MStar CTP
Capacitive Touch Panel Controller

Device Driver User Guideline Version 0.3

MStar Semiconductor makes no representations or warranties including, for example but not limited to, warranties of merchantability, fitness for a particular purpose, non-infringement of any intellectual property right or the accuracy or completeness of this document, and reserves the right to make changes without further notice to any products herein to improve reliability, function or design. No responsibility is assumed by MStar Semiconductor arising out of the application or use of any product or circuit described herein; neither does it convey any license under its patent rights, nor the rights of others.

MStar is a trademark of MStar Semiconductor, Inc. Other trademarks or names herein are only for identification purposes only and owned by their respective owners.

REVISION HISTORY

Revision No.	Description	Date
0.1	Ÿ Initial release	10/07/2014
0.2	Ÿ Added description for device drivers which support MSG28xx Ÿ Added description for device drivers which support Hotknot	05/05/2015
0.3	Ÿ Added description for new features of touch device driver	05/13/2016

TABLE OF CONTENTS

1. INTRODUCTION

1.1. Purpose

Touch Device Driver 從 v2.0.0.0 版開始因應 MStar 的觸控 IC 被使用在不同的智慧型手機開發平台(EX. MTK/QCOM/SPRD)，減少 FAE 同仁或客戶合入與驗證 MStar 觸控 IC 裝置驅動程式的複雜度和所需花費的時間， 並增加 Touch Device Driver 往後在功能擴充上的彈性。目前此一新版 Touch Device Driver 已支援 MStar 的互容屏(MSG26xxM/MSG28xx/MSG58xx)及自容屏(MSG21xxA/MSG22xx)觸控 IC 在 MTK/QCOM/SPRD 等三種不同開發平台上的使用。 至於詳細的使用說明，請參閱下各章節的描述。

注意事項:

1. 本篇使用說明文件主要針對 MStar 新 Release 的 Touch Device Driver v2.0.0.0 版以後的觸控驅動裝置代碼相關說明，並不包含 Touch Device Driver v2.0.0.0 版以前的版本。
2. 本篇文章中所使用的公司名稱縮寫 MTK/QCOM/SPRD，分別代表 MediaTek Inc/Qualcomm International, Inc/ Spreadtrum Communication,Inc 等公司。
3. FAE 同仁或客戶在調適 MStar 的 Touch Device Driver 時，可直接先參照章節 2.1. MStar_drv_common.h & MStar_drv_common.c 及章節 2.6 MStar_drv_platform_porting_layer.h & MStar_drv_platform_porting_layer.c 的說明。另外，依據客戶專案所使用的手機開發平台(EX. SPRD/QCOM/MTK)，可再另行參閱章節 2.9. MStar_drv_sprd.c 或 2.10. Mstar_drv_qcom.c 或 2.11. Mstar_drv_mtk.c。

2. FUNCTION STATEMENT

2.1. MStar_drv_common.h & MStar_drv_common.c

在整個 Touch Device Driver 代碼中，MStar 的互容屏(MSG26xxM/MSG28xx/MSG58xx)及自容屏(MSG21xxA/MSG22xx)觸控 IC 共用的宏(Compile Option)， 函式(Function)及常數值(Constant Value)定義等， 都定義在 MStar_drv_common.h 中並詳述如下。

2.1.1 Compile Option

下列針對 MStar_drv_common.h 中較重要的 Compile Option 作進一步說明， 其餘請直接參照 MStar_drv_common.h 和 MStar_drv_common.c。

2.1.1.1. #define CONFIG_TOUCH_DRIVER_RUN_ON_SPRD_PLATFORM
#define CONFIG_TOUCH_DRIVER_RUN_ON_QCOM_PLATFORM
#define CONFIG_TOUCH_DRIVER_RUN_ON_MTK_PLATFORM

上述的三個不同 Compile Option 是用來分別 Enable 在 Touch Device Driver 代碼中，針對各個不同智慧型手機開發平台的相關代碼處理，請依據目前客戶專案所使用的手機開發平台來決定要 Enable 哪一個 Compile Option， 並 Disable 另外兩個 Compile Option。

2.1.1.2. #define CONFIG_PLATFORM_USE_ANDROID_SDK_6_UPWARD

如果客戶專案用的是 MTK 平台的 BB chip，且該 MTK 平台搭配安卓系統是 Android 6.0(包括)以上版本，則必須 Enable 上述的 Compile Option，Touch Device Driver 才能正常註冊和使用。此 Compile Option 只針對 MTK 平台，不針對 QCOM 平台或 SPRD 平台(此 Compile Option 預設值為 Disable)。

2.1.1.3. #define CONFIG_ENABLE_REGULATOR _POWER_ON

在 SPRD 平台的某些特定 BB Chip(EX. SC7715)或 QCOM 平台的某些特定 BB Chip(EX. MSM8610)或 MTK 平台的某些特定 BB Chip(EX. MT6582)，必須多次執行用此 Compile Option 包起來的函式 DrvPlatformLyrTouchDeviceRegulatorPowerOn()才能讓主板可以供電給觸控 IC(此 Compile Option 預設值是 Disable). 因此在客戶端合入 MStar 的 Touch Device Driver 後，如果有觸控 IC 無法正常上電的狀況，需要 FAE 同仁跟客戶的驅動工程師確認該專案所使用的 BB Chip 是否需要 Enable 上述的 Compile Option，甚至是需修改 DrvPlatformLyrTouchDeviceRegulatorPowerOn()內上電的作法。

2.1.1.4. #define CONFIG_ENABLE_TOUCH_PIN_CONTROL

上述的 Compile Option 主要是針對 SPRD 平台或 QCOM 平台的某些特定 BB Chip，必須 Enable 上述的 Compile Option，才能透過 Pin Control 機制取得 RESET Pin 跟 INT Pin 的正確設定值。至於客戶專案所使用的 SPRD 平台或 QCOM 平台是否需要 Enable 上述的 Compile Option，需要 FAE 同仁跟客戶的驅動工程師確認(此 Compile Option 預設值為 Disable)。

2.1.1.5. #define CONFIG_USE_IRQ_INTERRUPT_FOR_MTK_PLATFORM

當中斷 Touch Device Driver 收到 Firmware 打上來的報點後，目前在 MTK 平台看到可以透過底下的兩種方式(Method A/Method B)來創建一個 Work Queue， 並透過此 Work Queue 來 Schedule 處理報點封包的讀取及解析。目前在新版的 Touch Device Driver 架構， 針對 MTK 平台會同時支援下述兩種 Work Queue 機制， 並用 Compile Option CONFIG_USE_IRQ_INTERRUPT_FOR_MTK_PLATFORM 作為區隔(此 Compile Option 預設值為 Disabled). 之後可由 FAE 同仁或客戶自行決定要使用何種 Work Queue 機制。

詳細請參閱在 MStar_drv_platform_porting_layer.c 中的實作步驟，部份代碼如下。

```
#ifdef CONFIG_USE_IRQ_INTERRUPT_FOR_MTK_PLATFORM
    /* initialize the finger touch work queue */
    INIT_WORK(&_gFingerTouchWork， _DrvPlatformLyrFingerTouchDoWork); // Method A
#else
    _gThread = kthread_run(_DrvPlatformLyrFingerTouchHandler， 0， TPD_DEVICE); // Method
B
    if (IS_ERR(_gThread))
    {
        nRetVal = PTR_ERR(_gThread);
        DBG("Failed to create kernel thread: %d\n"， nRetVal);
    }
#endif //CONFIG_ENABLE_IRQ_INTERRUPT_FOR_MTK_PLATFORM
```

2.1.1.6. #define CONFIG_ENABLE_DMA_IIC

因為在 MTK 手機開發平台上，某些型號的 BB(Base Band) Chip(Ex: MT6572/MT6582/MT6589)有 I2C 的限制: read/write 一次最多 8 Byte，因此 Touch Device Driver 和 TP Firmware 或 Touch IC 之間的資料傳輸，必須另外 Enable 另一個 DMA Mode 的機制才能跨越此 I2C read/write 的限制(此 Compile Option 預設值是 Disable， 請依據客戶專案所使用的 MTK BB Chip 是否有 I2C read/write 一次最多 8 Byte 的限制，進而決定是否 Enable)。

2.1.1.7. #define CONFIG_TP_HAVE_KEY

上述的 Compile Option 是用來 Enable 在 Touch Device Driver 代碼中， 針對手機 TP 上四個 Virtual Key(EX. Menu、Home、 Back、Search)的相關代碼處理(此 Compile Option 預設值為 Enable)。

