO
EINT3	0	EINT3	B:GPIO3		B0:TP_GPIO3_AO
EINT4	0	EINT4	B:GPIO4		B0:TP_GPIO4_AO
EINT5	0	EINT5	B:GPIO5		B0:TP_GPIO5_AO
EINT6	0	EINT6	B:GPIO6		B0:TP_GPIO6_AO
EINT7	0	EINT7	B:GPIO7		B0:TP_GPIO7_AO
EINT8	0	EINT8	B:GPIO8	B0:TP_GPIO8_AO	
EINT9	0	EINT9	B:GPIO9	B0:TP_GPIO9_AO	
EINT10	0	EINT10	B:GPIO10		B0:TP_GPIO10_AO
EINT11	0	EINT11	B:GPIO11		B0:TP_GPIO11_AO
EINT12	0	EINT12	B:GPIO12		B0:TP_GPIO12_AO
EINT13	0	EINT13	B:GPIO13		B0:TP_GPIO13_AO
EINT14	0	EINT14	B:GPIO14		B0:TP_GPIO14_AO
EINT15	0	EINT15	B:GPIO15		B0:TP_GPIO15_AO

7.7.1 GPIO Usage

The GPIO function must be set to TP_GPIO?_AO first. Please refer to GPIO pin mux setting document for the detail. The GPIO control register table shown in Table 6-1.

Table 7-2. The GPIO control register table

Register				
Offset	**Name**	**Description**	**Access**	**Enumeration**
0x25000	**GPIO_DIR**	**GPIO direction**	**RW**	**0: Input** **1: Output**
0x25004	**GPIO_OUT**	**GPIO output**	**RW**	**GPIO output [1:0]**
0x25008	**GPIO_IN**	**GPIO input**	**RO**	**GPIO input[1:0]**
0x2500C	**GPIO_PULL_EN**	**GPIO pull enable**	**RW**	**0: Disable** **1: Enable**
0x25010	**GPIO_PULL_CTRL**	**GPIO pull control**	**RW**	**0: Pull down** **1: Pull up**

The control examples:

- Pull high GPIO 1
  - Set GPIO_DIR, 0x60525000[1] = 1
  - Set GPIO_OUT, 0x65025004[1] = 1
- Read GPIO 0
  - Set GPIO_DIR, 0x60525000[0] = 0
  - Read GPIO_IN, 0x60525008[0]
- Internal pull up GPIO 3
  - Set GPIO_PULL_EN, 0x6052500c[3] = 1
  - Set GPIO_PULL_CTRL, 0x60525010[3] = 1

7.7.2 EINT Usage

- **Path**
  - tinysys/common/drivers/eint/v02/src
- **API:** register EINT call back

```
void mt_eint_registration(unsigned int eint_num, unsigned int sens, unsigned int pol,
                          void (EINT_FUNC_PTR) (int),
                          unsigned int unmask, unsigned int is_auto_umask)
```

description

Register eint interrupt handler.

parameters

eint num: the EINT number to register

sens: LEVEL_SENSITIVE, EDGE_SENSITIVE

pol: HIGH_LEVEL_TRIGGER, LOW_LEVEL_TRIGGER

EINT_FUNC_PTR: the ISR callback function

Unmask: enable this EINT trigger after register

Is_auto_unmask: auto reenable EINT trigger after finish a EINT service routine

Return values

NA.

- **Example**

```
mt_eint_registration(eint_num, LEVEL_SENSITIVE, HIGH_LEVEL_TRIGGER, xxx_Isr,
                     EINT_INT_UNMASK, EINT_INT_AUTO_UNMASK_OFF);
```

- void mt_eint_dis_hw_debounce(unsigned int eint_num): disable HW debounce
- void mt_eint_soft_set(unsigned int eint_num) : software trigger to clear specified EINT.

8 Debug Methods

8.1 PRINTF_* Usage

Please **DO NOT** use printf in SCP SW, because it could be linked to C library and cause problem. Instead, we use PRINTF_* as table shows. Developers **MUST** include the header file <mt_printf.h> before using PRINTF_*. The log level and the use scenario of PRINTF_x shown in Table.

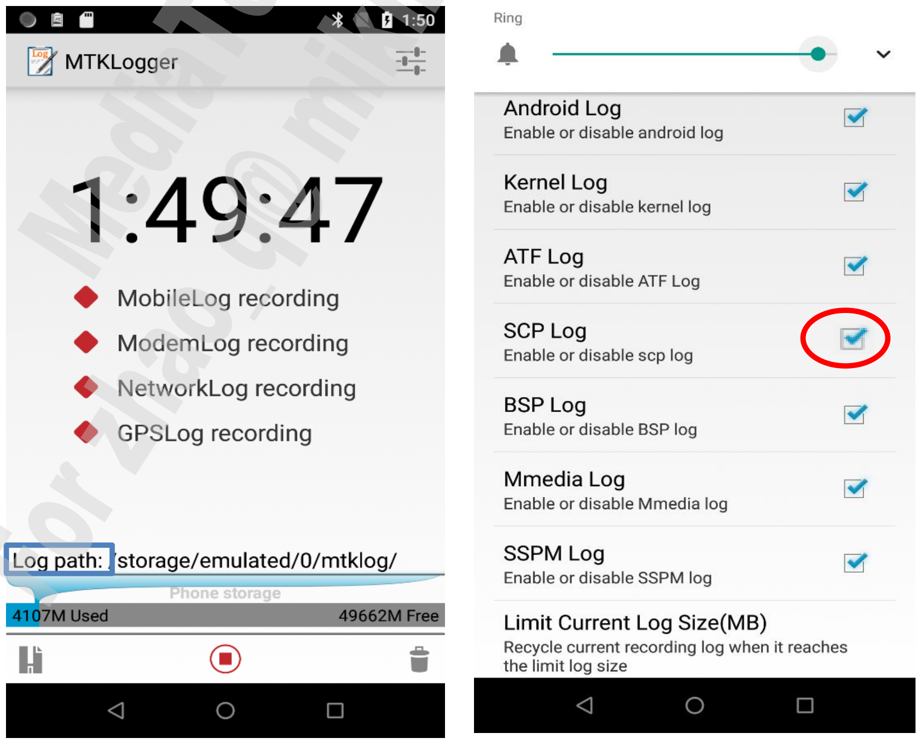
Table 8-1. PRINTF use scenario

PRINTF_*	level	Use scenario	
PRINTF_E	<0>	Error conditions	
PRINTF_W	<1>	Warning conditions	User load
PRINTF_I	<2>	Informational	
PRINTF_D	<3>	Debug-level messages	Engineer load

8.2 Mobile log

MTK Logger is an APK which records various log into storage such as SD card. After launch it and enable SCP log, SCP log could be gotten in the following path:

- Log path: /mobilelog/APLog_XXXX_XXXX_XXXXXX/scp_log_XXXX.curf



Android Log — Enable or disable android log ✓
Kernel Log — Enable or disable kernel log ✓
ATF Log — Enable or disable ATF Log ✓
SCP Log — Enable or disable scp log ✓
BSP Log — Enable or disable BSP log ✓
Mmedia Log — Enable or disable Mmedia log ✓
SSPM Log — Enable or disable SSPM log ✓
Limit Current Log Size(MB) — Recycle current recording log when it reaches the limit log size

Figure 8-1. MTK Logger

8.3 UART

- **Output pin**

SCP has 2 dedicate UART. Please make sure the PC UART port connect to Pin ball name URTD1 and UTXD1 (as shown in Table 7-2) and then set the software compiler option as below.

