MT793X IoT SDK for
System Log Developer's Guide

Version:	1.0
Release date:	2021-03-24

Use of this document and any information contained therein is subject to the terms and conditions set forth in Exhibit 1. This document is subject to change without notice.

Version History

Version	Date	Description
0.1	2020-11-11	Initial draft
1.0	2021-03-24	Correct and remove unused information

Table of Contents

List of Figures

List of Tables

1 Introduction

Mediatek IoT development platform provides comprehensive support to develop and connect wearables and IoT applications and devices. System Log facilitates debugging during software development and provides a convenient logging system that supports log filtering and multitasking. Log filtering allows you to focus on the specific parts of the logs. Multitasking support ensures the logging is properly managed when multiple tasks call the log API simultaneously. For more details about the System Log API usage, refer to the header file at `<sdk_root>/kernel/service/inc/syslog.h`.

1.1 System Log Architecture

The System Log architecture is shown in Figure 1. The target prints logs using the UART interface. The logs are then displayed on the host side by calling a terminal program such as TeraTerm and Putty. Installing and setting up TeraTerm is described in "Terminal application setup" and "Serial port settings" sections of the "Mediatek IoT SDK Get Started Guide" in `<sdk_root>/doc` folder. The target side of Figure 1 is detailed in Section 1.4 Multitasking Support. The host side UART configuration is described in Chapter 3 Host UART Configuration.

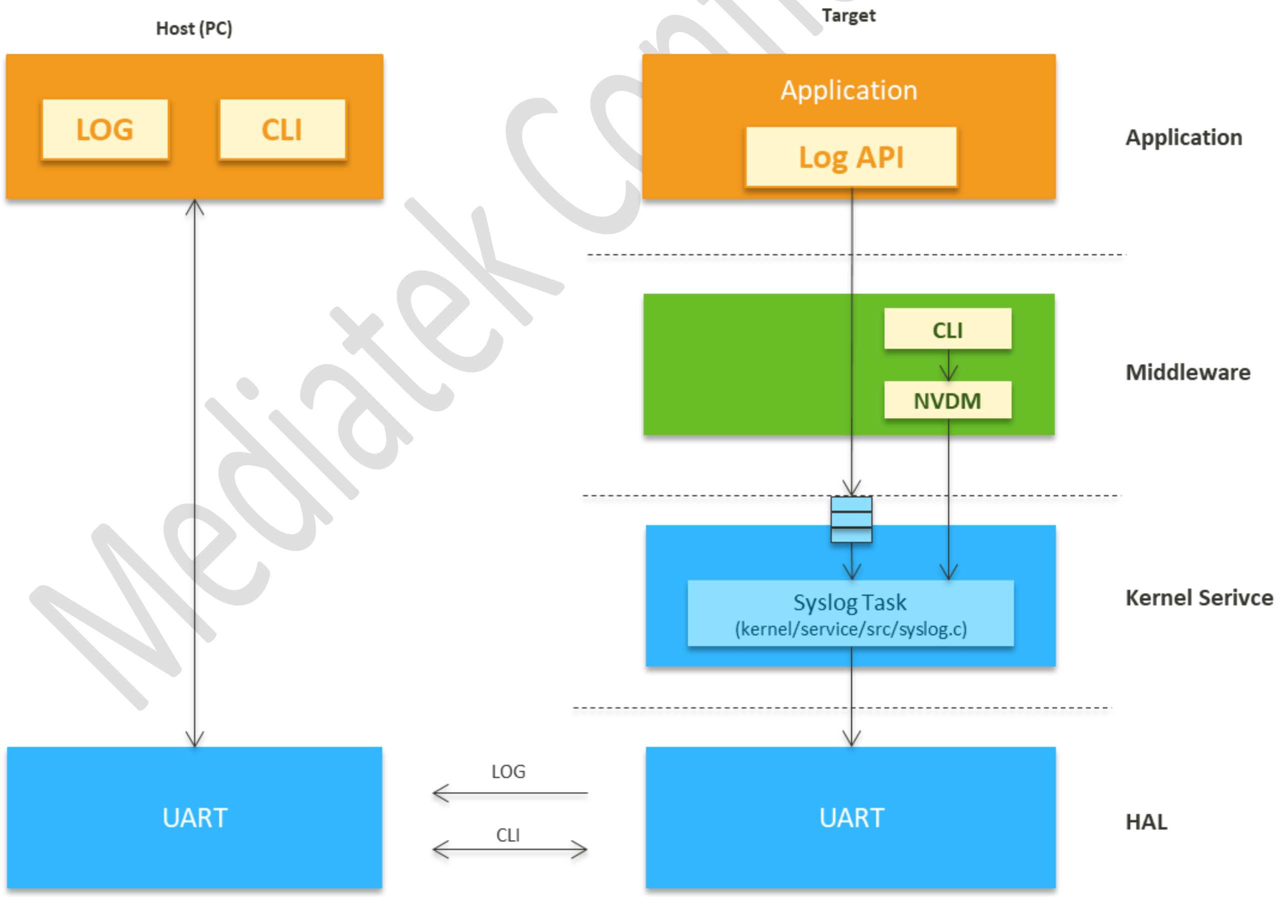


Figure 1. System Log architecture

1.2 Log Filtering

The logs are grouped by modules. Each module is associated with a log control block to specify whether the log messages within that module will be printed or not. The log control block contains the current log level setting for each module. The caller of the log API specifies the log level, the module and the log message. The System Log then performs filtering based on the log level and the current settings in the module's log control block. There are three configurable log levels: INFO, WARNING and ERROR ordered by ascending log level. The log message is printed only if the log level of the current log message is greater than or equal to the log level specified in the log control block of the module. A common module is defined in `syslog.c` for the type of logs not assigned to any particular module. You can update the log setting of each module dynamically through the CLI commands for the MT7933 HDK. The update takes effect immediately after the commands are issued and log configuration settings are stored in the NVDM.

1.3 Log Setting Commands

You can create a log module with the function `log_create_module()` (see Section 2.1.4 log_create_module()). The system log is enabled by default with the debug level specified in the macro as an input to the function `log_create_module()`.
The log settings are stored in the NVDM. Once the system reboots, the log settings read from the NVDM override the default settings in the image. If there are no log settings in the NVDM, the default log settings in the image are written to the NVDM. The log settings can be updated through CLI commands and then the updated settings are stored in the NVDM.
The log control blocks are defined in separate modules. For the convenience of log filtering routine implementation, aggregate the log control blocks (see Section 1.2 Log Filtering) for all modules of a project into a table and then apply the CLI commands. More details can be found in Sections 2.1.1 `log_init()`, 2.1.2 LOG_CONTROL_BLOCK_DECLARE() and 2.1.3 LOG_CONTROL_BLOCK_SYMBOL.
An example description to provide the log setting commands is shown below.