2.1.1.8. #define CONFIG_ENABLE_REPORT_KEY_WITH_COORDINATE

因為在 MTK 手機開發平台上，Touch Device Driver 是使用轉換後的座標資訊來回報 TP 哪一個 Virtual Key 給 Linux's Input Sub-System，而非直接使用 Virtual Key 的鍵值回報。因此在 MTK 平台上，必須 Enable 上述的 Compile Option，TP 上的 Virtual Key 才能正常被使用。如果在其它手機開發平台(EX. QCOM/SPRD)，亦有使用轉換後的座標資訊回報 TP 哪一個 Virtual Key，那麼只需 Enable 上述 Compile Option 即可(此 Compile Option 在 MTK 平台上預設值是 Enable，在 QCOM/SPRD 平台上預設值是 Disable)。

2.1.1.9. #define CONFIG_ENABLE_FIRMWARE_DATA_LOG (1)

上述的 Flag 是用以控制觸控固件(TP Firmware)可以透過 MStar 提供的 MTPTool.apk 讓 TP Firmware 從 Demo Mode 切換至 Debug Mode，並將 Debug Mode 資料輸出到手機 T Card 上的一個.csv 檔，最後再透過 MStar 提供的 MxViewer 工具回播.csv 檔，用 UI 顯示的方式方便 TP Firmware 同仁分析觸控固件問題(此 Flag 預設值是 1。1 是 Enable，0 是 Disable)。

至於如何使用上述整機錄製觸控固件 Debug Mode 資料操作說明及相關工具，可以請 MStar 的 FAE 同仁協助提供。

2.1.1.10. #define CONFIG_ENABLE_SEGMENT_READ_FINGER_TOUCH_DATA

客戶專案所使用的手機開發平台通常有 I2C 的 read/write 長度限制(EX. 一次最多 255/1024 Byte)，如果客戶專案的 Touch Device Driver 有開啟#define CONFIG_ENABLE_FIRMWARE_DATA_LOG (1)功能的話，則此 Compile Option 必須 Enable，如此 Touch Device Driver 才能透過分段讀取機制從 TP Firmware 取回封包長度超過平台 I2C 讀取限制的 Debug Mode 資料封包(此 Compile Option 預設值為 Enable)。

2.1.1.11. #define CONFIG_ENABLE_GESTURE_WAKEUP

上述的 Compile Option 是用以 Enable 在 Touch Device Driver 代碼中，針對 MStar 觸控 IC 支援手勢喚醒功能的相關代碼處理。目前從 Touch Device Driver v3.0.0.0 版開始之後的版本，都已支援 MStar 的互容屏(MSG26xxM/MSG28xx/MSG58xx)及自容屏(MSG21xxA/MSG22xx)觸控 IC 的手勢喚醒功能(此 Compile Option 預設值是 Disable，請依據客戶需求決定是否 Enable)。

2.1.1.12. #define CONFIG_SUPPORT 64_TYPES_GESTURE_WAKEUP_MODE

目前 MStar 的互容屏(MSG28xx/MSG58xx)及自容屏(MSG22xx)觸控 IC 支援的手勢喚醒功能(預設已支援手勢有 13 種)，有區分最多 16 種手勢和最多 64 種手勢。但 MStar 的互容屏(MSG26xxM)及自容屏(MSG21xxA)觸控 IC 僅支援最多 16 種手勢，不支援最多 64 種手勢。如果 Enable 上述的 Compile Option，表示 Touch Device Driver 支援最多 64 種手勢的處理。否則，表示 Touch Device Driver 支援最多 16 種手勢的處理(此 Compile Option 預設值為 Disable)。

2.1.1.13. #define CONFIG_ENABLE_GESTURE_DEBUG_MODE

上述的 Compile Option 是 TP Firmware 用來釐清手勢喚醒問題的偵錯功能，需搭配 MStar 的 MTPTool APK 一起使用，客戶專案不會用到(此 Compile Option 預設值為 Disable)。

2.1.1.14. #define CONFIG_ENABLE_GESTURE_INFORMATION_MODE

上述的 Compile Option 是用以 Enable 在 Touch Device Driver 代碼中，針對 MStar 觸控 IC 支援手勢回顯功能的相關代碼處理，用來回傳 TP Firmware 的手勢辨識軌跡資訊給上層 APK(此 Compile Option 預設值為 Disable)。

至於如何使用上述 Gesture Wakeup 功能的操作說明， 要麻煩 FAE 同仁向文管中心(DCC)申請 "mstar_ctp_手勢喚醒功能規格說明_internal.pdf" (給 FAE 同仁使用) 或 "mstar_ctp_手勢喚醒功能規格說明_customer.pdf" (給客戶使用)文件的最新版本，內有詳細說明。

2.1.1.15. #define CONFIG_ENABLE_CHIP_TYPE_MSG21XXA
#define CONFIG_ENABLE_CHIP_TYPE_MSG22XX
#define CONFIG_ENABLE_CHIP_TYPE_MSG26XXM
#define CONFIG_ENABLE_CHIP_TYPE_MSG28XX

當底下的 Compile Option #define CONFIG_ENABLE_ITO_MP_TEST 或#define CONFIG_UPDATE_FIRMWARE_BY_SW_ID 這兩項功能其中之一有被 Enable 的前提下，請依據客戶專案所使用的觸控 IC 型號來決定要 Enable 哪一個 Chip Type 的 Compile Option，並 Disable 其它另三個 Chip Type 的 Compile Option，如此編譯代碼時才會只編譯跟該 Chip Type 相關的 MP Test 功能或 SW ID 更新固件功能的代碼。

2.1.1.16. #define CONFIG_ENABLE_ITO_MP_TEST

上述的 Compile Option 是用以 Enable 在 Touch Device Driver 代碼中，針對 MStar 觸控 IC 支援整機 MP Test 功能的相關代碼處理。目前從 Touch Device Driver v3.0.0.0 版開始之後的版本，MStar 自容屏觸控 IC，例如 ：MSG21xxA 和 MSG22xx 已支援整機 Open Test 及 Short Test 功能。另外，MStar 互容屏觸控 IC，例如 ：MSG26xxM 和 MSG28xx 也已支援整機 Open Test 及 Short Test 功能 (此 Compile Option 預設值是 Disable，請依據客戶需求決定是否 Enable)。

至於如何使用上述 MP Test 功能的操作說明(目前僅提供 MSG21xxA 和 MSG22xx 和 MSG28xx 的 MP Test 說明文件)，要麻煩 FAE 同仁向文管中心(DCC)申請 "mstar_ctp_phone_level_mp_test_user_guideline_internal.pdf" (給 FAE 同仁使用)或 "mstar_ctp_phone_level_mp_test_user_guideline_customer.pdf" (給客戶使用)文件的最新版本，內有詳細說明。

2.1.1.17. #define CONFIG_ENABLE_MP_TEST_ITEM_FOR_2R_TRIANGLE

當 Compile Option #define CONFIG_ENABLE_ITO_MP_TEST 有被 Enable 的前提下，且目前客戶專案所使用的觸控 IC 是自容屏觸控 IC(MSG21xxA/MSG22xx)，請依據該觸控 IC 所搭配的 TP 的 Pattern 來決定是否要 Enable 此 Compile Option。如果 TP 的 Pattern 是 2R Triangle Pattern(分區豎三角)的話，請 Enable 此 Compile Option。如果 TP 的 Pattern 是 Horizontal Triangle Pattern(橫三角)的話，請 Disable 此 Compile Option。

2.1.1.18. #define CONFIG_UPDATE_FIRMWARE_BY_SW_ID

如果上述的 Compile Option 被 Enable，那麼手機在開機時就會透過 SW ID 的機制來檢查是否需要更新 TP Firmware。目前從 Touch Device Driver v3.0.0.0 版開始之後的版本，都已支援 MStar 的互容屏(MSG26xxM/MSG28xx/MSG58xx)及自容屏(MSG21xxA/MSG22xx)觸控 IC 的 SW ID 功能(此 Compile Option 預設值是 Disable，請依據客戶需求決定是否 Enable)。

至於如何使用上述 SW ID 功能的操作說明(目前僅提供 MSG22xx 和 MSG26xxM 和 MSG28xx 的 SW ID 說明文件)，要麻煩 FAE 同仁向文管中心(DCC)申請 "mstar_ctp_sw_id_user_guideline.pdf"文件的最新版本，內有詳細說明。

2.1.1.19. #define CONFIG_UPDATE_FIRMWARE_BY_TWO_DIMENSIONAL_ARRAY

如果 Compile Option: CONFIG_UPDATE_FIRMWARE_BY_TWO_DIMENSIONAL_ARRAY 被 Enable，那麼要用來更新的 Firmware Bin 檔必須被轉換成一個二維陣列的格式才能在 Touch Device Driver 中使用。

如果 Compile Option: CONFIG_UPDATE_FIRMWARE_BY_TWO_DIMENSIONAL_ARRAY 被 Disable，那麼要用來更新的 Firmware Bin 檔必須被轉換成一個一維陣列的格式才能在 Touch Device Driver 中使用(此 Compile Option 預設值是 Disable)。