Table 8-2. UART Pin Name

Pin name	Function
URXD1	SCP UART RX
UTXD1	SCP UART TX

- **SW compiler option**
  - **Path:** configure flags
    vendor/mediatek/proprietary/tinysys/scp/project/RV55_A/MT6853/platform/platform.mk

– CFG_UART_SUPPORT = yes /* Uart enable, **Default No** */
– CFG_MTK_SCPUART_SUPPORT = yes /* Use SCP uart, Default Yes*/
– CFG_MTK_APUART_SUPPORT = no /* Use AP uart, Default No*/

Warning

1. CFG_MTK_SCPUART_SUPPORT and CFG_MTK_APUART_SUPPORT **CAN NOT** be Yes at the same time.
2. **DO NOT** enable CFG_MTK_APUART_SUPPORT to ENG build, because AP and SCP log will mix together and hard to recognize.
3. **DO NOT** enable CFG_MTK_APUART_SUPPORT for power measurement, it keeps AP awake.

- **UART terminal setting**
  - Baud rate: 921600

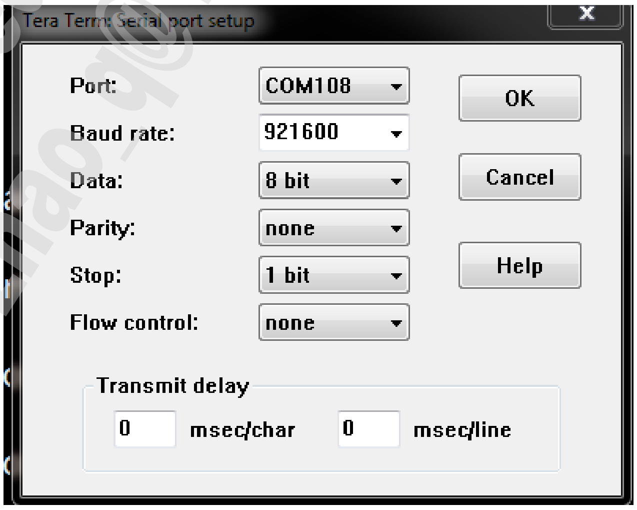
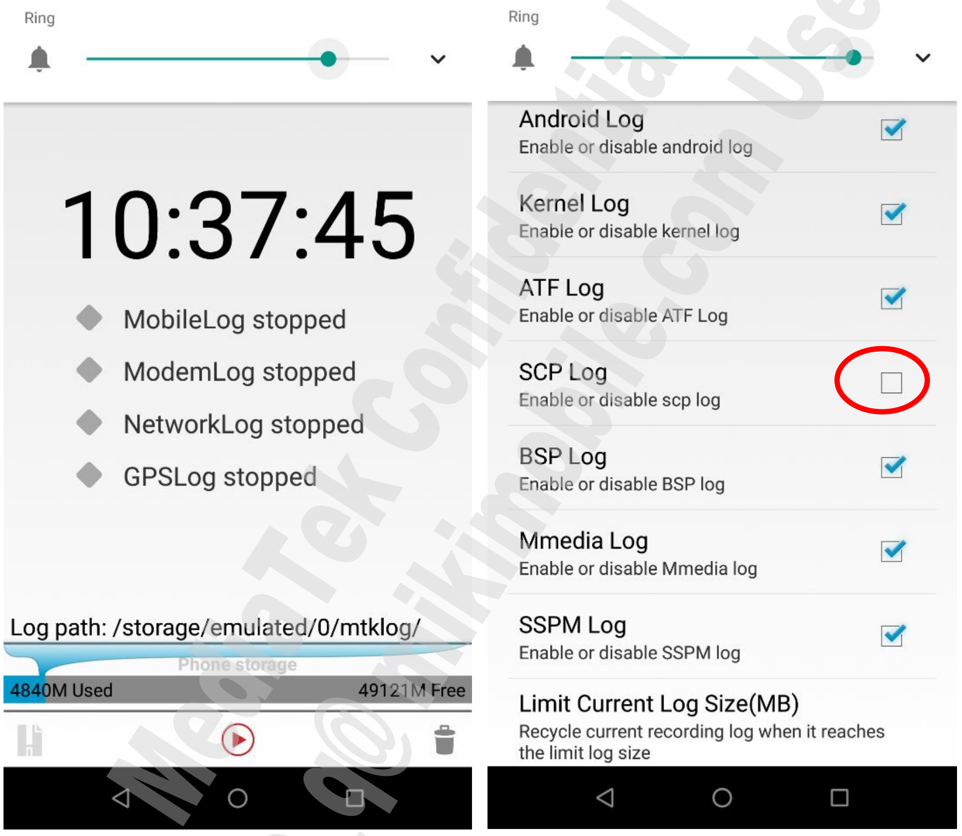

Tera Term: Serial port setup
Port: COM108
Baud rate: 921600
Data: 8 bit
Parity: none
Stop: 1 bit
Flow control: none
Transmit delay
0 msec/char 0 msec/line

Figure 8-2. UART setting

8.4 ADB Logcat

ADB logcat is able to output SCP log directly from ADB or UART console.

- Usage:
  1. Make sure SCP Log in MTK Logger is disabled (as shown in Figure 7-3.)
  2. Enter shell and enter command “echo 1 > /sys/class/misc/scp/scp_mobile_log”
  3. Enter command: “while true; do cat /dev/scp;done” and the log output directly, as shown in Figure 7-4.

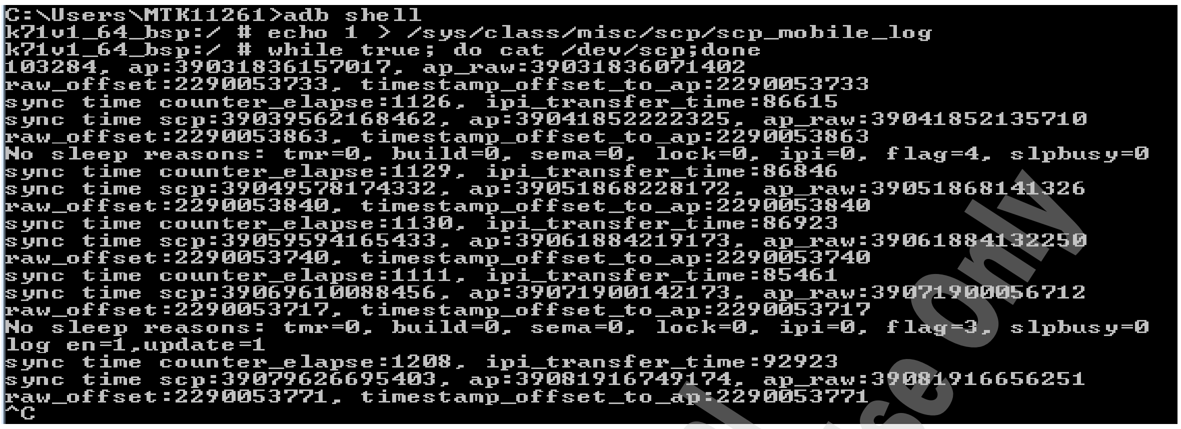


◆ MobileLog stopped
◆ ModemLog stopped
◆ NetworkLog stopped
◆ GPSLog stopped

Log path: /storage/emulated/0/mtklog/

Android Log Enable or disable android log	✓
Kernel Log Enable or disable kernel log	✓
ATF Log Enable or disable ATF Log	✓
SCP Log Enable or disable scp log	☐
BSP Log Enable or disable BSP log	✓
Mmedia Log Enable or disable Mmedia log	✓
SSPM Log Enable or disable SSPM log	✓
Limit Current Log Size(MB) Recycle current recording log when it reaches the limit log size	