1) Open the main source file of the reference project from `<sdk_root>/project/mt7933_hdk/<project_name>/src/sys_init.c`.
2) Identify the `LOG_CONTROL_BLOCK_DECLARE(module)` which declares the external reference to the log control block of the given module.

```
LOG_CONTROL_BLOCK_DECLARE(main);
LOG_CONTROL_BLOCK_DECLARE(common);
LOG_CONTROL_BLOCK_DECLARE(hal);
```

3) The table `syslog_control_blocks[]` is a reference to the log control block of each module.

```
log_control_block_t *syslog_control_blocks[] = {
    &LOG_CONTROL_BLOCK_SYMBOL(main),
    &LOG_CONTROL_BLOCK_SYMBOL(common),
    &LOG_CONTROL_BLOCK_SYMBOL(hal),

    NULL
};
```

4) The user-defined callback functions `syslog_config_save()` and `syslog_config_load()` save and load the log configuration settings of the system.

```
static void syslog_config_save(const syslog_config_t *config)
{
    nvdm_status_t status;
    char *syslog_filter_buf;

    syslog_filter_buf = (char*)pvPortMalloc(SYSLOG_FILTER_LEN);
    configASSERT(syslog_filter_buf != NULL);
    syslog_convert_filter_val2str((const log_control_block_t **)config-
>filters, syslog_filter_buf);
    status = nvdm_write_data_item("common",
                                  "syslog_filters",
                                  NVDM_DATA_ITEM_TYPE_STRING,
                                  (const uint8_t *)syslog_filter_buf,
                                  strlen(syslog_filter_buf));
    vPortFree(syslog_filter_buf);
    LOG_I(common, "syslog config save, status=%d", status);
}
static uint32_t syslog_config_load(syslog_config_t *config)
{
    uint32_t sz = SYSLOG_FILTER_LEN;
    char *syslog_filter_buf;

    syslog_filter_buf = (char*)pvPortMalloc(SYSLOG_FILTER_LEN);
    configASSERT(syslog_filter_buf != NULL);
    if           (nvdm_read_data_item("common",          "syslog_filters",
(uint8_t*)syslog_filter_buf, &sz) == NVDM_STATUS_OK) {
        syslog_convert_filter_str2val(config->filters,
syslog_filter_buf);
    } else {
        /* popuplate the syslog nvdm with the image setting */
        syslog_config_save(config);
    }
    vPortFree(syslog_filter_buf);
```

```
    return 0;
}
```

5) The log configuration settings are stored in NVDM. Call the function `nvdm_init()` before calling the function `log_init()`. `syslog_config_save()`, `syslog_config_load()` and `syslog_control_blocks[]` are the input parameters for the `log_init()` function.

```
int system_init(void)
{
    /* Configure the hardware ready to run the test. */
    prvSetupHardware();
    ...

#ifdef MTK_NVDM_ENABLE
    /* nvdm init */
    nvdm_init();

    /* nvdm module init */
    nvdm_module_init();
#endif
    ...
#ifdef MTK_NVDM_ENABLE
    log_init(syslog_config_save,                    syslog_config_load,
syslog_control_blocks);
#else
    log_init(NULL, NULL, syslog_control_blocks);
#endif
    ...

}
```

1.3.1 CLI Commands

The CLI commands are applicable to the MT7933 HDK only, as shown in the following table.

Table 1. CLI commands for the MT7933 HDK

Function	Command
Query for the usage	`$log set`
Query for the current setting	`$log`

Update the settings of a module	`$log set <module> <log_switch> <print_level>`
	`<module>: module name`
	`<log_swith>: on, off`
	`<print_level>: info, warning, error`

Query for the usage of format setting	`$log fmt`
Query for the log header format for the specified module	`$log fmt <module>` `<module>: module name`

Set the log header format for the specified module	`$log fmt <module> <format>`
	`<module>: module name`
	`<format >: log header format`

Switch on/off saved log to flash feature	`$log switch <syslog_switch_from_uart_to_flash>` `<syslog_switch_from_uart_to_flash>:on, off`

1.3.1.1 CLI Command Example

In this example, the following log modules are presented — main, common, HAL. The HAL log is enabled and the debug level is set as warning to print warning and error messages.
The debug level of HAL is updated to INFO with the CLI command - `$log set hal on info`. After the command is issued, the HAL INFO message is printed in the log terminal. The setting for HAL is still effective after the device is powered down and up again. There is a special command to update the setting of all modules, by specifying the module name as the wild card "*". For example, the command `$log set * off warning` turns off logging and updates the debug level as WARNING for all modules.

```
$ log
log
module  on/off  level
------  ------  -----
main    on      error
common  on      warning
hal     on      warning

$ log set
log set
required parameters: <module_name> <log_switch> <print_level>
<log_switch>   := on | off
<print_level>  := info | warning | error

$ log set hal on info
log set hal on info

$ log
log
module  on/off  level
------  ------  -----
main    on      error
common  on      warning
hal     on      info

$ log fmt
Set format  : log fmt [module] [format]
Check format: log fmt [module]

$ log fmt hal
sntp   id     stm    line   func   level  module time
0      0      0      1      1      1      1      1
```

```
$ log fmt hal 0x3
sntp    id      stm     line    func    level   module  time
0       0       0       0       0       0       1       1

$ log switch on

 this is flash mode

$ log switch off

this is uart mode
```

1.4 Multitasking Support

Since there is only one serial port for system logging, a single log task is created to write the output log data to the serial port when it is scheduled. The other tasks call the System Log API to allocate a log buffer, format the log data by calling the C library function offered by the toolchain and send the log data to the log task. A fixed number of log buffers are allocated as a shared resource. The allocation and release of a log buffer is protected to ensure only one task can access it at a time.

1.5 Supported `printf()` Function

Syslog calls the `printf()` function implemented in the toolchain (GCC, KEIL, IAR). In GCC, the floating print function (`%f`) is not enabled by default. To enable it, specify it in linker flag (`LDFLAGS += -u _printf_float`). However, note that the code size increases if floating print is enabled.

1.6 Conditional Compile Options

The project makefile’s feature flag `MTK_DEBUG_LEVEL` defines whether a particular debug log is compiled. It can be configured in project’s `feature.mk` makefile. More compile options are described in the following table.