另外，要特別注意的是因為 MSG22xx 的 Firmware Bin 檔是 48.5K，只能以一維陣列的方式來儲存。

2.1.1.20. #define CONFIG_ENABLE_HOTKNOT

上述的 Compile Option 是用以 Enable 在 Touch Device Driver 代碼中，針對 MStar 觸控 IC 支援 HotKnot 功能的相關代碼處理。目前僅互容屏觸控 IC(MSG28xx)支援 HotKnot 功能，其它互容屏(MSG26xxM)及自容屏(MSG21xxA/MSG22xx)觸控 IC 都不支援(此 Compile Option 預設值是 Disable)。

2.1.1.21. #define CONFIG_ENABLE_PROXIMITY_DETECTION

上述的 Compile Option 是用以 Enable 在 Touch Device Driver 代碼中，針對 MStar 觸控 IC 支援 Proximity Detection 功能的相關代碼處理。目前互容屏(MSG26xxM/MSG28xx/MSG58xx)及自容屏(MSG21xxA/MSG22xx)觸控 IC 都支援此功能(此 Compile Option 預設值是 Disable)。

2.1.1.22. #define CONFIG_ENABLE_NOTIFIER_FB

上述的 Compile Option 僅針對 SPRD 平台或 QCOM 平台使用，不包括 MTK 平台。在手機熄屏或亮屏時，Touch Device Driver 接收 Linux 系統發送的 Suspend 或 Resume 通知，必須使用 Notify Callback 的方式來接收通知。如果客戶專案用的是 SPRD 平台或 QCOM 平台，此 Compile Option 必須 Enable(此 Compile Option 預設值是 Disable)。

2.1.1.23. #define CONFIG_ENABLE_COUNT_REPORT_RATE

上述的 Compile Option 是用以 Enable 在 Touch Device Driver 代碼中，計算 TP Firmware 報點率的相關代碼處理。這是 TP Firmware 同仁偶而會用來分析報點率問題，需搭配 MStar 的 MTPTool APK 一起使用，客戶專案不會用到(此 Compile Option 預設值是 Enable)。

2.1.1.24. #define CONFIG_ENABLE_GLOVE_MODE

上述的 Compile Option 是用以 Enable 在 Touch Device Driver 代碼中，針對 MStar 觸控 IC 支援手套模式功能的相關代碼處理。目前僅互容屏觸控 IC(MSG28xx)支援手套模式功能，其它互容屏(MSG26xxM)及自容屏(MSG21xxA/MSG22xx)觸控 IC 都不支援(此 Compile Option 預設值是 Disable)。

2.1.1.25. #define CONFIG_ENABLE_LEATHER_SHEATH_MODE

上述的 Compile Option 是用以 Enable 在 Touch Device Driver 代碼中，針對 MStar 觸控 IC 支援皮套模式功能的相關代碼處理。目前僅互容屏觸控 IC(MSG28xx)支援皮套模式功能，其它互容屏(MSG26xxM)及自容屏(MSG21xxA/MSG22xx)觸控 IC 都不支援(此 Compile Option 預設值是 Disable)。

2.1.1.26. #define CONFIG_ENABLE_JNI_INTERFACE

上述的 Compile Option 是用以 Enable 在 Touch Device Driver 代碼中，可以透過 JNI 接口機制跟 MStar 的 MTPTool

APK 溝通的相關代碼處理(此 Compile Option 預設值是 Enable)。

2.1.1.27. #define CONFIG_ENABLE_CHARGER_DETECTION

上述的 Compile Option 是用以 Enable 在 Touch Device Driver 代碼中，針對 MStar 觸控 IC 支援 Charger Detection 機制的相關代碼處理。主要目的是為了讓 Touch Device Driver 可以通知 TP Firmware 目前在手機上是否有插拔充電器的狀態，讓 TP Firmware 針對有插充電器的狀況下，所引起的 Noise 干擾作特別處理(此 Compile Option 預設值是 Disable)。

2.1.1.28. #define CONFIG_ENABLE_ESD_PROTECTION

上述的 Compile Option 是用以 Enable 在 Touch Device Driver 代碼中，針對 MStar 觸控 IC 支援 ESD Protection 機制的相關代碼處理(此 Compile Option 預設值是 Disable)。

2.1.1.29. #define CONFIG_ENABLE_TYPE_B_PROTOCOL

上述的 Compile Option 是用以 Enable 在 Touch Device Driver 代碼中，針對 MStar 觸控 IC 支援 Multi-Touch Type A/Type B 報點機制的相關代碼處理。如果上述的 Compile Option 被 Enable，表示 Touch Device Driver 使用 Multi-Touch Type B 報點機制報點給 Linux 系統的 Input Sub-System。否則，表示 Touch Device Driver 使用 Multi-Touch Type A 報點機制報點給 Linux 系統的 Input Sub-System。

2.1.2 Constant Value

下列針對 MStar_drv_common.h 中較重要的 Constant Value 定義作進一步說明，其餘部分請直接參照 MStar_drv_common.h。

2.1.2.1. Chip Type

```
#define CHIP_TYPE_MSG21XX     (0x01) // EX. MSG2133
#define CHIP_TYPE_MSG21XXA    (0x02) // EX. MSG2133A/MSG2138A
#define CHIP_TYPE_MSG26XXM    (0x03) // EX. MSG2633M
#define CHIP_TYPE_MSG22XX     (0x7A) // EX. MSG2238/MSG2256
#define CHIP_TYPE_MSG28XX     (0x85) // EX. MSG2835/MSG2836/MSG2840/MSG2856/MSG5835/MSG5846
```

上述是 MStar 自容屏(EX. MSG21xxA/MSG22xx)和互容屏(EX. MSG26xxM/MSG28xx/MSG58xx)觸控 IC 的 Chip ID，Touch Device Driver 會透過 DrvFwCtrlGetChipType()從 Touch IC 上取得 Chip ID，並依此來決定因應不同型號的觸控 IC，觸控裝置驅動代碼在某些功能上的處理也會有所不同。因此請勿更改上述的四個 Chip ID 設定值。

2.1.2.2. Width & Height for Touch Panel

```
#define TOUCH_SCREEN_X_MAX     (480) //LCD_WIDTH
#define TOUCH_SCREEN_Y_MAX     (854) //LCD_HEIGHT
```

上述是用來設定 Touch Panel 的寬度和高度範圍，請依客戶專案所使用 Touch Panel 的尺寸確定其寬度和高度的設定值。

2.1.2.3. Authority for proc File Node

#define PROCFS_AUTHORITY (0666)

上述是用來設定 Touch Device Driver 跟上層 APK 間透過 procfs 虛擬檔案系統機制溝通的 proc 節點權限，預設值 0666 代表有 read 跟 write 的權限。