Figure 8-3. Disable SCP Mobile log



```
C:\Users\MTK11261>adb shell
k71v1_64_bsp:/ # echo 1 > /sys/class/misc/scp/scp_mobile_log
k71v1_64_bsp:/ # while true; do cat /dev/scp;done
103284, ap:39031836157017, ap_raw:39031836071402
raw_offset:2290053733, timestamp_offset_to_ap:2290053733
sync time counter_elapse:1126, ipi_transfer_time:86615
sync time scp:39039562168462, ap:39041852222325, ap_raw:39041852135710
raw_offset:2290053863, timestamp_offset_to_ap:2290053863
No sleep reasons: tmr=0, build=0, sema=0, lock=0, ipi=0, flag=4, slpbusy=0
sync time counter_elapse:1129, ipi_transfer_time:86846
sync time scp:39049578174332, ap:39051868228172, ap_raw:39051868141326
raw_offset:2290053840, timestamp_offset_to_ap:2290053840
sync time counter_elapse:1130, ipi_transfer_time:86923
sync time scp:39059594165433, ap:39061884219173, ap_raw:39061884132250
raw_offset:2290053740, timestamp_offset_to_ap:2290053740
sync time counter_elapse:1111, ipi_transfer_time:85461
sync time scp:39069610088456, ap:39071900142173, ap_raw:39071900056712
raw_offset:2290053717, timestamp_offset_to_ap:2290053717
No sleep reasons: tmr=0, build=0, sema=0, lock=0, ipi=0, flag=3, slpbusy=0
log en=1,update=1
sync time counter_elapse:1208, ipi_transfer_time:92923
sync time scp:39079626695403, ap:39081916749174, ap_raw:39081916656251
raw_offset:2290053771, timestamp_offset_to_ap:2290053771
^C
```

Figure 8-4. ADB logcat output

8.5 Exception Log Analysis

When exception happens, SCP will print exception log automatically. Developers can get it from UART or mobile log.

- FAULT FETCH, FAULT LOAD, and FAULT STORE: Due to the wrong access of a MPU protect address.
  - FAULT FETCH: PC jumps to wrong address
    - the system register: mepc will show the incorrect target address.
  - FAULT LOAD, FAULT STORE: access a protected address
    - the system register mepc will show the wrong PC and mtval will show the incorrect address to be accessed.

```
[70.168](0) exception: CAUSE_FAULT_LOAD
[70.168](0)         exception pc: 0x00018b2c
[70.168](0)         fault load address: 0xffff1110
[70.168](0) Regs dump
[70.168](0) x0: 0xca0801e4 ra: 0x00018b24
[70.168](0) sp: 0x00036ec0 gp: 0x00000000
…
[70.168](0) t3: 0x00000000 t4: 0x00000011
[70.168](0) t5: 0x00000000 t6: 0x00000000
[70.168](0) pc: 0x00001454 mstatus: 0x05015800
[70.168](0) mepc: 0x00018b2c
[70.168](0) mcause: 0x00000005
[70.168](0) mtval: 0xffff1110
[70.168](0) T Buffer (10)
[70.168](0) 00 0x0000fdca:0x0002252c
[70.168](0) 01 0x0002252c:0x0000fdce
…
[70.169](0) 28 0x000108c0:0x0002368e
[70.169](0) 29 0x00023698:0x000108c4
[70.169](0) 30 0x000108ca:0x00024518
[70.169](0) 31 0x00024524:0x00018b24
[70.169](0) Code: 7545 85aa 0513 1105 <4108> 8593 1145 c188 9205
```

MISALIGNED FETCH, MISALIGNED LOAD, and MISALIGNED STORE: Usually happened when the integer pointer (4 byte aligned) access a 1 or 2 bytes aligned address.

- MISALIGNED FETCH: jump to misaligned address
  - the system register: mepc will show the wrong address
- MISALIGNED LOAD, MISALIGNED STORE: access a misaligned address
  - the system register mepc will show the wrong PC and mtval will show the access address

- ILLEGAL_INSTRUCTION: Due to the instruction cannot be decoded.
  For example:

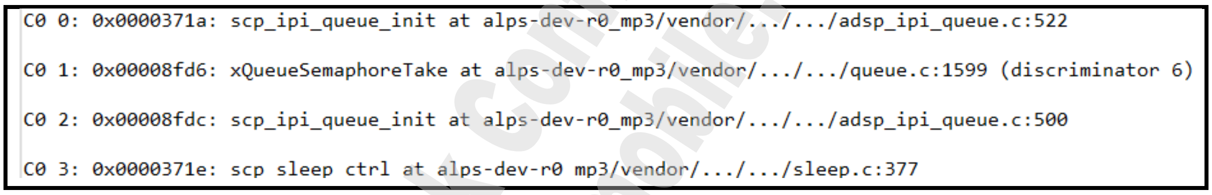
```
[2.643](0) exception: CAUSE_ILLEGAL_INSTRUCTION
[2.643](0) Regs dump
[2.644](0) x0: 0xa5a5a5a5 ra: 0xa5a5a5a5
[2.644](0) sp: 0xa5a5a5a5 gp: 0xa5a5a5a5
…
[2.651](0) t3: 0x00000000 t4: 0x00000000
[2.652](0) t5: 0x00000000 t6: 0x00000000
[2.652](0) pc: 0x00001454 mstatus: 0x05015880
[2.653](0) mepc: 0x00201720
[2.654](0) mcause: 0x00000005
[2.654](0) mtval: 0x2ff13748
[2.655](0) T Buffer (10)
[2.656](0) 00 0x0000371a:0x00008fd6
[2.656](0) 01 0x00008fdc:0x0000371e
…
[2.661](0) 28 0x0000f2a4:0x00002eec
[2.662](0) 29 0x00002eec:0x00006d52
[2.662](0) 30 0x00006d5c:0x00002ef0
[2.663](0) 31 0x0000f2a4:0x00003728
[2.664](0) Code: d194 5133 4521 873f <3748> 2ff1 a5a5 a5a5 a5a5
```

8.5.1 Trace Buffer

Trace buffer log can help us to analyze the latest function call that CPU have been executed. There are total 16 entries, and could be gotten form the bottom of the exception log.
For example:
The address could be translated with tools like lldb or addr2line

```
[2.655](0) T Buffer (10)
[2.656](0) 00 0x0000371a:0x00008fd6
[2.656](0) 01 0x00008fdc:0x0000371e
…
[2.661](0) 28 0x0000f2a4:0x00002eec
[2.662](0) 29 0x00002eec:0x00006d52  ← frame 2
[2.662](0) 30 0x00006d5c:0x00002ef0  ← frame 1
[2.663](0) 31 0x0000f2a4:0x00003728  ← latest frame: function turn_off_clk_sys_from_isr(0xf2a4)
                                                    jump to hostIntfHandleEvent (0x3728)
```

After execute SCP_COREDUMP file via *coredump_cmd.sh,* a readable text file would be generated, analysis.txt. The file list the combo of function call address, function name and file path of source file.



```
C0 0: 0x0000371a: scp_ipi_queue_init at alps-dev-r0_mp3/vendor/.../.../adsp_ipi_queue.c:522

C0 1: 0x00008fd6: xQueueSemaphoreTake at alps-dev-r0_mp3/vendor/.../.../queue.c:1599 (discriminator 6)

C0 2: 0x00008fdc: scp_ipi_queue_init at alps-dev-r0_mp3/vendor/.../.../adsp_ipi_queue.c:500

C0 3: 0x0000371e: scp sleep ctrl at alps-dev-r0 mp3/vendor/.../.../sleep.c:377
```

8.6 Core Dump

If AP is aware that SCP is not responding to IPI or SCP WDT event happens, the Core Dump flow will start automatically. Core Dump is a snapshot of the SCP memory and the processor registers like program counter, stack pointer, return address will be saved as well. And much of the system information like system registers, cache contents will be saved as much as possible. These information make it possible to restore the status of system before fault happened.
By default, LLDB is offered to support core dump debug. For LLDB detail, please reference https://lldb.llvm.org/.