Table 2. Compile options for logging

MTK_DEBUG_LEVEL	The effect of the setting
none	All logs are removed.
info	LOG_I, LOG_W, LOG_E, LOG_HEXDUMP_I, LOG_HEXDUMP_W, LOG_HEXDUMP_E are compiled.
warning	LOG_W, LOG_E, LOG_HEXDUMP_W, LOG_HEXDUMP_E are compiled.
error	LOG_E, LOG_HEXDUMP_E are compiled.

1.7 Save Log to Flash

If the feature is enabled, logs are put in a dump buffer during printing and System Log tries to write the dump buffer to flash if there are no messages in the System Log queue. Considering the inconsistency between flash erase size and write size, the unit of writing the dump buffer to flash is 4 K, which means writing is not executed if the count of log in dump buffer is less than 4 K.
Sometimes there are always messages in the log queue. In this case, the dump buffer has no chance to be written to flash. After the dump buffer is full, System log forces to write the whole dump buffer to flash and clean it.
When exceptions happen, you have to write the logs to flash right before exceptions. However, the count of log in the dump buffer may be less than 4 K at this time. To prevent losing logs, System Log flushes the whole dump buffer to flash when exceptions happen.

1.7.1 Feature Enable

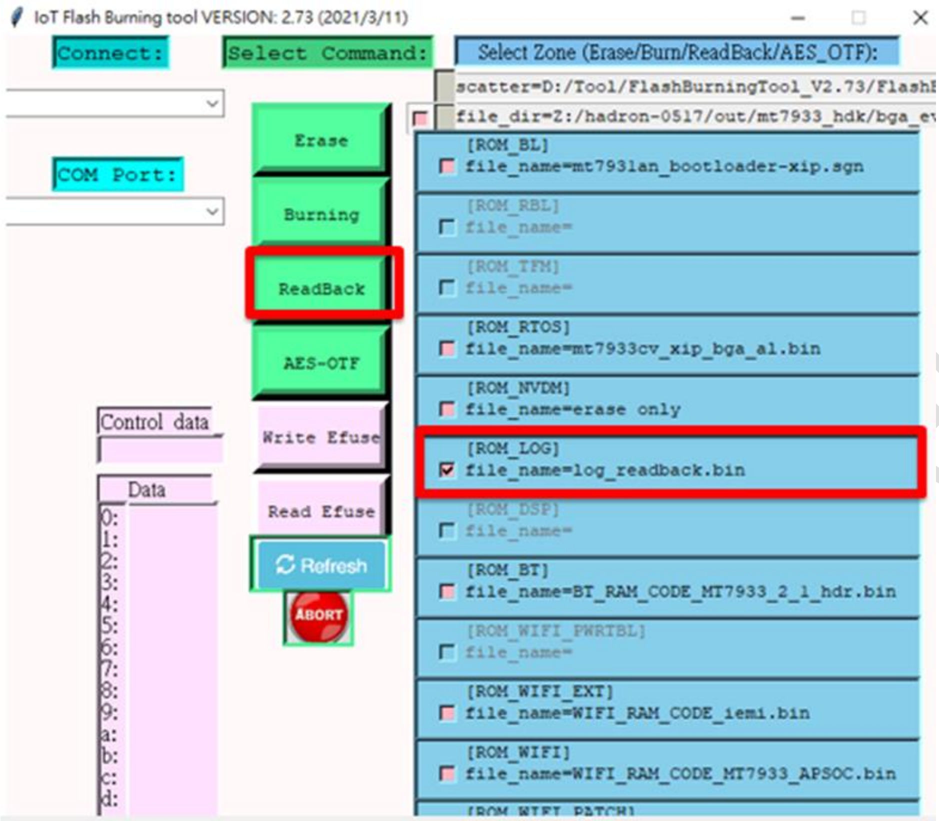
1. Feature Enable: MTK_SAVE_LOG_AND_CONTEXT_DUMP_ENABLE = y
2. Use cli "log switch on" to switch to flash mode.

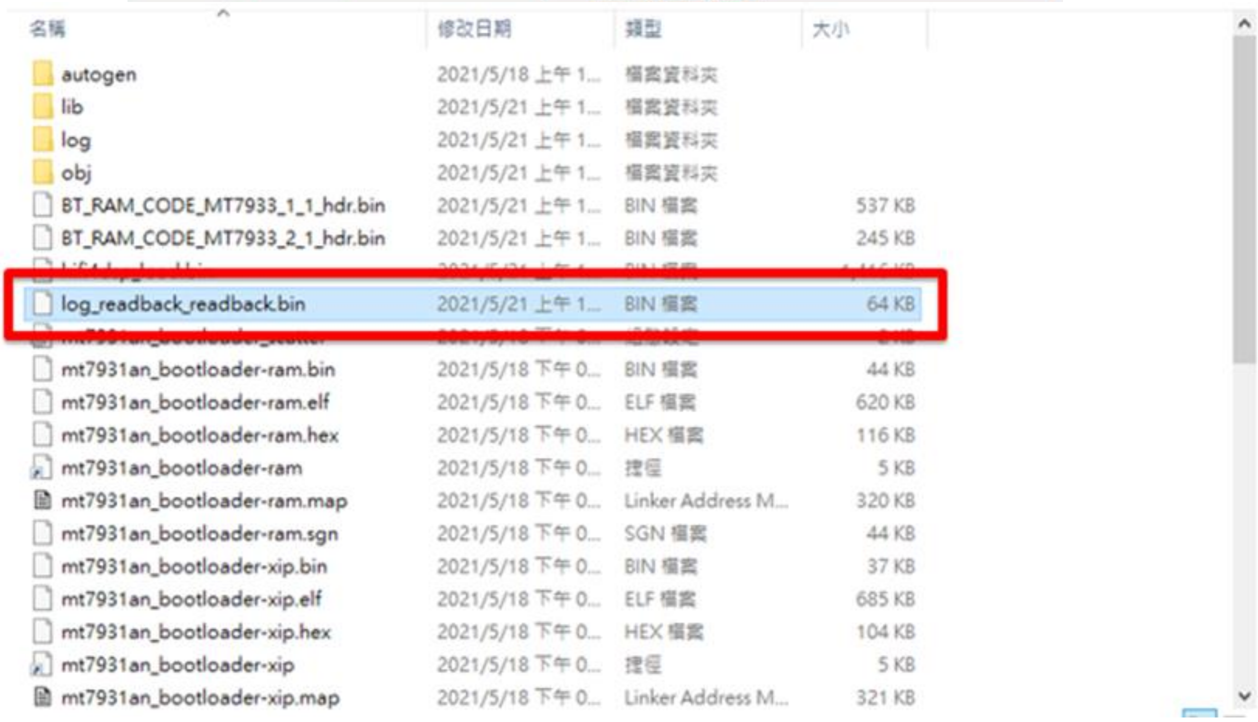
1.7.2 Read Logs from Flash

1. Readback the flash log section with FlashTool (Please refer to FlashTool User Guide). You should get a 64-K .bin file after you readback with FlashTool
2. Open the .bin you just readback with HxD – Freeware Hex Editor and Disk Editor. The logs are at 0x1000 of the .bin file.