2.1.2.4. Wakeup Mode for Gesture Wakeup

```
#ifdef CONFIG_ENABLE_GESTURE_WAKEUP
#define GESTURE_WAKEUP_MODE_DOUBLE_CLICK_FLAG      0x00000001      //0000 0000 0000 0000 0000 0000 0000 0001
#define GESTURE_WAKEUP_MODE_UP_DIRECT_FLAG         0x00000002      //0000 0000 0000 0000 0000 0000 0000 0010
#define GESTURE_WAKEUP_MODE_DOWN_DIRECT_FLAG       0x00000004      //0000 0000 0000 0000 0000 0000 0000 0100
#define GESTURE_WAKEUP_MODE_LEFT_DIRECT_FLAG       0x00000008      //0000 0000 0000 0000 0000 0000 0000 1000
#define GESTURE_WAKEUP_MODE_RIGHT_DIRECT_FLAG      0x00000010      //0000 0000 0000 0000 0000 0000 0001 0000
#define GESTURE_WAKEUP_MODE_m_CHARACTER_FLAG       0x00000020      //0000 0000 0000 0000 0000 0000 0010 0000
#define GESTURE_WAKEUP_MODE_W_CHARACTER_FLAG       0x00000040      //0000 0000 0000 0000 0000 0000 0100 0000
#define GESTURE_WAKEUP_MODE_C_CHARACTER_FLAG       0x00000080      //0000 0000 0000 0000 0000 0000 1000 0000
#define GESTURE_WAKEUP_MODE_e_CHARACTER_FLAG       0x00000100      //0000 0000 0000 0000 0000 0001 0000 0000
#define GESTURE_WAKEUP_MODE_V_CHARACTER_FLAG       0x00000200      //0000 0000 0000 0000 0000 0010 0000 0000
#define GESTURE_WAKEUP_MODE_O_CHARACTER_FLAG       0x00000400      //0000 0000 0000 0000 0000 0100 0000 0000
#define GESTURE_WAKEUP_MODE_S_CHARACTER_FLAG       0x00000800      //0000 0000 0000 0000 0000 1000 0000 0000
#define GESTURE_WAKEUP_MODE_Z_CHARACTER_FLAG       0x00001000      //0000 0000 0000 0000 0001 0000 0000 0000
#define GESTURE_WAKEUP_MODE_RESERVE1_FLAG          0x00002000      //0000 0000 0000 0000 0010 0000 0000 0000
#define GESTURE_WAKEUP_MODE_RESERVE2_FLAG          0x00004000      //0000 0000 0000 0000 0100 0000 0000 0000
#define GESTURE_WAKEUP_MODE_RESERVE3_FLAG          0X00008000      //0000 0000 0000 0000 1000 0000 0000 0000

#ifdef CONFIG_SUPPORT_64_TYPES_GESTURE_WAKEUP_MODE
#define GESTURE_WAKEUP_MODE_RESERVE4_FLAG          0x00010000      //0000 0000 0000 0001 0000 0000 0000 0000
```

```
#define GESTURE_WAKEUP_MODE_RESERVE5_FLAG          0x00020000    //0000 0000 0000 0010  0000 0000 0000 0000
#define GESTURE_WAKEUP_MODE_RESERVE6_FLAG          0x00040000    //0000 0000 0000 0100  0000 0000 0000 0000
#define GESTURE_WAKEUP_MODE_RESERVE7_FLAG          0x00080000    //0000 0000 0000 1000  0000 0000 0000 0000
#define GESTURE_WAKEUP_MODE_RESERVE8_FLAG          0x00100000    //0000 0000 0001 0000  0000 0000 0000 0000
#define GESTURE_WAKEUP_MODE_RESERVE9_FLAG          0x00200000    //0000 0000 0010 0000  0000 0000 0000 0000
#define GESTURE_WAKEUP_MODE_RESERVE10_FLAG         0x00400000    //0000 0000 0100 0000  0000 0000 0000 0000
#define GESTURE_WAKEUP_MODE_RESERVE11_FLAG         0x00800000    //0000 0000 1000 0000  0000 0000 0000 0000
#define GESTURE_WAKEUP_MODE_RESERVE12_FLAG         0x01000000    //0000 0001 0000 0000  0000 0000 0000 0000
#define GESTURE_WAKEUP_MODE_RESERVE13_FLAG         0x02000000    //0000 0010 0000 0000  0000 0000 0000 0000
#define GESTURE_WAKEUP_MODE_RESERVE14_FLAG         0x04000000    //0000 0100 0000 0000  0000 0000 0000 0000
#define GESTURE_WAKEUP_MODE_RESERVE15_FLAG         0x08000000    //0000 1000 0000 0000  0000 0000 0000 0000
#define GESTURE_WAKEUP_MODE_RESERVE16_FLAG         0x10000000    //0001 0000 0000 0000  0000 0000 0000 0000
#define GESTURE_WAKEUP_MODE_RESERVE17_FLAG         0x20000000    //0010 0000 0000 0000  0000 0000 0000 0000
#define GESTURE_WAKEUP_MODE_RESERVE18_FLAG         0x40000000    //0100 0000 0000 0000  0000 0000 0000 0000
#define GESTURE_WAKEUP_MODE_RESERVE19_FLAG         0x80000000    //1000 0000 0000 0000  0000 0000 0000 0000

#define GESTURE_WAKEUP_MODE_RESERVE20_FLAG         0x00000001    //0000 0000 0000 0000  0000 0000 0000 0001
#define GESTURE_WAKEUP_MODE_RESERVE21_FLAG         0x00000002    //0000 0000 0000 0000  0000 0000 0000 0010
#define GESTURE_WAKEUP_MODE_RESERVE22_FLAG         0x00000004    //0000 0000 0000 0000  0000 0000 0000 0100
#define GESTURE_WAKEUP_MODE_RESERVE23_FLAG         0x00000008    //0000 0000 0000 0000  0000 0000 0000 1000
#define GESTURE_WAKEUP_MODE_RESERVE24_FLAG         0x00000010    //0000 0000 0000 0000  0000 0001 0000 0000
#define GESTURE_WAKEUP_MODE_RESERVE25_FLAG         0x00000020    //0000 0000 0000 0000  0000 0010 0000 0000
```

```
#define GESTURE_WAKEUP_MODE_RESERVE26_FLAG          0x00000040      //0000 0000 0000 0000    0000 0100 0000
#define GESTURE_WAKEUP_MODE_RESERVE27_FLAG          0x00000080      //0000 0000 0000 0000    0000 1000 0000
#define GESTURE_WAKEUP_MODE_RESERVE28_FLAG          0x00000100      //0000 0000 0000 0000    0001 0000 0000
#define GESTURE_WAKEUP_MODE_RESERVE29_FLAG          0x00000200      //0000 0000 0000 0000    0010 0000 0000
#define GESTURE_WAKEUP_MODE_RESERVE30_FLAG          0x00000400      //0000 0000 0000 0000    0100 0000 0000
#define GESTURE_WAKEUP_MODE_RESERVE31_FLAG          0x00000800      //0000 0000 0000 0000    1000 0000 0000
#define GESTURE_WAKEUP_MODE_RESERVE32_FLAG          0x00001000      //0000 0000 0000 0000    0001 0000 0000 0000
#define GESTURE_WAKEUP_MODE_RESERVE33_FLAG          0x00002000      //0000 0000 0000 0000    0010 0000 0000 0000
#define GESTURE_WAKEUP_MODE_RESERVE34_FLAG          0x00004000      //0000 0000 0000 0000    0100 0000 0000 0000
#define GESTURE_WAKEUP_MODE_RESERVE35_FLAG          0X00008000      //0000 0000 0000 0000    1000 0000 0000 0000
#define GESTURE_WAKEUP_MODE_RESERVE36_FLAG          0x00010000      //0000 0000 0000 0001    0000 0000 0000 0000
#define GESTURE_WAKEUP_MODE_RESERVE37_FLAG          0x00020000      //0000 0000 0000 0010    0000 0000 0000 0000
#define GESTURE_WAKEUP_MODE_RESERVE38_FLAG          0x00040000      //0000 0000 0000 0100    0000 0000 0000 0000
#define GESTURE_WAKEUP_MODE_RESERVE39_FLAG          0x00080000      //0000 0000 0000 1000    0000 0000 0000 0000
#define GESTURE_WAKEUP_MODE_RESERVE40_FLAG          0x00100000      //0000 0000 0001 0000    0000 0000 0000 0000
#define GESTURE_WAKEUP_MODE_RESERVE41_FLAG          0x00200000      //0000 0000 0010 0000    0000 0000 0000 0000
#define GESTURE_WAKEUP_MODE_RESERVE42_FLAG          0x00400000      //0000 0000 0100 0000    0000 0000 0000 0000
#define GESTURE_WAKEUP_MODE_RESERVE43_FLAG          0x00800000      //0000 0000 1000 0000    0000 0000 0000 0000
#define GESTURE_WAKEUP_MODE_RESERVE44_FLAG          0x01000000      //0000 0001 0000 0000    0000 0000 0000 0000
#define GESTURE_WAKEUP_MODE_RESERVE45_FLAG          0x02000000      //0000 0010 0000 0000    0000 0000 0000 0000
#define GESTURE_WAKEUP_MODE_RESERVE46_FLAG          0x04000000      //0000 0100 0000 0000    0000 0000 0000 0000
#define GESTURE_WAKEUP_MODE_RESERVE47_FLAG          0x08000000      //0000 1000 0000 0000    0000 0000 0000 0000
```

```
#define GESTURE_WAKEUP_MODE_RESERVE48_FLAG          0x10000000      //0001 0000 0000 0000      0000 0000 0000 0000
#define GESTURE_WAKEUP_MODE_RESERVE49_FLAG          0x20000000      //0010 0000 0000 0000      0000 0000 0000 0000
#define GESTURE_WAKEUP_MODE_RESERVE50_FLAG          0x40000000      //0100 0000 0000 0000      0000 0000 0000 0000
#define GESTURE_WAKEUP_MODE_RESERVE51_FLAG          0x80000000      //1000 0000 0000 0000      0000 0000 0000 0000
#endif //CONFIG_SUPPORT_64_TYPES_GESTURE_WAKEUP_MODE

#define GESTURE_WAKEUP_PACKET_LENGTH      (6)
#endif //CONFIG_ENABLE_GESTURE_WAKEUP
```