- LLDB path

  LLDB could be found in alps prebuilts folder, path "prebuilts/clang/md32rv/linux-x86/lldbv2".
- Get Core dump

  The SCP core dump will be named as "SYS_SCP_DUMP", this will inside SCP EE DB (ex. db.00.EE.dbg).

  If not found, please check the EE DB, confirm the __exp_main.txt, the Exception type should be "scp".
- Start debug
- Type below command, and the initial log shown in Figure 7-8.

```
$ prebuilts/clang/md32rv/linux-x86/lldb_v2/coredump_cmd.sh rv55 tinysys-scp-RV55_A.elf SYS_SCP_DUMP
```

```
$ addr2line -e tinysys-scp-RV55_A.elf -af 0xf2a4
turn_off_clk_sys_from_isr
alps/vendor/mediatek/proprietary/tinysys/scp/drivers/RV55_A/MT6853/dvfs/src/dvfs.c:449
```

Note
LLDB version must compatible with SCP core

lldb version	Core type	Shell script command format
lldb	rv33 only	Coredump_cmd.sh tinysys-scp-RV33_A.elf SYS_SCP_DUMP
Lldb_v2	1 : rv33 2 : rv55 3 : rv55_v2 (mt6833,mt6877)	Coredump_cmd.sh <core type> <elf file> SYS_SCP_DUMP or Coredump_cmd.sh <core type> <elf file> SYS_SCP_DUMP core1

```
(lldb) command script import
/proj/mtk11261/misc/coredump/scp_coredump_cmdline/bin/coredump/freertos.py
(lldb) freertos

FreeRTOS Awareness is working ...

CPU 0 -- OS Tick: 70125
CPU 1 -- OS Tick: 0
OS is in normal state (neither in Critical Section nor in HW ISR)
TASK-0: IDLE@cpu0 ( TCB: 0x00042c60, State: Running, TCB Number: 8, Priority: 0)
 Stack (unit in word) size=1024, current used=124, max used=197
   * frame #0: 0x000231c6 tinysys-scp-RV55_A.elf`mrv_coredump(epc=-61168, regs=0xffff1000) at
exception.c:1055
    frame #1: 0x00003eb0 tinysys-scp-RV55_A.elf`atomicCmpXchg32bits(word=<unavailable>,
prevVal=238376, newVal=<unavailable>) at atomic.c:181
...
Flight Recorder is working ...
No Flight Recorder Support
(lldb)
```

If LLDB tool were launched unexpected,

1. Please confirm the file structure under *prebuilts/clang/md32rv/linux-x86/lldb_v2* was not been modified.
2. Please check the related log file "debug_prosim.log" and "debug_ocd.log". If the libprofile.so.x.x.x libraries were not found, add the following command to *"coredump_cmd.sh "* before PROSIM launch command.

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$MRV_PDK_PROSIM_HOME

#add above this command
$PROSIM_EXE $PROSIM_OPT
```

3. There are three tools (LLDB, Openocd, PROSIM) launched when used "coredump_cmd.sh" and tools communicate each other via TCP/IP protocol. If remote working with LLDB related tools, please confirm that the TCP/IP connect ports were not blocked by firewall.

8.6.1 LLDB Basic Commands

For the advanced developers or developers who are familiar with GDB, please refer to command map for the detail.

- Back trace: bt

```
 (lldb) bt
* thread #1, name = 'Hart0', stop reason = instruction step into
  * frame #0: 0x000231c6 tinysys-scp-RV55_A.elf`mrv_coredump(epc=-61168, regs=0xffff1000) at
exception.c:1055
    frame #1: 0x00003eb0 tinysys-scp-RV55_A.elf`atomicCmpXchg32bits(word=<unavailable>,
prevVal=238376, newVal=<unavailable>) at atomic.c:181
    frame #2: 0x0000befa tinysys-scp-RV55_A.elf`osSetCurrentTid [inlined]
osSetCurrentTask(task=0x0003a328) at seos.c:153
    frame #3: 0x0000bee6 tinysys-scp-RV55_A.elf`osSetCurrentTid(tid=<unavailable>) at seos.c:212
    frame #4: 0x0000d872 tinysys-scp-RV55_A.elf`timFireAsNeededAndUpdateAlarms at timer.c:145
    frame #5: 0x0000d608 tinysys-scp-RV55_A.elf`timTimerSetEx(length=<unavailable>,
jitterPpm=<unavailable>, driftPpm=<unavailable>, info=<unavailable>, data=<unavailable>,
oneShot=<unavailable>) at timer.c:179
    frame #6: 0x0000cf4e tinysys-scp-RV55_A.elf`osDefer(callback=<unavailable>,
cookie=<unavailable>, urgent=<unavailable>) at seos.c:1184
    frame #7: 0x000467f4 tinysys-scp-RV55_A.elf`ucHeap + 43796
```

- Dump OS tasks information: freertos

```
(lldb) freertos
Find 18 tasks
CPU 0 -- OS Tick: 70125
CPU 1 -- OS Tick: 0
Coredump cpu: 0
OS is in normal state (neither in Critical Section nor in HW ISR)


TASK-0: IDLE@cpu0 ( TCB: 0x00042c60, State: Running, TCB Number: 8, Priority: 0)
 Stack (unit in word) size=1024, current used=124, max used=197
  * frame #0: 0x000231c6 tinysys-scp-RV55_A.elf`mrv_coredump(epc=-61168, regs=0xffff1000) at
exception.c:1055
    frame #1: 0x00003eb0 tinysys-scp-RV55_A.elf`atomicCmpXchg32bits(word=<unavailable>,
prevVal=238376, newVal=<unavailable>) at atomic.c:181
    frame #2: 0x0000befa tinysys-scp-RV55_A.elf`osSetCurrentTid [inlined]
osSetCurrentTask(task=0x0003a328) at seos.c:153
    frame #3: 0x0000bee6 tinysys-scp-RV55_A.elf`osSetCurrentTid(tid=<unavailable>) at seos.c:212
    frame #4: 0x0000d872 tinysys-scp-RV55_A.elf`timFireAsNeededAndUpdateAlarms at timer.c:145
    frame #5: 0x0000d608 tinysys-scp-RV55_A.elf`timTimerSetEx(length=<unavailable>,
jitterPpm=<unavailable>, driftPpm=<unavailable>, info=<unavailable>, data=<unavailable>,
oneShot=<unavailable>) at timer.c:179
    frame #6: 0x0000cf4e tinysys-scp-RV55_A.elf`osDefer(callback=<unavailable>,
cookie=<unavailable>, urgent=<unavailable>) at seos.c:1184
    frame #7: 0x000467f4 tinysys-scp-RV55_A.elf`ucHeap + 43796

...