1.7.3 Reference Software

HxD – Freeware Hex Editor and Disk Editor: https://mh-nexus.de/en/hxd/


IoT Flash Burning tool VERSION: 2.73 (2021/3/11)
Connect:
Select Command:
Select Zone (Erase/Burn/ReadBack/AES_OTF):
scatter=D:/Tool/FlashBurningTool_V2.73/FlashB
file_dir=Z:/hadron-0517/out/mt7933_hdk/bga_ev
COM Port:
Erase
Burning
ReadBack
AES-OTF
Write Efuse
Read Efuse
Refresh
ABORT
Control data
Data
[ROM_BL]
file_name=mt7931an_bootloader-xip.sgn
[ROM_RBL]
file_name=
[ROM_TFM]
file_name=
[ROM_RTOS]
file_name=mt7933cv_xip_bga_al.bin
[ROM_NVDM]
file_name=erase only
[ROM_LOG]
file_name=log_readback.bin
[ROM_DSP]
file_name=
[ROM_BT]
file_name=BT_RAM_CODE_MT7933_2_1_hdr.bin
[ROM_WIFI_PWRTBL]
file_name=
[ROM_WIFI_EXT]
file_name=WIFI_RAM_CODE_iemi.bin
[ROM_WIFI]
file_name=WIFI_RAM_CODE_MT7933_APSOC.bin


名稱
修改日期
類型
大小
autogen
lib
log
obj
BT_RAM_CODE_MT7933_1_1_hdr.bin
BT_RAM_CODE_MT7933_2_1_hdr.bin
log_readback_readback.bin
mt7931an_bootloader-ram.bin
mt7931an_bootloader-ram.elf
mt7931an_bootloader-ram.hex
mt7931an_bootloader-ram
mt7931an_bootloader-ram.map
mt7931an_bootloader-ram.sgn
mt7931an_bootloader-xip.bin
mt7931an_bootloader-xip.elf
mt7931an_bootloader-xip.hex
mt7931an_bootloader-xip
mt7931an_bootloader-xip.map

log_readback (1).bin

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	解碼的文字
00000FF0	C7	CE	87	CF	A0	92	9A	92	A0	8C	86	8C	00	80	00	00	ÇÎ‡Ï ’š’ Œ†Œ.€..
00001000	5B	33	39	33	31	36	5D	5B	42	54	5D	5B	49	5D	5B	62	[39316][BT][I][b
00001010	74	5F	64	65	62	75	67	5F	6C	6F	67	5D	5B	31	32	38	t_debug_log][128
00001020	5D	5B	62	74	5F	61	70	70	5F	69	6F	5F	63	61	6C	6C	][bt_app_io_call
00001030	62	61	63	6B	28	36	34	32	29	5D	5B	49	5D	5B	41	50	back(642)][I][AP
00001040	50	5D	62	74	5F	61	70	70	5F	69	6F	5F	63	61	6C	6C	P]bt_app_io_call
00001050	62	61	63	6B	20	63	6C	69	5F	63	6D	64	3A	20	69	6E	back cli_cmd: in
00001060	69	74	0D	0A	5B	33	39	33	31	37	5D	5B	42	54	5D	5B	it..[39317][BT][
00001070	49	5D	5B	62	74	5F	64	65	62	75	67	5F	6C	6F	67	5D	I][bt_debug_log]
00001080	5B	31	32	38	5D	5B	62	74	5F	73	65	74	74	69	6E	67	[128][bt_setting
00001090	5F	69	6E	69	74	28	33	30	35	29	5D	5B	49	5D	5B	42	_init(305)][I][B
000010A0	54	5D	72	65	61	64	20	74	68	65	20	76	61	6C	75	65	T]read the value
000010B0	20	66	61	69	6C	2C	20	72	65	73	75	6C	74	20	3D	20	fail, result =
000010C0	2D	34	0D	0A	5B	33	39	33	31	37	5D	5B	42	54	5D	5B	-4..[39317][BT][
000010D0	49	5D	5B	62	74	5F	64	65	62	75	67	5F	6C	6F	67	5D	I][bt_debug_log]
000010E0	5B	31	32	38	5D	5B	62	74	5F	73	65	74	74	69	6E	67	[128][bt_setting
000010F0	5F	69	6E	69	74	28	33	31	39	29	5D	5B	49	5D	5B	42	_init(319)][I][B
00001100	54	5D	72	65	61	64	20	74	68	65	20	76	61	6C	75	65	T]read the value
00001110	20	66	61	69	6C	2C	20	72	65	73	75	6C	74	20	3D	20	fail, result =
00001120	2D	34	0D	0A	5B	33	39	33	31	37	5D	5B	42	54	5D	5B	-4..[39317][BT][
00001130	49	5D	5B	62	74	5F	64	65	62	75	67	5F	6C	6F	67	5D	I][bt_debug_log]
00001140	5B	31	32	38	5D	5B	62	74	5F	73	65	74	74	69	6E	67	[128][bt_setting
00001150	5F	69	6E	69	74	28	33	33	30	29	5D	5B	45	5D	5B	42	_init(330)][E][B
00001160	54	5D	72	65	61	64	20	74	68	65	20	76	61	6C	75	65	T]read the value
00001170	20	66	61	69	6C	2C	20	72	65	73	75	6C	74	20	3D	20	fail, result =
00001180	2D	34	0D	0A	5B	33	39	33	31	37	5D	5B	42	54	5D	5B	-4..[39317][BT][
00001190	49	5D	5B	62	74	5F	64	65	62	75	67	5F	6C	6F	67	5D	I][bt_debug_log]
000011A0	5B	31	32	38	5D	5B	62	74	5F	73	65	74	74	69	6E	67	[128][bt_setting
000011B0	5F	69	6E	69	74	28	33	33	30	29	5D	5B	45	5D	5B	42	_init(330)][E][B
000011C0	54	5D	72	65	61	64	20	74	68	65	20	76	61	6C	75	65	T]read the value
000011D0	20	66	61	69	6C	2C	20	72	65	73	75	6C	74	20	3D	20	fail, result =
000011E0	2D	34	0D	0A	5B	33	39	33	31	37	5D	5B	42	54	5D	5B	-4..[39317][BT][
000011F0	49	5D	5B	62	74	5F	64	65	62	75	67	5F	6C	6F	67	5D	I][bt_debug_log]

2 System Log APIs

This section describes the source code and header file hierarchy for the System Log and their usage. The files could be found under `<sdk_root>/kernel/service/src`.
The following table provides details on the main functions of the System Log tree hierarchy.