上述是MStar觸控IC目前已支援的13種不同手勢喚醒模式設定值(雙擊/向上滑動/向下滑動/向左滑動/向右滑動/m字/W字/C字/e字/V字/O字/S字/Z字)。另外，TP Firmware可支援最多16種(預設)或64種(如果compile option CONFIG_SUPPORT_64_TYPES_GESTURE_WAKEUP_MODE有被Enable)的手勢喚醒模式。目前MStar的互容屏觸控IC MSG28xx和自容屏觸控IC MSG22xx已經支援16種和64種手勢喚醒模式，而互容屏觸控IC MSG26xxM和自容屏觸控IC MSG21xxA則僅支援16種手勢喚醒模式。如果手勢喚醒功能有被Enable，Touch Device Driver會依據被Enable手勢喚醒模式的種類，設定上述的相對應值至TP Firmware。另外，上述的GESTURE_WAKEUP_PACKET_LENGTH是MSG22xx和MSG26xxM和MSG28xx的TP Firmware，採用0xA7的Packet Format回報手勢喚醒的資訊到Touch Device Driver，該類型的Gesture Wakeup Packet長度固定為6 Byte大小。

因此請勿更改上述的13種Gesture Wakeup Mode和Gesture Wakeup Packet的長度設定值。

2.1.2.5. Packet Type for 0xA7 Packet

```
#define PACKET_TYPE_TOOTH_PATTERN     (0x20)
#define PACKET_TYPE_GESTURE_WAKEUP    (0x50)
#define PACKET_TYPE_GESTURE_DEBUG             (0x51)
#define PACKET_TYPE_GESTURE_INFORMATION       (0x52)
#define PACKET_TYPE_ESD_CHECK_HW_RESET(0x60)
```

上述分別是當 TP Firmware 使用 0xA7 Packet Format 回報資訊給 Touch Device Driver 時， 讓 Touch Device Driver 可以透過內含在 Packet 中的 Packet Type 判斷 TP Firmware 回報的 Packet 是一個 Debug Mode Packet 或一個 Gesture Wakeup Packet 或一個 Gesture Debug Mode Packett 或一個 Gesture Information Mode Packet。因此請勿更改上述的五個 Packet Type 設定值。

2.1.2.6. Maximum Input Register Number

```
#define MAX_DEBUG_REGISTER_NUM        (10)
```

目前 Touch Device Driver 有支援透過 Android 的 adb Tool 下 Command 觸發 Touch Device Driver 去讀取 Touch IC 上常用來偵錯使用的 Register 數值， 讓 Firmware 同仁可以自行輸入想查詢的 Register 數值。預設最多可輸入 10 個 Registers(Bank+Address)，之後可依 Firmware 同仁偵錯需求自行擴充。

2.1.2.7. Maximum Temp Update Firmware Buffer

```
#define MAX_UPDATE_FIRMWARE_BUFFER_SIZE       (130)
```

透過 MStar 提供的 MTPTool APK 來觸發 Touch Device Driver 更新 TP Firmware， 在 Touch Device Driver 必須準備一暫存的 Data Buffer 來儲存 APK 傳送過來要更新的 TP Firmware 資料。 因為 Touch Device Driver 要能兼容下述觸控 IC(MSG26xxM：40KB， MSG28xx/MSG58xx：130 KB， MSG21xxA：33 KB， MSG22xx：48.5 KB)的更新 TP Firmware 處理， 所以此暫存的 Data Buffer 至少要有 130KB。 因此請勿更改上述的設定值。

2.1.2.8. BIT0~BIT15 Value

```
#define BIT0  (1<<0)  // 0x0001
#define BIT1  (1<<1)  // 0x0002
#define BIT2  (1<<2)  // 0x0004
#define BIT3  (1<<3)  // 0x0008
#define BIT4  (1<<4)  // 0x0010
#define BIT5  (1<<5)  // 0x0020
#define BIT6  (1<<6)  // 0x0040
#define BIT7  (1<<7)  // 0x0080
#define BIT8  (1<<8)  // 0x0100
#define BIT9  (1<<9)  // 0x0200
#define BIT10 (1<<10) // 0x0400
#define BIT11 (1<<11) // 0x0800
#define BIT12 (1<<12) // 0x1000
#define BIT13 (1<<13) // 0x2000
#define BIT14 (1<<14) // 0x4000
```

#define BIT15 (1<<15) // 0x8000

上述是 Touch Device Driver 中常用的 BIT0~BIT15 設定值。 因此請勿更改上述的 15 個設定值。

2.1.2.9. Enable/Disable Kernel Log for Touch Device Driver

#define CONFIG_TOUCH_DRIVER_DEBUG_LOG_LEVEL (1)

上述的 Flag 是用來讓 FAE 同仁或客戶在合入 Touch Device Driver 代碼後，如果有某些功能無法正常運作(EX. 報點或更新觸控固件)，可以透過吐 Kernel Log 的方式偵錯，用以確認 Touch Device Driver 代碼是否有問題. 通常在手機開發階段客戶會 Enable 此 Flag 方便偵錯，等到手機要正式出貨階段客戶會 Disable 此 Flag(此 Flag 預設值是 1。1 是 Enable，0 是 Disable)。

2.1.3 Function

下列針對 MStar_drv_common.h 中較重要的 Function 宣告作進一步說明，其餘部分請直接參照 MStar_drv_common.h 和 MStar_drv_common.c。

2.1.3.1. CRC Calculation

extern u32 DrvCommonCrcDoReflect(u32 nRef, s8 nCh);
extern u32 DrvCommonCrcGetValue(u32 nText, u32 nPrevCRC);
extern void DrvCommonCrcInitTable(void);

上述的函式是 Touch Device Driver 用來計算 TP Firmware 的 Main Block 或 Info Block 的 CRC，藉此確認透過 APK 或 SW ID 機制更新的 TP Firmware 是否正確無誤。

2.1.3.2. Checksum Calculation

extern u8 DrvCommonCalculateCheckSum(u8 *pMsg, u32 nLength);

上述的函式是 Touch Device Driver 用以計算 TP Firmware 透過 IRQ Interrupt 回報的每一筆報點資料的 Checksum，藉此 TP Firmware 送給 Touch Device Driver 的每一筆報點資料是否正確無誤。

2.2. MStar_drv_utility_adaption.h & MStar_drv_utility_adaption.c

在整個 Touch Device Driver 代碼中， MStar 的觸控 IC 或 TP Firmware 和 Touch Device Driver 間的常用 I2C read/write 函式及在 MTK 平台才有可能會使用的 DMA Mode 機制， 都定義在 MStar_drv_utility_adaption.h 和 MStar_drv_utility_adaption.c 中。詳述如下：

2.2.1 Function

底下針對 MStar_drv_utility_adaption.h 中較重要的 Function 宣告作進一步說明， 其餘請直接參照 MStar_drv_utility_adaption.h 和 MStar_drv_utility_adaption.c。

2.2.1.1. I2C Read/Write for TP Firmware or Touch IC

```
extern s32 IicWriteData(u8 nSlaveId, u8* pBuf, u16 nSize);
extern s32 IicReadData(u8 nSlaveId, u8* pBuf, u16 nSize);
```

上述的函式是 Touch Device Driver 透過 I2C 走 SMBUS 跟 TP Firmware 或走 DBBUS 跟 Touch IC 進行資料傳輸的常用函式。

2.2.1.2. I2C Read/Write for Register of Touch IC

```
extern u16 RegGet16BitValue(u16 nAddr);
extern u8 RegGetLByteValue(u16 nAddr);
extern u8 RegGetHByteValue(u16 nAddr);
extern void RegSet16BitValue(u16 nAddr, u16 nData);
extern void RegSetLByteValue(u16 nAddr, u8 nData);
extern void RegSetHByteValue(u16 nAddr, u8 nData);
extern void RegSet16BitValueOn(u16 nAddr, u16 nData);
extern void RegSet16BitValueOff(u16 nAddr, u16 nData);
extern u16   RegGet16BitValueByAddressMode(u16 nAddr, AddressMode_e eAddressMode);
extern void RegSet16BitValueByAddressMode(u16 nAddr, u16 nData, AddressMode_e eAddressMode);
extern void RegMask16BitValue(u16 nAddr, u16 nMask, u16 nData, AddressMode_e eAddressMode);
```

上述的函式是 Touch Device Driver 透過 I2C 走 DBBUS 用來跟 Touch IC 上的 Register 進行資料傳輸的常用函式。

2.2.1.3. Switch I2C to Use DBBUS

```
extern s32 DbBusEnterSerialDebugMode(void);
extern void DbBusExitSerialDebugMode(void);
extern void DbBusIICUseBus(void);
extern void DbBusIICNotUseBus(void);
extern void DbBusIICReshape(void);
extern void DbBusStopMCU(void);
extern void DbBusNotStopMCU(void);
```

上述的函式是 Touch Device Driver 透過 I2C 走 DBBUS 用來跟 Touch IC 進行資料傳輸前，必須先透過上述的對應函式將 I2C 切換成走 DBBUS。在資料傳輸結束後，也必須先透過上述的對應函式將 I2C 切換成不走 DBBUS。