TASK-17: CHRE@cpu0 ( TCB: 0x0003f910, State: Suspended, TCB Number: 4, Priority: 4)
 Stack (unit in word) size=1024, current used=132, max used=364
  * frame #0: 0x00013bc4 tinysys-scp-RV55_A.elf`vTaskExitCritical at tasks_smp.c:4287
    frame #1: 0x00007aee tinysys-scp-RV55_A.elf`evtQueueDequeue(q=<unavailable>,
evtTypeP=<unavailable>, evtDataP=<unavailable>, evtFreeDataP=<unavailable>,
sleepIfNone=<unavailable>) at eventQ.c:171
    frame #2: 0x0000cbb8 tinysys-scp-RV55_A.elf`osMainDequeueLoop at seos.c:1052
    frame #3: 0x0000cdfc tinysys-scp-RV55_A.elf`osMain at seos.c:1093
```

- Print variable and address: p $(variable name)

```
(lldb) p mTask
(alsPsTask) $0 = {
 id = 259
 handle = ([0] = 16973827, [1] = 16973828, [2] = 16973829)
 prevRtcTime = 0
 mDataSlab = 0x0003c340
 dataEvt = 0x00000000
 alsLastSample = 0
 psLastSample = 1
 bcRecv = {
  [0] = {
   list = {
    prev = 0x00048290
    next = 0x002109e0
   }
   sensor_type = '\f'
   receive_event = 0x00206a34 (tinysys-scp-RV33_A.elf`alsPsReceiveEvent at alsps.c:908)
}
```

- Dump memory: x/FMT address

```
(lldb) x/32xw 0x1ef08
0x0001ef08: 0x003a9593 0x093d3533 0x10063667 0x00bb0ab3
0x0001ef18: 0x01340633 0x016ab5b3 0x00b60c33 0x008c0463
0x0001ef28: 0x008c35b3 0x01248633 0x0433955e 0x35b300b6
0x0001ef38: 0x36330096 0x952e00c4 0x00c50bb3 0xd5334d32
0x0001ef48: 0xc11d094b 0x001ad593 0x001af513 0xfe1c05b3
0x0001ef58: 0x001c5c13 0xfe140c33 0x84338005 0xdb93fe1b
0x0001ef68: 0xeab3001b 0x088500a5 0x652144a2 0x09f4a4b3
0x0001ef78: 0xcf63157d 0x053700a8 0x24237fff 0x8d451001
```

- Read cpu register: register read $REG

```
(lldb) register read pc
   pc = 0x00000038  tinysys-scp-RV33_A.elf`__divtf3 + 54 at divtf3.c
(lldb) register read
general:
     x0 = 0x00000000  tinysys-scp-RV33_A.elf`vPortInitialiseBlocks at heap_2.c:239
     x1 = 0x0000a32c  tinysys-scp-RV33_A.elf`stackDump + 614 at scp_it.c:128
...
    x27 = 0x00000000  tinysys-scp-RV33_A.elf`vPortInitialiseBlocks at heap_2.c:239
    x28 = 0x0000006d  tinysys-scp-RV33_A.elf`__divtf3 + 107 at divtf3.c:30
(lldb) register read mepc
  mepc = 0x0000a32c  tinysys-scp-RV33_A.elf`stackDump + 614 at scp_it.c:128
```

9 Appendix

9.1 Common Issues and How to Fix

9.1.1 Malloc Faill

Because the SRAM size is limited and small, malloc pool size is tailored to the feature, malloc fail is common when a new feature in involve without enlarge the pool size.

Warning

When malloc fail is happened, a fail message will be showed.

"

[2.232](0) malloc fail

[2.232](0) [ASSERT] task: CHRE

"

This message shows which task malloc fail and then will follow the coredump log

- How to Fix

1. Modify the scp/project/RV55_A/MT6853/platform/platform.mk, enlarge the heap size, ex 80*1024

```
ifeq ($(CFG_CHRE_SUPPORT),yes)
$(eval TOTAL_HEAP_SIZE=$(shell echo $$(($(TOTAL_HEAP_SIZE) + (80 * 1024))))) ← change 40 to 80
endif
```

2. Modify the scp/drivers/common/scpctl/scp_scpctl.c, force enable Task monitor.

```
void scpctl_init(void)
{
...
        else {  /* monitor task is in suspened state */
                scpctl.stat = SCPCTL_STAT_INACTIVE;
                scpctl.op = SCPCTL_OP_INACTIVE;
                //vTaskSuspend(xMonitorTask[id]);  ← about line: 244, mark it
        }
...
}
```

3. Check the remain heap size and compute the require heap size

```
[210.009](0) Heap:free/total:40946/108544
```

4. Correct the heap size, The remain heap size is **40946, 40946/1024 = 39.98,** the minimum heap size is 41, and set to 43 or 44 is more safe. Please remember restore scp/drivers/common/scpctl/scp_scpctl.c change after issue fixed.

```
ifeq ($(CFG_CHRE_SUPPORT),yes)
$(eval TOTAL_HEAP_SIZE=$(shell echo $$(($(TOTAL_HEAP_SIZE) + (43 * 1024)))))
endif
```

9.1.2 Unaligned Access

The SCP MDSP-RV33 processor does not have hardware to handle unaligned access, the implementation is done by software. When unaligned access happens, the processor will raise exceptions and do software solution in handler. Therefore, the performance cost is significantly.

Warning
When unaligned access is happened, a warning message will be showed.

"Warning: MISALIGNED LOAD, pc:0x00001234, addr:0x00042232"

This message shows the PC with problems and address the processor wants to access. The system may be busy to print this message when unaligned access happens continuously and may cause timeout assert.

- Example and How to Fix

Most of unaligned access is due to the packed qualifier applied to structure. This could be avoid by remove it.
Example:

```
struct pack_struct {
    unit32_t size;
    uint32_t crc;
    uint8_t type;
} __attribute__((packed));
```

When a structure array is declared, for ex pack_struct st_array[10], the access of structure member will be unaligned.

Another case is accessing character array with integer pointer. This could be avoided by adding __attribute__ ((aligned (4))). The compiler does not guarantee &char_array[0] is 4 byte alignment.
Example:

```
uint8_t char_array[64];
test_value = *(uint32_t *)(&char_array[0])
```

9.2 Code Size Limitation

The total SRAM size of MT6853 is 768KB, the actual SCP SRAM size cloud check the symbol _end.

The left SRAM size will be:

MT6853: 0xc0000 - _end

In MT6853, we can set SRAM region LENGTH to 768KB in the project/RV55_A/MT6853 /platform/link.ld.c
Example: "sram : ORIGIN = 0x00000000, LENGTH = 0x000c0000".
This will cause build fail when size exceeded 768KB.

The code size check tool will run every build and output details. Check the **Error! Reference source not found.**, vertical is the item name and horizontal is code size of each item. For example the total size of CHRE is 62985(Sum) bytes and the .text part takes 35598 bytes of total size.

```
                     . = ALIGN ( ...sync     .text      Sum
    C-lib            0    ...        0         0
    CHRE             0    ...                  35598   62985
    DSP              0    ...        0         0
    DVFS             0    ...        7574      8606
    Heap             0    ...        334       82275
  Peripheral         0    ...        18064             26981
   Platform        256    ...        66300             313481
    RTOS             0    ...        13776             14536
   Sensor            0    ...        0         0
    VOW              0    ...        0         0
```

9.2.1 Code size tool usage

- The tool: **memoryReport.py** is a script which is used to limit code size at the build time. If code size over your settings, it will cause build errors.
  - **Path**
    vendor/mediatek/proprietary/tinysys/common/tools/memoryReport.py
- Setting configuration file at following path
  - **Path**
    vendor/mediatek/proprietary/tinysys/scp/project/RV55_A/MT6853/platform/Setting.ini

```
Configuration file format (setting.ini)
----------------------------------------------------------------
[TinySys-SCP]
$File_Name: $Main_feature: $Sub_feature
[SCP-mt6853]
$Main_feature : Max_code_size
$Sub_feature : Max_code_size
----------------------------------------------------------------
* File_name: Full file path or Partial file path (Ex:middleware/contexthub/perf)
* Main feature, (Ex: Sensor, Audio), the main feature that this file belong to
* Sub feature, (Ex: gyro, pedometer), the sub feature that this file belong to
* Main_feature/Sub_feature (after SCP-mt6873):
   - Main or Sub feature maximum size limit
```

- Memory check fail
  - Reference for example

```
SCP: I2C(3958>110) is out of memory limitation
SCP: SPI(6316>1100) is out of memory limitation
make: *** [tinysys_out/RV55_A/scp/tinysys-scp-RV55_A.elf] Error 13
```

9.3 Scp_Region_Info Structure

The struct scp_region_info_st is pointer to a fixed address at SCP SRAM. It's used to pass parameter from bootloader to SCP before the ipi ready. Remember to sync bootloader(lk), kernel(kernel-4.14) and scp repos code if new member is add. SCP may boot fail if structure is not sync between repos.