Table 3. System Log source code description

File	Description
`<project>/GCC/feature.mk`	Define MTK_DEBUG_LEVEL
`kernel/service/inc/syslog.h`	The main header file for the System Log. It includes the macros and APIs for initialization, logging and filtering purposes.
`kernel/service/src/syslog.c`	Contains the internal implementation of the System Log API. It also declares the log control block for a common module that enables you to print log messages of the common module.
`kernel/service/src/syslog_cli.c`	Includes the syslog CLI commands for the MT7933 HDK.

2.1 Supported APIs

The following lists the most commonly used APIs.

2.1.1 log_init()

Description	This function initializes the system log. The syslog control blocks table (see Section 1.3 Log Setting Commands) is used in syslog related CLI commands.
Parameter	**syslog_save_fn save**. User-defined callback function to store log settings. **syslog_load_fn load**. User-defined callback function to read log settings. **log_control_block_t **entries**. A pointer to syslog control blocks table for the project.

2.1.2 LOG_CONTROL_BLOCK_DECLARE()

Description	This macro declares the external reference to a log control block (see Section 1.3 Log Setting Commands) of a module.
Parameter	**module:** Module name.

2.1.3 LOG_CONTROL_BLOCK_SYMBOL

Description	This macro refers to the log control block (see Section 1.3 Log Setting Commands) of a module.
Parameter	**module**. Module name.

2.1.4 log_create_module()

Description	This macro creates the log control block of a module.
Parameter	**module**. Module name. **level**. "PRINT_LEVEL_INFO", the debug level for the module is INFO. "PRINT_LEVEL_WARNING", the debug level for the module is WARNING. "PRINT_LEVEL_ERROR", the debug level for the module is ERROR.

2.1.5 log_config_print_switch()

Description	This macro configures whether the log for the module is enabled.
Parameter	**module**. Module name. **log_switch**. "DEBUG_LOG_ON", the log for the module is enabled. "DEBUG_LOG_OFF", the log for the module is disabled.

2.1.6 log_config_print_level()

Description	This macro configures the debug level for the module.
Parameter	**module**. Module name. **log_switch**. "PRINT_LEVEL_INFO", the debug level of the module is INFO. "PRINT_LEVEL_WARNING", the log level of the module is WARNING. "PRINT_LEVEL_ERROR", the debug level of the module is ERROR.

2.1.7 LOG_I

Description	This macro adds an INFO log message.
Parameter	**module**. Module name. **message**. Format specifiers. **variadic arguments**. The parameters corresponding to the format specifiers defined in the **message** parameter.

2.1.8 LOG_W

Description	This macro adds a WARNING log message.
Parameter	**module**. Module name. **message**. Format specifiers. **variadic arguments**. The parameters corresponding to the format specifiers defined in the **message** parameter.

2.1.9 LOG_E

Description	This macro adds an ERROR log message.
Parameter	**module**. The module name. **message**. Format specifiers. **variadic arguments**. The parameters corresponding to the format specifiers defined in the **message** parameter.

2.1.10 LOG_HEXDUMP_I

Description	This macro adds an INFO log message and displays the contents of a specified range of memory. The memory is displayed in hexadecimal format.
Parameter	**module**. The module name. **message**. Format specifiers. **data**. The start address of the memory region to be displayed. **len**. The length of the memory region to be displayed, in bytes. **variadic arguments**. The parameters corresponding to the format specifiers defined in the **message** parameter.

2.1.11 LOG_HEXDUMP_W

Description	This macro adds a WARNING log message and displays the contents of a specified range of memory. The memory is displayed in hexadecimal format.
Parameter	**module**. The module name. **message**. Format specifiers. **data**. The start address of the memory region to be displayed. **len**. The length of the memory region to be displayed, in bytes. **variadic arguments**. The parameters corresponding to the format specifiers defined in the **message** parameter.

2.1.12 LOG_HEXDUMP_E

Description	This macro adds an ERROR log message and displays the contents of a specified range of memory. The memory is displayed in hexadecimal format.
Parameter	**module**. The module name. **message**. Format specifiers. **data**. The start address of the memory region to be displayed. **len**. The length of the memory region to be displayed, in bytes. **variadic arguments**. The parameters corresponding to the format specifiers defined in the **message** parameter.

2.2 System Log API Usage

In this section, the System Log APIs are utilized to create and configure logs for a specific module. A user defined function is then invoked to demonstrate the logging operation for different levels of log configuration. The code below declares a log control block for the Hardware Abstraction Layer (HAL) module. In this example, the log level for HAL is configured as `PRINT_LEVEL_WARNING`. Call `log_create_module` to set the configuration, as shown below.

```
#include "syslog.h"

log_create_module(hal, PRINT_LEVEL_WARNING);
```

A log control block for the HAL module with the content shown below is created:

```
log_control_block_t log_control_block_hal =
{
    .module_name = "hal",
    .log_switch = (DEBUG_LOG_ON),
    .print_level = ((PRINT_LEVEL_WARNING)),
    .f_print_handler = default_print_ext,
    .f_dump_buffer = default_dump_ext
};
```

The following user defined function (`demo_function()`) demonstrates the usage of the System Log API. Only the log messages with a print level greater than or equal to PRINT_LEVEL_WARNING are printed.

```
void demo_function(int id1, int id2, char *statement)
{
    LOG_I(hal, "INFO message %d\n", id1);         /* It is not printed
*/
    LOG_W(hal, "WARNING message %d\n", id2);      /* It is printed */
    LOG_E(hal, "ERROR message %s\n", statement);  /* It is printed */
}
```

2.3 System Log Output Format

Each log message includes a log header and a log body. By default, the log header shows **timestamp**, **sequence number, module name**, **level**, **function**, and **line number**. The log body shows the user-defined log.

```
[35349]<101>[common][I][_os_cli_test][369] Hello World
```

To configure log header format for log module through cli, refer to 1.3.1 CLI Commands

Log header format is denoted as a uint8_t number. Each bit of the number represents a different kind of information. 1 means enable while 0 means disable.