2.2.1.4. DMA Mode

```
#ifdef CONFIG_ENABLE_DMA_IIC
extern void DmaAlloc(void);
extern void DmaReset(void);
extern void DmaFree(void);
#endif //CONFIG_ENABLE_DMA_IIC
```

上述的函式是使用在 I2C 的 read/write 具備一次最多 8 Byte 大小限制的 MTK 平台 BB Chip 上。 在 Touch Device Driver 的代碼中， DmaAlloc()只需在手機開機時、Touch Device Driver 進行註冊動作時執行一次。 DmaFree()只需在手機關機時、Touch Device Driver 進行卸載動作時執行一次。 如果跟 TP Firmware 或 Touch IC 之間的資料傳輸一次超過 8 Byte， 必須在該 I2C 命令的前面先呼叫 DmaReset()。在 Touch Device Driver v3.0.0.0 版內， 有 I2C 的 read/write 超過 8 Byte 大小限制的命令， 皆已加上相關處理。 另外， 在 IicWriteData()和 IicReadData()內也有加上跟 DMA Mode 的相關處理， 詳細可參照 MStar_drv_utility_adaption.c。

2.3. MStar_drv_main.h & MStar_drv_main.c

從 Touch Device Driver v3.0.0.0 版開始, 在整個 Touch Device Driver 代碼中， MStar 的 APK 和 Touch Device Driver 之間已改為透過 Linux 的 proc 虛擬檔案系統機制進行資料的傳輸。 APK 藉此機制驅動 Touch Device Driver(EX. 更新固件/顯示固件版本等)。 APK 和 Touch Device Driver 間使用到的所有 proc 虛擬檔案節點和函式都定義在 MStar_drv_main.h 和 MStar_drv_main.c 中。詳述如下：

2.3.1 Function

底下針對 MStar_drv_main.h 中較重要的 Function 宣告作進一步說明，其餘請直接參照 MStar_drv_main.h 和 MStar_drv_main.c。

2.3.1.1. Device Driver Initialization

extern s32 DrvMainTouchDeviceInitialize(void);

上述的函式是 Touch Device Driver 把一部份當手機開機時， MStar 觸控裝置驅動的初始化動作都集中在此函式內處理， 如 proc 虛擬檔案節點的創建， 透過下 Command 跟 Touch IC 取得 Chip Type。

2.3.1.2. Display Customer/Platform Firmware Version

extern ssize_t DrvMainProcfsCustomerFirmwareVersionRead(struct file *pFile, char __user *pBuffer, size_t nCount, loff_t *pPos);
extern ssize_t DrvMainProcfsCustomerFirmwareVersionWrite(struct file *pFile, const char __user *pBuffer, size_t nCount, loff_t *pPos);
extern ssize_t DrvMainProcfsPlatformFirmwareVersionRead(struct file *pFile, char __user *pBuffer, size_t nCount, loff_t *pPos);
extern ssize_t DrvMainProcfsPlatformFirmwareVersionWrite(struct file *pFile, const char __user *pBuffer, size_t nCount, loff_t *pPos);

上述的函式是 APK 透過對 proc 虛擬檔案節點的 read/write 動作， 藉此觸發 Touch Device Driver 向 TP Firmware 查詢 Customer Firmware Version 及 Platform Firmware Version。

2.3.1.3. Update Firmware

extern ssize_t DrvMainProcfsFirmwareDataRead(struct file *pFile, char __user *pBuffer, size_t nCount, loff_t *pPos);
extern ssize_t DrvMainProcfsFirmwareDataWrite(struct file *pFile, const char __user *pBuffer, size_t nCount, loff_t *pPos);
extern ssize_t DrvMainProcfsFirmwareUpdateRead(struct file *pFile, char __user *pBuffer, size_t nCount, loff_t *pPos);
extern ssize_t DrvMainProcfsFirmwareUpdateWrite(struct file *pFile, const char __user *pBuffer, size_t nCount, loff_t *pPos);
extern ssize_t DrvMainProcfsSdCardFirmwareUpdateRead(struct file *pFile, char __user *pBuffer, size_t nCount, loff_t *pPos);
extern ssize_t DrvMainProcfsSdCardFirmwareUpdateWrite(struct file *pFile, const char __user *pBuffer, size_t nCount, loff_t *pPos);

上述的函式是 APK 透過對 proc 虛擬檔案節點的 read/write 動作，藉此觸發 Touch Device Driver 更新使用者從手機的 T Card 上所選取的 TP Firmware。另外，也可以觸發 Touch Device Driver 去更新內存或 T Card 上一固定路徑下且固定檔名的 TP Firmware。

2.3.1.4. Gesture Wakeup

```
#ifdef CONFIG_ENABLE_GESTURE_WAKEUPextern ssize_t DrvMainProcfsGestureWakeupModeRead(struct file *pFile, char __user *pBuffer, size_t nCount, loff_t *pPos);
extern ssize_t DrvMainProcfsGestureWakeupModeWrite(struct file *pFile, const char __user *pBuffer, size_t nCount, loff_t *pPos);

#ifdef CONFIG_ENABLE_GESTURE_DEBUG_MODE
extern ssize_t DrvMainProcfsGestureDebugModeRead(struct file *pFile, char __user *pBuffer, size_t nCount, loff_t *pPos);
extern ssize_t DrvMainProcfsGestureDebugModeWrite(struct file *pFile, const char __user *pBuffer, size_t nCount, loff_t *pPos);
extern ssize_t DrvMainKObjectGestureDebugShow(struct kobject *pKObj, struct kobj_attribute *pAttr, char *pBuf);
extern ssize_t DrvMainKObjectGestureDebugStore(struct kobject *pKObj, struct kobj_attribute *pAttr, const char *pBuf, size_t nCount);
#endif //CONFIG_ENABLE_GESTURE_DEBUG_MODE

#ifdef CONFIG_ENABLE_GESTURE_INFORMATION_MODE
extern ssize_t DrvMainProcfsGestureInforModeRead(struct file *pFile, char __user *pBuffer, size_t nCount, loff_t *pPos);
extern ssize_t DrvMainProcfsGestureInforModeWrite(struct file *pFile, const char __user *pBuffer, size_t nCount, loff_t *pPos);
#endif //CONFIG_ENABLE_GESTURE_INFORMATION_MODE

#endif //CONFIG_ENABLE_GESTURE_WAKEUP
```

上述的函式是 APK 透過對 proc 虛擬檔案節點的 read/write 動作， 藉此觸發 Touch Device Driver 跟 TP Firmware 設定要 Enable/Disable 哪些手勢喚醒的手勢， 以驗證手勢喚醒功能是否能正常運作。另外，還有 Gesture Debug Mode(TP Firmware 用來 Debug 手勢喚醒問題)及 Gesture Information Mode(手勢回顯功能)的相關函式，用來串接 Touch Device Driver 跟上層 APK 間的溝通。

2.3.1.5. MP Test

```
#ifdef CONFIG_ENABLE_ITO_MP_TEST
extern ssize_t DrvMainProcfsMpTestRead(struct file *pFile, char __user *pBuffer, size_t nCount, loff_t *pPos);
extern ssize_t DrvMainProcfsMpTestWrite(struct file *pFile, const char __user *pBuffer, size_t nCount, loff_t *pPos);
extern ssize_t DrvMainProcfsMpTestLogRead(struct file *pFile, char __user *pBuffer, size_t nCount, loff_t *pPos);
extern ssize_t DrvMainProcfsMpTestLogWrite(struct file *pFile, const char __user *pBuffer, size_t nCount, loff_t *pPos);
```

```
extern ssize_t DrvMainProcfsMpTestFailChannelRead(struct file *pFile, char __user *pBuffer, size_t nCount, loff_t *pPos);
extern ssize_t DrvMainProcfsMpTestFailChannelWrite(struct file *pFile, const char __user *pBuffer, size_t nCount, loff_t *pPos);
extern ssize_t DrvMainProcfsMpTestScopeRead(struct file *pFile, char __user *pBuffer, size_t nCount, loff_t *pPos);
extern ssize_t DrvMainProcfsMpTestScopeWrite(struct file *pFile, const char __user *pBuffer, size_t nCount, loff_t *pPos);

#ifdef CONFIG_ENABLE_CHIP_TYPE_MSG28XX
extern ssize_t DrvMainProcfsMpTestLogAllRead(struct file *pFile, char __user *pBuffer, size_t nCount, loff_t *pPos);
extern ssize_t DrvMainProcfsMpTestLogAllWrite(struct file *pFile, const char __user *pBuffer, size_t nCount, loff_t *pPos);
#endif //CONFIG_ENABLE_CHIP_TYPE_MSG28XX
#endif //CONFIG_ENABLE_ITO_MP_TEST
```