```
LK header:
        vendor/mediatek/proprietary/bootable/bootloader/lk/platform/mt6853/include/platfor
        m/mt_scp.h:166
Kernel header:
        kernel-4.14/drivers/misc/mediatek/scp/mt6853/scp_helper.h:138
SCP header:
        vendor/mediatek/proprietary/tinysys/scp/project/RV55_A/mt6853/platform/inc/main.h:47
SCP boot:
        vendor/mediatek/proprietary/tinysys/scp/project/RV55_A/mt6853/platform/boot55.S
```

scp_region_info also defined in boot55.S.

For example, if a new member called test ID is added in structure scp_region_info, the default value must be added in boot55.S.

```
scp_region_info:
.long     0x00000000      /* 0x04 ap_loader_start */
.long     0x00000000      /* 0x08 ap_loader_size */
...
.long     0x00100000      /* 0x34 regdump size */
.long     0x00000000      /* 0x38 param start address */
.long     0x00000000      /* 0x3c ap_params_start  */
.long     0x12345678      /* 0x40 a test ID */
```

Access scp_region_info_st in SCP side must assign it to 0xBFE00004, for example

```
void scpctl_init(void)
{
    int ret  = 0;
#ifdef CFG_NULLPTR_TRAP
    struct scp_region_info_st region_info = *(struct scp_region_info_st *)(0xBFE00004);
#endif
```

9.4 SCP Recovery

The Linux SCP driver will do SCP recovery when SCP crashes or does not respond. After the recovery process, the SCP will be back to normal. But we should note several things.

- SCP recovery will clear SCP SRAM/DRAM contents, reset RV55 processor and execute boot flow again. (clear all program text, bss, and data section)
- SCP recovery does reset only processor **but not** peripherals (ex. sensors, i2c module and other devices…), so peripheral drivers should do self-reset at initial stage if necessary.
- Developers of Linux SCP driver API **MUST** follow "**Reset notify flow**" in 9.4.2 and ensure there are no any communication between Linux driver and SCP during the recovery.
- The Linux SCP driver will re-initial when SCP reboot. Drivers that relates to SCP driver **MUST** ensure the reboot flow will not affect its functions.

9.4.1 Recovery behavior

The recovery starts when SCP exception or no response for a while. The SCP kernel driver and SCP will enter out of service state. During this period, SCP kernel driver will reset SCP and send the SCP_EVENT_STOP to all drivers that have registered notify chain. After SCP back to normal, SCP kernel driver will send SCP_EVENT_READY to all the driver.

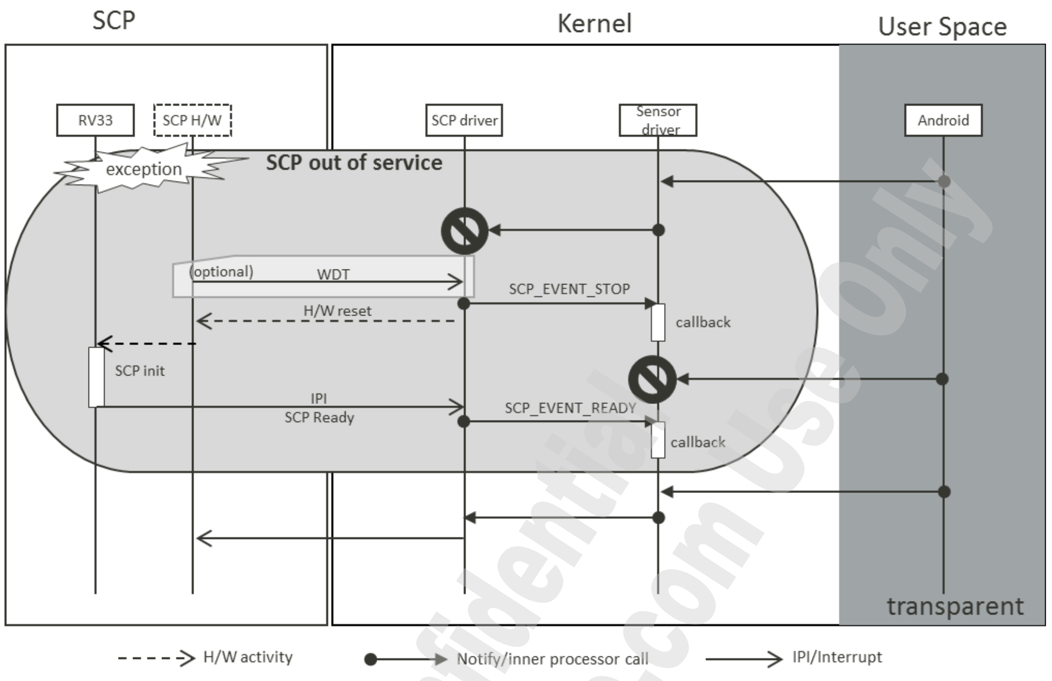


Figure 9-1. SCP recovery behavior

9.4.2 Recovery Notify Flow

Developers of the Linux SCP driver **MUST** follow this SCP EVENT notify chain, to ensure SCP recovery effective.

- **API**: receives notifications and invokes registered callback functions

void scp_A_register_notify(struct notifier_block *nb)
description
To register callback function of notify chain
parameters
nb: callback function of notify chain
callback parameters
SCP_EVENT_READY: when SCP initial done, start tasks corresponding to SCP
SCP_EVENT_STOP: when SCP is going to be reset, stop tasks corresponding to SCP

- **Note**

  All callback functions may be called multiple times and **MUST NOT** be blocked

- **Example**

  1. Please include <linux/notifier.h>, <mach/scp_helper.h>

2. Call scp_register_notify () with argument: SCP_EVENT_READY or SCP_EVENT_STOP