Bit	Meaning
0	time
1	module
2	level
3	function
4	line
5	STM

Bit	Meaning
6	id
7	sntp

For example, if you want to show only “module”, “function” and “line”, you can set the format as “0x1A” because 0x1A is equal to “00011010” in binary. With the binary string, you can see that “module”, “function” and “line” are enabled while others are disabled.

3 Host UART Configuration

The System Log messages on the host side are displayed using a terminal program, such as TeraTerm and Putty. The terminal program is able to receive text data from the serial port. TeraTerm terminal program is selected for the demonstration purposes. The hardware development board is the MT7933 development board.

The host UART settings are configured as follows:

1) Set up the port.
   a) Open Windows Control Panel, then click **System** and:
      - On Windows 7 and 8, click **Device Manager**.
      - On Windows XP, click the **Hardware** tab and then **Device Manager**.
   b) In **Device Manager**, navigate to **Ports (COM & LPT)** (see Figure 2).
   c) Connect the development board to your computer using a Micro-USB cable.
   d) A new **COM** device should appear under **Ports (COM & LPT)** in **Device Manager**, as shown in Figure 2. Note the **COMx** port number of the serial communication port; this information is needed to complete configuration of the Tera Term terminal program.
2) Launch the Tera Term terminal program and open the **Serial port setup** window.
   a) Assign the COM port number found in 1) Set up the port. The rest of the parameters such as **Baud rate**, **Data**, **Parity**, **Stop** and **Flow control** should be defined, as shown in Figure 3.

Figure 2. Enumerated COM port on the host

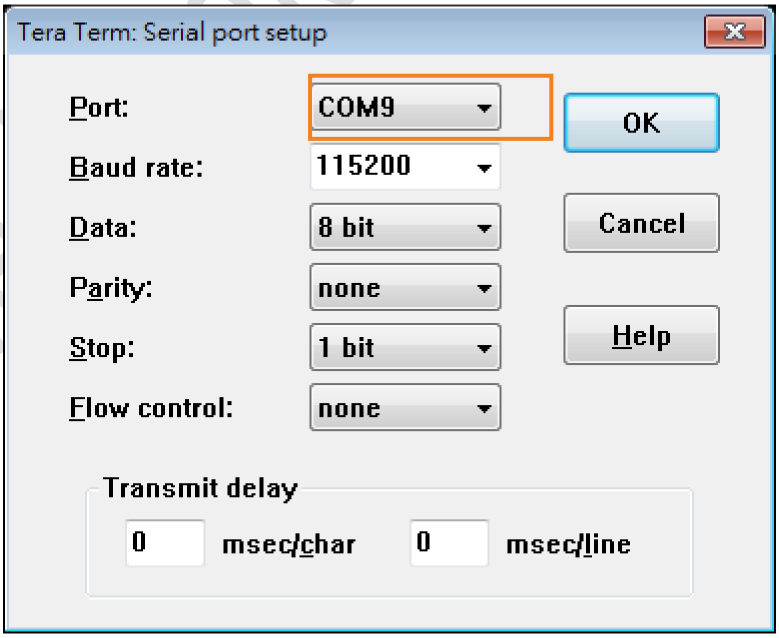


Figure 3. UART configurations in Tera Term terminal program

4 Exception Handler

Exception handler assists in debugging programming errors. When an exception occurs, the content of the processor's core registers and memory are dumped into a log file. The data in the dump file can be analyzed given the symbol information in the corresponding ELF file. Exceptions are either processor detected exceptions or user defined assert failures. Four types of hardware exceptions are supported in Exception Handler for ARM Cortex-M33: **HardFault**, **MemMange**, **BusFault**, and **UsageFault**. Refer to ARM v8-M Architecture Reference Manual for more detail.

The following table describes the source code and header file hierarchy for the Exception Handler and their usage.

Table 4 Exception Handler source code description

File	Description
`kernel/service/inc/exception_handler.h`	The main header file for the Exception Handler. It includes data types and APIs.
`kernel/service/src/memory_regions.c`	Contains the system memory map definition to facilitate the Exception Handler to dump memory. It is mainly a table of linker generated symbols of the system.
`kernel/service/src/exception_handler.c`	Contains the internal implementation of the Exception Handler.

4.1 Supported APIs

The most commonly used APIs are listed below..

4.1.1 configASSERT()

Description	Use this macro to add assert expression. If the expression is evaluated as FALSE, platform_assert() is called.
Parameter	**expr**. The expression to be evaluated.

4.1.2 abort()

Description	A porting function used in GCC toolchain. It is linked with toolchain's assert().
Parameter	**none**

4.1.3 __aeabi_assert()

Description	A porting function used in Keil or IAR embedded workbench toolchain. It is linked with tool chain's assert().
Parameter	**const char *expr**. User-defined assert expression. **const char *file**. Source file where assert expression is added. **int line**. Line number of the assert expression.

4.1.4 platform_assert()

Description	This function is called when assert check fails. It is used in the configASSERT macro. It intentionally performs invalid memory access to bring the system into exception.
Parameter	**const char *expr**. User-defined assert expression. **const char *file**. Source file where assert expression is added. **int line**. Line number of the assert expression.

4.1.5 exception_register_callbacks()

Description	You can register init_cb and dump_cb. init_cb is called when an exception occurs. It is designed for user-defined functions, such as trigger system log buffer flush operation. dump_cb is called after memory dump is complete.
Parameter	**exception_config_type *cb**. User-defined callback functions.

4.1.6 exception_dump_config()

Description	This function is used to customize the exception handler behavior.
Parameter	**int flag**. DISABLE_MEMDUMP_MAGIC: Exception handler calls exception_reboot() after dumping the core registers. Other values: no effect, reserved for future use.