上述的函式是 APK 透過對 proc 虛擬檔案節點的 read/write 動作， 藉此觸發 Touch Device Driver 進行整機上的 MP Test 測試， 並將測試結果顯示在 APK 的畫面。

2.3.1.6. Firmware Debug Mode Data Log

```
#ifdef CONFIG_ENABLE_FIRMWARE_DATA_LOG
extern ssize_t DrvMainProcfsFirmwareModeRead(struct file *pFile, char __user *pBuffer, size_t nCount, loff_t *pPos);
extern ssize_t DrvMainProcfsFirmwareModeWrite(struct file *pFile, const char __user *pBuffer, size_t nCount, loff_t *pPos);
extern ssize_t DrvMainProcfsFirmwareSensorRead(struct file *pFile, char __user *pBuffer, size_t nCount, loff_t *pPos);
extern ssize_t DrvMainProcfsFirmwareSensorWrite(struct file *pFile, const char __user *pBuffer, size_t nCount, loff_t *pPos);
extern ssize_t DrvMainProcfsFirmwarePacketHeaderRead(struct file *pFile, char __user *pBuffer, size_t nCount, loff_t *pPos);
extern ssize_t DrvMainProcfsFirmwarePacketHeaderWrite(struct file *pFile, const char __user *pBuffer, size_t nCount, loff_t *pPos);
extern ssize_t DrvMainKObjectPacketShow(struct kobject *pKObj, struct kobj_attribute *pAttr, char *pBuf);
extern ssize_t DrvMainKObjectPacketStore(struct kobject *pKObj, struct kobj_attribute *pAttr, const char *pBuf, size_t nCount);
#endif //CONFIG_ENABLE_FIRMWARE_DATA_LOG
```

上述的函式是 APK 透過對 proc 虛擬檔案節點的 read/write 動作， 讓 Touch Device Driver 可以觸發 TP Firmware 從 Demo Mode 切換到 Debug Mode， 並將 Debug Mode 資料輸出到手機 T Card 上的一個.csv 檔， 最後再透過 MStar 提供的 MxViewer 工具來回播.csv 檔， 用 UI 顯示的方式方便 TP Firmware 同仁分析觸控固件問題。

2.3.1.7. Adb Command Debug

```
extern ssize_t DrvMainProcfsFirmwareDebugRead(struct file *pFile, char __user *pBuffer, size_t nCount, loff_t *pPos);
extern ssize_t DrvMainProcfsFirmwareDebugWrite(struct file *pFile, const char __user *pBuffer, size_t nCount, loff_t *pPos);
extern ssize_t DrvMainProcfsFirmwareSetDebugValueRead(struct file *pFile, char __user *pBuffer, size_t nCount, loff_t *pPos);
extern ssize_t DrvMainProcfsFirmwareSetDebugValueWrite(struct file *pFile, const char __user *pBuffer, size_t nCount, loff_t *pPos);
extern ssize_t DrvMainProcfsFirmwareSmBusDebugRead(struct file *pFile, char __user *pBuffer, size_t nCount, loff_t *pPos);
extern ssize_t DrvMainProcfsFirmwareSmBusDebugWrite(struct file *pFile, const char __user *pBuffer, size_t nCount, loff_t *pPos);
```

上述的函式是可以透過 Android 的 adb Tool 下 Command 觸發 Touch Device Driver 去讀取 Touch IC 上常拿來偵錯用的 Register 數值， 讓 Firmware 同仁可以自行輸入想查詢哪些 Register 數值。 預設最多可輸入 10 個 Register(Bank+Address)， 之後可依 Firmware 同仁偵錯需求自行擴充。另外，還可以設值給 Touch IC 上的拿來偵錯用的 Register 或下 SmBus Command 給 TP Firmware。

2.4. MStar_drv _fw_control.h & MStar_drv _fw_control.c

在整個 Touch Device Driver 代碼中， 跟 MStar 的互容屏觸控 IC(MSG26xxM/MSG28xx/MSG58xx)和自容屏觸控 IC(MSG21xxA/MSG22xx)或觸控固件相關的功能實作，如 Get Chip Type， Get Firmware Version， Update Firmware by APK， Update Firmware by SW ID， Gesture Wakeup 等， 都定義在 MStar_drv _fw_control.h 和 MStar_drv _fw_control.c 中。 詳述如下：

2.4.1 Constant Value

下列針對 MStar_drv _fw_control.h 中較重要的 Constant Value 定義作進一步說明， 其餘請直接參照 MStar_drv _fw_control.h。

2.4.1.1. Demo Mode Packet Length

```
#define MUTUAL_DEMO_MODE_PACKET_LENGTH      (43)
#define SELF_DEMO_MODE_PACKET_LENGTH        (8)
```

上述是 MStar 互容屏觸控固件(MSG26xxM/MSG28xx/MSG58xx)或自容屏觸控固件(MSG21xxA/MSG22xx)打報點封包給 Touch Device Driver 處理的 Demo Mode Packet Length 長度設定， 目前此長度設定值為互容屏觸控 IC 是 43 Byte，而自容屏觸控 IC 是 8 Byte 是固定不變的。 因此請勿更改上述的設定值。

2.4.1.2. Maximum Finger Touch Number

```
#define MUTUAL_MAX_TOUCH_NUM              (10)
#define SELF_MAX_TOUCH_NUM                (2)
```

上述是 MStar 互容屏觸控 IC(MSG26xxM/MSG28xx/MSG58xx)或自容屏觸控 IC(MSG21xxA/MSG22xx)最多可支援同時幾指在 Touch Panel 進行觸控操作， 目前此最多手指數設定為互容屏觸控 IC 是 10，而自容屏觸控 IC 是 2 是固定不變的。 因此請勿更改上述的設定值。

2.4.1.3. Firmware Size

```
#define MSG21XXA_FIRMWARE_MAIN_BLOCK_SIZE (32) //32K
#define MSG21XXA_FIRMWARE_INFO_BLOCK_SIZE (1)   //1K
#define MSG21XXA_FIRMWARE_WHOLE_SIZE
(MSG21XXA_FIRMWARE_MAIN_BLOCK_SIZE+MSG21XXA_FIRMWARE_INFO_BLOCK_SIZE) //33K

#define MSG22XX_FIRMWARE_MAIN_BLOCK_SIZE (48)   //48K
#define MSG22XX_FIRMWARE_INFO_BLOCK_SIZE (512) //512Byte

#define MSG26XXM_FIRMWARE_MAIN_BLOCK_SIZE (32) //32K
#define MSG26XXM_FIRMWARE_INFO_BLOCK_SIZE (8) //8K
#define MSG26XXM_FIRMWARE_WHOLE_SIZE
(MSG26XXM_FIRMWARE_MAIN_BLOCK_SIZE+MSG26XXM_FIRMWARE_INFO_BLOCK_SIZE) //40K

#define MSG28XX_FIRMWARE_MAIN_BLOCK_SIZE (128) //128K
#define MSG28XX_FIRMWARE_INFO_BLOCK_SIZE (2) //2K
#define MSG28XX_FIRMWARE_WHOLE_SIZE
(MSG28XX_FIRMWARE_MAIN_BLOCK_SIZE+MSG28XX_FIRMWARE_INFO_BLOCK_SIZE) //130K
```

上述分別是 MStar 互容屏觸控固件(MSG26xxM/MSG28xx/MSG58xx)或自容屏觸控固件(MSG21xxA/MSG22xx)的 Size 大小設定， 目前上述的 Firmware 大小設定是固定不變的。因此請勿更改上述的設定值。

2.4.1.4. Firmware Mode

```
#define MSG21XXA_FIRMWARE_MODE_DEMO_MODE        (0x00)
#define MSG21XXA_FIRMWARE_MODE_DEBUG_MODE       (0x01)
#define MSG21XXA_FIRMWARE_MODE_RAW_DATA_MODE    (0x02)

#define MSG22XX_FIRMWARE_MODE_DEMO_MODE         (0x00)
#define MSG22XX_FIRMWARE_MODE_DEBUG_MODE        (0x01)
#define MSG22XX_FIRMWARE_MODE_RAW_DATA_MODE     (0x02)

#define MSG26XXM_FIRMWARE_MODE_UNKNOWN_MODE (0xFFFF)
#define MSG26XXM_FIRMWARE_MODE_DEMO_MODE      (0x0005)
#define MSG26XXM_FIRMWARE_MODE_DEBUG_MODE     (0x0105)
```

```
#define MSG28XX_FIRMWARE_MODE_UNKNOWN_MODE (0xFF)
#define MSG28XX_FIRMWARE_MODE_DEMO_MODE    (0x00)
#define MSG28XX_FIRMWARE_MODE_DEBUG_MODE   (0x01)
```

上述分別是 MStar 互容屏觸控固件(MSG26xxM/MSG28xx/MSG58xx)或自容屏觸控固件(MSG21xxA/MSG22xx)內的幾種不同 Firmware Mode 設定值， 目前上述幾種不同 Firmware Mode 設定值是固定不變的。因此請勿更改上述的設定值。