```
static void task_start(void) {
  // start tasks
}
static void task_stop(void) {
  // stop tasks
}
static int app_event(struct notifier_block *this, unsigned long event, void *ptr) {
    switch (event) {
        case SCP_EVENT_READY:
            task_start();
            break;
        case SCP_EVENT_STOP:
            task_stop();
            break;
    }
    return NOTIFY_DONE;
}
static struct notifier_block app_notifier = {
    .notifier_call = app_event,
};
static int __init scp_app_init(void) {
    if (scp_is_ready()) {
        task_start();
    }
    scp_register_notify(&app_notifier);
}
```

WARNING

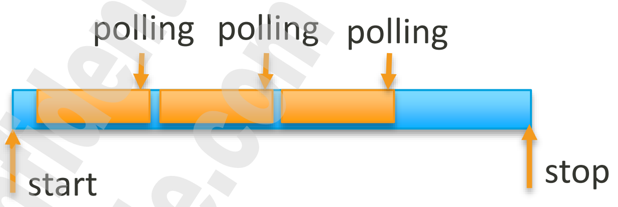
1. **MUST** register notification chain because the scp_ipi_send() may return error during recovery.
2. An error handling flow **MUST** be applied:
   - Stop calling scp_ipi_send() right after receiving SCP_EVENT_READY
   - Resume after receiving SCP_EVENT_READY.

9.5 SCP Performance Budget Center (PBFR)

9.5.1 Introduction

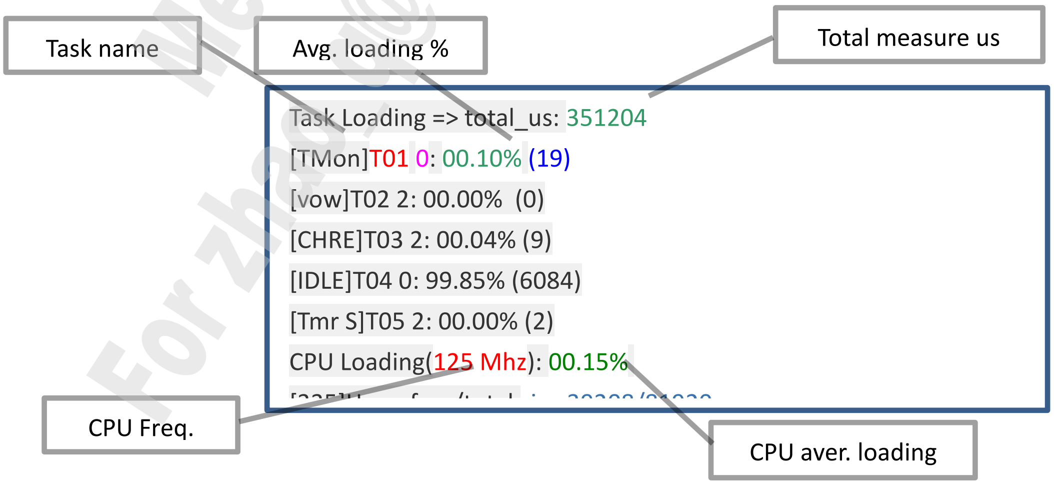
PBFR(Performance Budget Center): Support Task level profiling, ex: each task loading, cache miss and whole system loading information.

9.5.2 Options

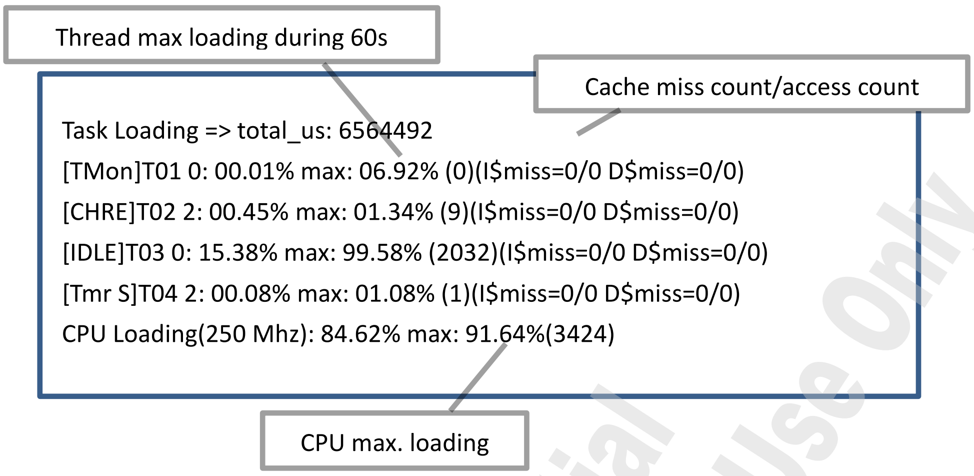
Compiler options	description
CFG_PBFR_SUPPORT = yes	Main function control
PBFR_SUPPORT_POLLING = yes	Support save the max loading of each task by polling 
#define POLLING_TIME 100	Require PBFR_SUPPORT_POLLING = yes Polling thread active every 100 ms
PBFR_SUPPORT_CACHE_COUNT =yes	Support cache miss information

9.5.3 Log interpretation

PBFR automatic report information log every 60s, you can manual disable it on your demand.
Task relative information: Task name, Avg. Loading %(during 60 s), max loading (during 60 s)
System information: Total measure us, CPU Frequence, CPU average loading,



```
Task Loading => total_us: 351204
[TMon]T01 0: 00.10% (19)
[vow]T02 2: 00.00%  (0)
[CHRE]T03 2: 00.04% (9)
[IDLE]T04 0: 99.85% (6084)
[Tmr S]T05 2: 00.00% (2)
CPU Loading(125 Mhz): 00.15%
```



```
Task Loading => total_us: 6564492
[TMon]T01 0: 00.01% max: 06.92% (0)(I$miss=0/0 D$miss=0/0)
[CHRE]T02 2: 00.45% max: 01.34% (9)(I$miss=0/0 D$miss=0/0)
[IDLE]T03 0: 15.38% max: 99.58% (2032)(I$miss=0/0 D$miss=0/0)
[Tmr S]T04 2: 00.08% max: 01.08% (1)(I$miss=0/0 D$miss=0/0)
CPU Loading(250 Mhz): 84.62% max: 91.64%(3424)
```

9.5.4 Manual usage APIs

For customize demand, ex: we want measure it during audio decode routine only, or start measure depend on special event.

Remove origiral pbfr_start_loadinfo(PERIOD_MOD) and pbfr_report_loadinfo(PERIOD_MOD) in code base and add new APIs to the place you need.

- **API**

int pbfr_start_loadinfo(int mode)

Description

If the system changed the timestamp (ex: time sync) after this function, loading info may be wrong.

It starts to save loading of each tasks

Parameters

PERIOD_MOD: information reset every perioid

ACCCMULATE_MODE: report accumulate information without resetname: a string to recognize IPI handler

Return values

Always 0

int pbfr_report_loadinfo(int mode)

Description

usually be called every 60 s.

report the log info from the previous report to now.

It can be called before or after pbfr_stop_loading.

Parameters

PERIOD_MOD: information reset every perioid

ACCCMULATE_MODE: report accumulate information without resetname: a string to recognize IPI handler

Return values

0 : Success

-1 : Error with pbfr status

int pbfr_ stop _loadinfo(void)

Description

If the system changed the timestamp (ex: time sync) before this function, loading info may be wrong.

It stops to save loading of each tasks.

Parameters

PERIOD_MOD: information reset every perioid

ACCCMULATE_MODE: report accumulate information without resetname: a string to recognize IPI handler

Return values

Always 0

10 Q&A List from Users

10.1 How to Enlarge DRAM region code

By default, we reserve 1MB DRAM is for store DRAM code. In extreme case, if more space is need to store code, here is the modify method.

1. Modify LK, enlarge 0x100000

```
vendor/mediatek/proprietary/bootable/bootloader/lk/platform/mt6853/include/mt_scp.h
#ifdef MTK_MINIMUM_SCP_DRAM_SIZE
#define SCP_DRAM_IMG_SIZE      0x080000    // 0.5MB dram image
#else
#define SCP_DRAM_IMG_SIZE      0x100000    // 1.0MB dram image
#endif
```

2. Modify SCP, enlarge 0x00100000