4.1.7 exception_reboot()

Description	The exception handler calls this function after the core register is dumped if the memory dump feature is disabled (see Section 4.1.6 exception_dump_config()). exception_reboot() is implemented as a weak function in exception handler to allow user customization as the reboot function is chip and project dependent (see Section 4.3 Reboot without Memory Dump on Exception).
Parameter	**none**

4.2 Enable Exception Handling in Projects

For GCC environment, you need to include `kernel/service/module.mk` in the project's Makefile.

```
# kernel service files
include $(SOURCE_DIR)/kernel/service/module.mk
"project/mt7933_hdk/apps/iot_sdk_demo/GCC/Makefile" 300 lines --25%--
```

Using `configASSERT()` instead of `assert()`is recommended. The "`__noreturn__`" attribute specified for the `assert()` function informs the compiler that the function does not return. The compiler, performing optimization, may not save the key registers necessary for callstack unwind. The `configASSERT()` macro is defined in the project's `FreeRTOSConfig.h`.

```
#if defined(__ICCARM__) || defined(__CC_ARM) || defined(__GNUC__)
extern void abort(void);
extern void platform_assert(const char *, const char *, int);
...
#define configASSERT( x ) if( (x) == 0 ) { platform_assert(#x, __FILE__,
__LINE__); }
#include "syslog.h"
#endif /*#if defined(__ICCARM__) || defined(__CC_ARM) ||
defined(__GNUC__)*/
"project/mt7933_hdk/apps/iot_sdk_demo/inc/FreeRTOSConfig.h"
```

The project's memory map defined in `kernel/service/src/memory_regions.c` facilitates the exception handler to create a memory dump, which is a table of the linker-generated symbols specifying the location of image sections.

```
const memory_region_type memory_regions[] =
{
    {"ram_text", Image$$RAM_TEXT$$Base, Image$$RAM_TEXT$$Limit, 1},
    {"noncached_data", Image$$NONCACHED_DATA$$Base,
Image$$NONCACHED_ZI$$Limit, 1},
    {"cached_data", Image$$CACHED_DATA$$RW$$Base,
Image$$CACHED_DATA$$ZI$$Limit, 1},
    {"tcm", Image$$TCM$$RO$$Base, Image$$STACK$$ZI$$Limit, 1},
    {"stack", Image$$STACK$$ZI$$Base, Image$$STACK$$ZI$$Limit, 0},
    {"scs", (unsigned int*)SCS_BASE, (unsigned int*)(SCS_BASE + 0x1000),
1},
    {0}
};
"kernel/service/src/memory_regions.c"
```

4.3 Reboot without Memory Dump on Exception

You should call `exception_dump_config(DISABLE_MEMDUMP_MAGIC)` during system initialization and implement the `exception_reboot()` function. Normally reboot can be implemented by hardware watchdog. Refer to HAL/WDT section in `<SDK_root>/doc/Mediatek IoT SDK for Chipset API Reference Manual.html`, for more details.

4.4 Exception Dump Example

An exception dump example for the MT7933 is shown in Figure 4 and Figure 5.

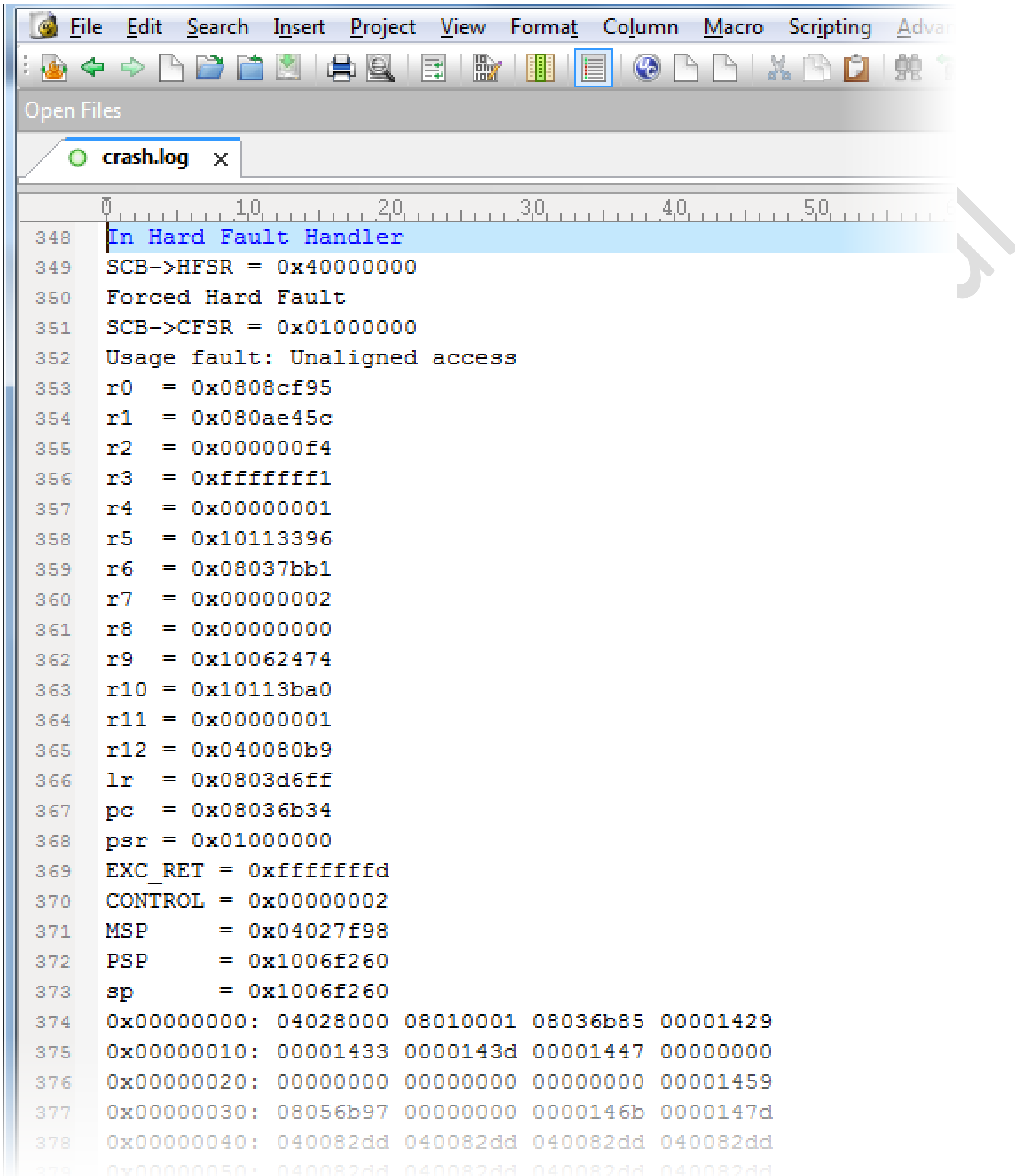


Figure 4. Exception dump example (1/2)

Then the system memory is dumped and "`memory dump completed.`" message is displayed, as shown in Figure 5.