```
vendor/mediatek/proprietary/tinysys/scp/project/RV55_A/mt6853/platform/link.ld.c
MEMORY {
  … … …
  dram      :  ORIGIN = 0x00200000, LENGTH = 0x00100000
  … … …
```

10.2 How to Share Information by DRAM between AP and SCP.

Overview of sharing the information by DRAM between AP and SCP, shown as **Figure 10-1**

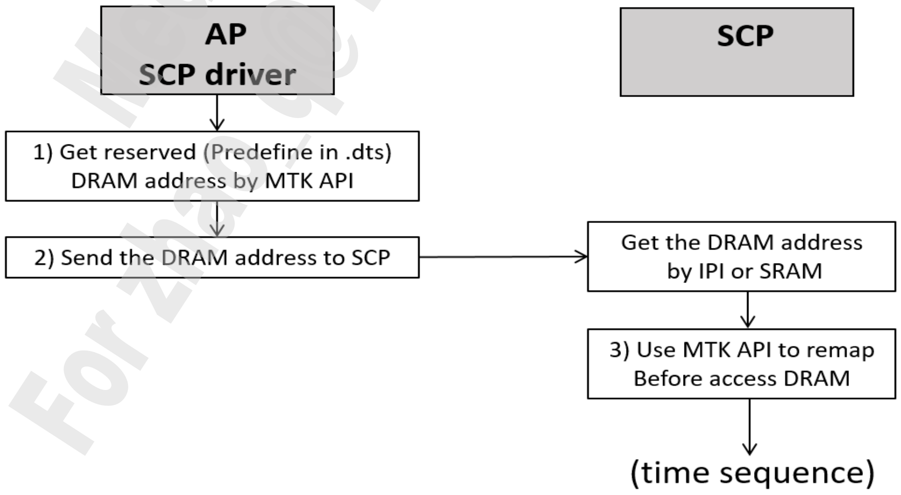
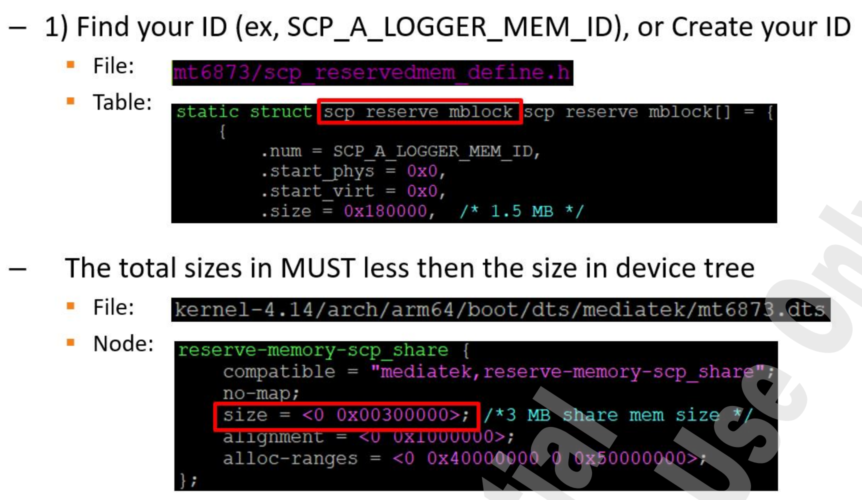

AP
SCP driver
SCP
1) Get reserved (Predefine in .dts) DRAM address by MTK API
2) Send the DRAM address to SCP
Get the DRAM address by IPI or SRAM
3) Use MTK API to remap Before access DRAM
(time sequence)

Figure 10-2. Sharing information by DRAM between AP and SCP

Step 1: In AP side (Linux kernel driver), define feature ID before use by MTK API, shown as Figure ***10-3***.



– 1) Find your ID (ex, SCP_A_LOGGER_MEM_ID), or Create your ID
- File: `mt6873/scp_reservedmem_define.h`
- Table:

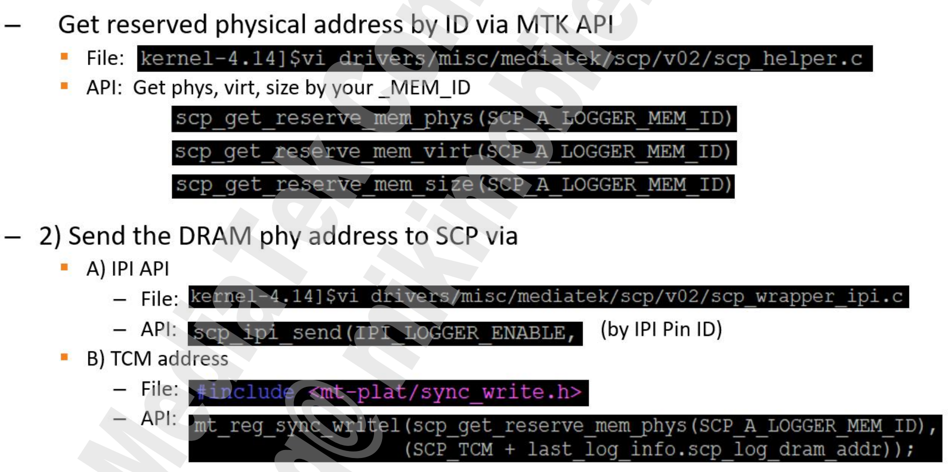
```
static struct scp_reserve_mblock scp_reserve_mblock[] = {
	{
		.num = SCP_A_LOGGER_MEM_ID,
		.start_phys = 0x0,
		.start_virt = 0x0,
		.size = 0x180000,  /* 1.5 MB */
```

– The total sizes in MUST less then the size in device tree
- File: `kernel-4.14/arch/arm64/boot/dts/mediatek/mt6873.dts`
- Node:

```
reserve-memory-scp_share {
	compatible = "mediatek,reserve-memory-scp_share";
	no-map;
	size = <0 0x00300000>; /*3 MB share mem size */
	alignment = <0 0x1000000>;
	alloc-ranges = <0 0x40000000 0 0x50000000>;
};
```

Figure 10-4. Define size of the feature ID

Step 2: Get the physical address and size of the reserved memory by feature ID via MTK API, shown as Figure 9-3.



– Get reserved physical address by ID via MTK API
- File: `kernel-4.14]$vi drivers/misc/mediatek/scp/v02/scp_helper.c`
- API: Get phys, virt, size by your _MEM_ID

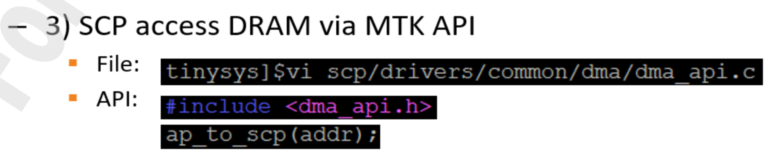
```
scp_get_reserve_mem_phys(SCP_A_LOGGER_MEM_ID)
scp_get_reserve_mem_virt(SCP_A_LOGGER_MEM_ID)
scp_get_reserve_mem_size(SCP_A_LOGGER_MEM_ID)
```

– 2) Send the DRAM phy address to SCP via
- A) IPI API
  - File: `kernel-4.14]$vi drivers/misc/mediatek/scp/v02/scp_wrapper_ipi.c`
  - API: `scp_ipi_send(IPI_LOGGER_ENABLE,` (by IPI Pin ID)
- B) TCM address
  - File: `#include <mt-plat/sync_write.h>`
  - API:

```
mt_reg_sync_writel(scp_get_reserve_mem_phys(SCP_A_LOGGER_MEM_ID),
			(SCP_TCM + last_log_info.scp_log_dram_addr));
```

Figure 10-5. Get Physical address and size of the reserved by feature ID

Step 3: In SCP side, transfer the address to SCP view by MTK API- ap_to_scp() before use the address, shown as Figure ***10-6***. Due to the reserved memory is DRAM, user **MUST** also request the DRAM resource (Refer 5.4 DRAM resource request)



– 3) SCP access DRAM via MTK API
- File: `tinysys]$vi scp/drivers/common/dma/dma_api.c`
- API:

```
#include <dma_api.h>
ap_to_scp(addr);
```

Figure 10-7. Use MTK API to transfer address from AP view

Between Step 2 and 3, users can use the IPI (Refer 6.4 Inter Processor Interrupt) or predefined Share TCM memory (ex, SCP_TCP + last_log_info.scp_log_dram_addr) to share the address between AP and SCP.

Exhibit 1 Terms and Conditions