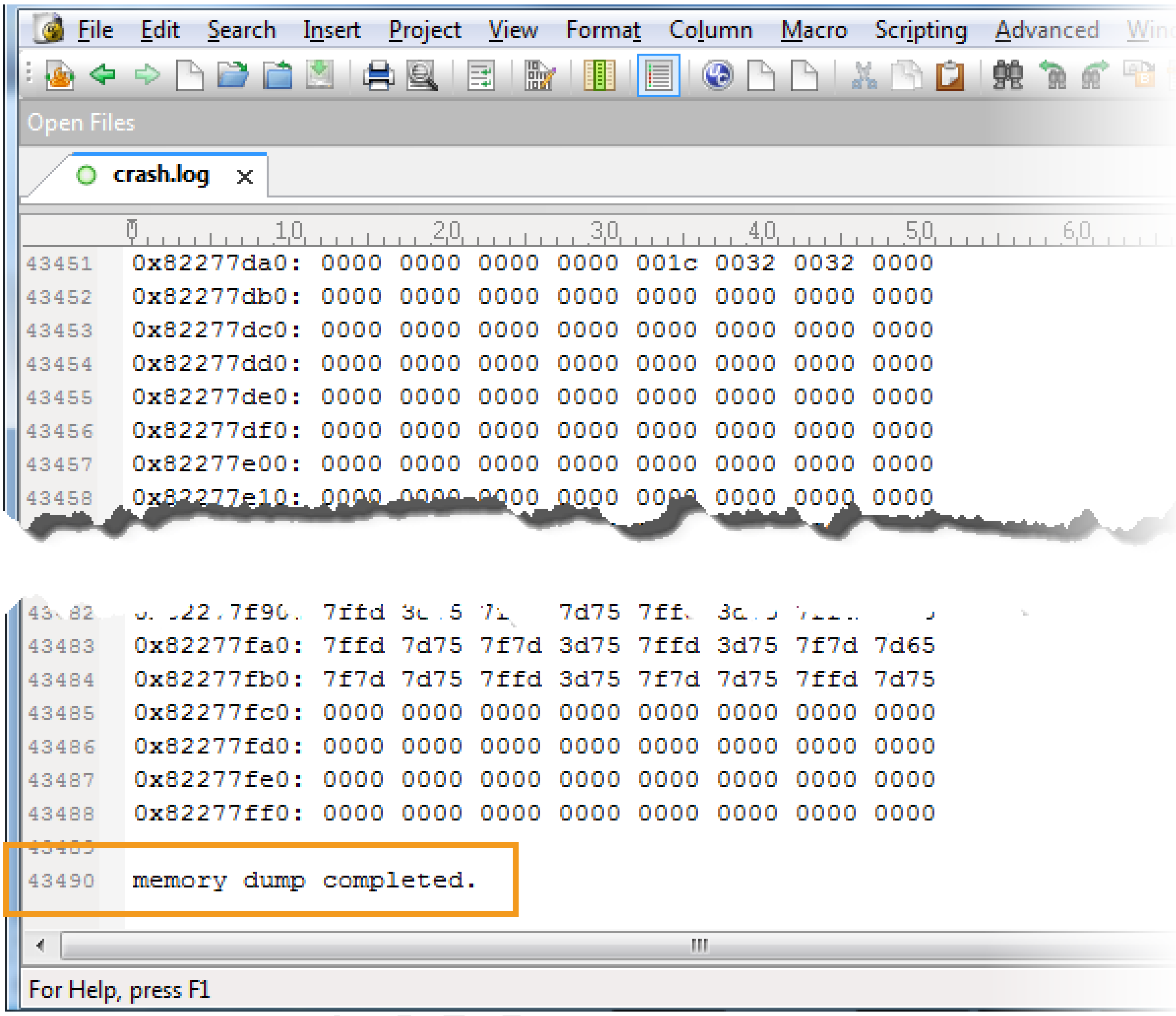


Figure 5. Exception dump example (2/2)

Exhibit 1 Terms and Conditions

Your access to and use of this document and the information contained herein (collectively this "Document") is subject to your (including the corporation or other legal entity you represent, collectively "You") acceptance of the terms and conditions set forth below ("T&C"). By using, accessing or downloading this Document, You are accepting the T&C and agree to be bound by the T&C. If You don't agree to the T&C, You may not use this Document and shall immediately destroy any copy thereof.

This Document contains information that is confidential and proprietary to MediaTek Inc. and/or its affiliates (collectively "MediaTek") or its licensors and is provided solely for Your internal use with MediaTek's chipset(s) described in this Document and shall not be used for any other purposes (including but not limited to identifying or providing evidence to support any potential patent infringement claim against MediaTek or any of MediaTek's suppliers and/or direct or indirect customers). Unauthorized use or disclosure of the information contained herein is prohibited. You agree to indemnify MediaTek for any loss or damages suffered by MediaTek for Your unauthorized use or disclosure of this Document, in whole or in part.

MediaTek and its licensors retain titles and all ownership rights in and to this Document and no license (express or implied, by estoppels or otherwise) to any intellectual propriety rights is granted hereunder. This Document is subject to change without further notification. MediaTek does not assume any responsibility arising out of or in connection with any use of, or reliance on, this Document, and specifically disclaims any and all liability, including, without limitation, consequential or incidental damages.

THIS DOCUMENT AND ANY OTHER MATERIALS OR TECHNICAL SUPPORT PROVIDED BY MEDIATEK IN CONNECTION WITH THIS DOCUMENT, IF ANY, ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE. MEDIATEK SPECIFICALLY DISCLAIMS ALL WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, FITNESS FOR A PARTICULAR PURPOSE, COMPLETENESS OR ACCURACY AND ALL WARRANTIES ARISING OUT OF TRADE USAGE OR OUT OF A COURSE OF DEALING OR COURSE OF PERFORMANCE. MEDIATEK SHALL NOT BE RESPONSIBLE FOR ANY MEDIATEK DELIVERABLES MADE TO MEET YOUR SPECIFICATIONS OR TO CONFORM TO A PARTICULAR STANDARD OR OPEN FORUM.

Without limiting the generality of the foregoing, MediaTek makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does MediaTek assume any liability arising out of the application or use of any product, circuit or software. You agree that You are solely responsible for the designing, validating and testing Your product incorporating MediaTek's product and ensure such product meets applicable standards and any safety, security or other requirements.

The above T&C and all acts in connection with the T&C or this Document shall be governed, construed and interpreted in accordance with the laws of Taiwan, without giving effect to the principles of conflicts of law.