2.4.1.5. SW ID

```
#ifdef CONFIG_UPDATE_FIRMWARE_BY_SW_ID
#define UPDATE_FIRMWARE_RETRY_COUNT (2)
#endif //CONFIG_UPDATE_FIRMWARE_BY_SW_ID
```

上述是透過 SW ID 機制在手機開機時更新觸控固件的 Retry Count， 此設定值可依據客戶需求決定是否更改。

2.4.2 Enumeration Value

底下針對 MStar_drv _fw_control.h 中較重要的 Enumeration Value 定義作進一步說明，其餘請直接參照 MStar_drv _fw_control.h。

2.4.2.1. SW ID

```
/*
 * Note.
 * The following is sw id enum definition for MSG21xxA.
 * SW_ID_UNDEFINED is a reserved enum value, do not delete it or modify it.
 * Please modify the SW ID of the below enum value depends on the TP vendor that you are using.
 */
typedef enum {
    MSG21XXA_SW_ID_XXXX = 0,
    MSG21XXA_SW_ID_YYYY,
    MSG21XXA_SW_ID_UNDEFINED
} Msg21xxaSwId_e;

/*
 * Note.
 * The following is sw id enum definition for MSG22xx.
 * 0x0000 and 0xFFFF are not allowed to be defined as SW ID.
 * SW_ID_UNDEFINED is a reserved enum value, do not delete it or modify it.
 * Please modify the SW ID of the below enum value depends on the TP vendor that you are using.
 */
typedef enum {
    MSG22XX_SW_ID_XXXX = 0x0001,
    MSG22XX_SW_ID_YYYY = 0x0002,
```

```
    MSG22XX_SW_ID_UNDEFINED = 0xFFFF
} Msg22xxSwId_e;

/*
 * Note.
 * The following is sw id enum definition for MSG26xxM.
 * 0x0000 and 0xFFFF are not allowed to be defined as SW ID.
 * SW_ID_UNDEFINED is a reserved enum value, do not delete it or modify it.
 * Please modify the SW ID of the below enum value depends on the TP vendor that you are using.
 */
typedef enum {
    MSG26XXM_SW_ID_XXXX = 0x0001,
    MSG26XXM_SW_ID_YYYY = 0x0002,
    MSG26XXM_SW_ID_UNDEFINED = 0xFFFF
} Msg26xxmSwId_e;

/*
 * Note.
 * The following is sw id enum definition for MSG28xx.
 * 0x0000 and 0xFFFF are not allowed to be defined as SW ID.
 * SW_ID_UNDEFINED is a reserved enum value, do not delete it or modify it.
 * Please modify the SW ID of the below enum value depends on the TP vendor that you are using.
 */
typedef enum {
    MSG28XX_SW_ID_XXXX = 0x0001,
    MSG28XX_SW_ID_YYYY = 0x0002,
    MSG28XX_SW_ID_UNDEFINED = 0xFFFF
} Msg28xxSwId_e;
```

上述分別是 MStar 互容屏觸控 IC(MSG26xxM/MSG28xx/MSG58xx)或自容屏觸控 IC(MSG21xxA/MSG22xx)的 SW ID 機制因應不同 TP 供應商必須在 Touch Device Driver 和 TP Firmware 內設定不同 TP 供應商的 SW ID，這些 SW ID 設定值需依據客戶需求決定如何更改。

2.4.3 Function

底下針對 MStar_drv _fw_control.h 中較重要的 Function 宣告作進一步說明， 其餘請直接參照 MStar_drv _fw_control.h 和 MStar_drv _fw_control.c。

2.4.3.1. Gesture Wakeup

```
#ifdef CONFIG_ENABLE_GESTURE_WAKEUP
extern void DrvFwCtrlOpenGestureWakeup(u32 *pMode);
extern void DrvFwCtrlCloseGestureWakeup(void);

#ifdef CONFIG_ENABLE_GESTURE_DEBUG_MODE
```

```
extern void DrvFwCtrlOpenGestureDebugMode(u8 nGestureFlag);
extern void DrvFwCtrlCloseGestureDebugMode(void);
#endif //CONFIG_ENABLE_GESTURE_DEBUG_MODE
#endif //CONFIG_ENABLE_GESTURE_WAKEUP
```

上述的函式是 Touch Device Driver 用來開關 MStar 互容屏觸控 IC(MSG26xxM/MSG28xx/MSG58xx)或自容屏觸控 IC(MSG21xxA/MSG22xx)上的手勢喚醒功能， 除此之外還可以設定要 Enable 的手勢喚醒手勢有哪幾種。另外，還有開關 Gesture Debug Mode(TP Firmware 用來 Debug 手勢喚醒問題)的相關函式。

2.4.3.2. Firmware Debug Mode Data Log

```
extern u16 DrvFwCtrlChangeFirmwareMode(u16 nMode);
extern void DrvFwCtrlSelfGetFirmwareInfo(SelfFirmwareInfo_t *pInfo);
extern void DrvFwCtrlMutualGetFirmwareInfo(MutualFirmwareInfo_t *pInfo);extern u16
DrvFwCtrlGetFirmwareMode(void);
extern void DrvFwCtrlRestoreFirmwareModeToLogDataMode(void);
```

上述的函式是用來讓 Touch Device Driver 可以觸發 MStar 互容屏觸控固件(MSG26xxM/MSG28xx/MSG58xx)或自容屏觸控固件(MSG21xxA/MSG22xx)從 Demo Mode 切換到 Debug Mode， 並將 Debug Mode 資料輸出到手機 T Card 上的一個.csv 檔， 最後再透過 MStar 提供的 MxViewer 工具來回播.csv 檔， 用 UI 顯示的方式方便 TP Firmware 同仁分析觸控固件問題。

2.4.3.3. SW ID

```
#ifdef CONFIG_UPDATE_FIRMWARE_BY_SW_ID
extern void DrvFwCtrlCheckFirmwareUpdateBySwId(void);
#endif //CONFIG_UPDATE_FIRMWARE_BY_SW_ID
```

上述的函式是用來讓 Touch Device Driver 在手機開機時透過 SW ID 機制來檢查是否需要更新觸控固件。

2.4.3.4. Optimize Current Consumption

```
extern void DrvFwCtrlOptimizeCurrentConsumption(void);
```

上述的函式特別針對互容屏觸控 IC(MSG28xx/MSG58xx)或自容屏觸控 IC(MSG22xx)這一顆 IC， 透過讓 Touch Device Driver 在手機休眠時呼叫上述函式， 藉此讓上述 IC 進入特別的省電模式， 用以優化在手機休眠狀態下觸控 IC 的電流耗損。

2.4.3.5. Glove Mode

```
#ifdef CONFIG_ENABLE_GLOVE_MODE
extern void DrvFwCtrlOpenGloveMode(void);
extern void DrvFwCtrlCloseGloveMode(void);
extern void DrvFwCtrlGetGloveInfo(u8 *pGloveMode);
#endif //CONFIG_ENABLE_GLOVE_MODE
```

上述的函式是 Touch Device Driver 用來控制開關 TP Firmware 的 Glove Mode(手套模式)功能，目前僅互容屏觸控 IC(MSG28xx/MSG58xx)有支援 Glove Mode 功能。除此之外，MStar 的其它觸控 IC 皆不支援 Glove Mode 功能。

2.4.3.6. Leather Sheath Mode

```
#ifdef CONFIG_ENABLE_LEATHER_SHEATH_MODE
extern void DrvFwCtrlOpenLeatherSheathMode(void);
extern void DrvFwCtrlCloseLeatherSheathMode(void);
extern void DrvFwCtrlGetLeatherSheathInfo(u8 *pLeatherSheathMode);
#endif //CONFIG_ENABLE_LEATHER_SHEATH_MODE
```

上述的函式是 Touch Device Driver 用來控制開關 TP Firmware 的 Leather Sheath Mode(皮套模式)功能，目前僅互容屏觸控 IC(MSG28xx/MSG58xx)有支援 Leather Sheath Mode 功能。除此之外，MStar 的其它觸控 IC 皆不支援 Leather Sheath Mode 功能。

2.4.3.7. Others

```
extern void DrvFwCtrlVariableInitialize(void);
extern u8 DrvFwCtrlGetChipType(void);
extern void DrvFwCtrlGetCustomerFirmwareVersion(u16 *pMajor, u16 *pMinor, u8 **ppVersion);
extern void DrvFwCtrlGetPlatformFirmwareVersion(u8 **ppVersion);
extern void DrvFwCtrlHandleFingerTouch(void);
extern s32 DrvFwCtrlUpdateFirmware(u8 szFwData[][1024], EmemType_e eEmemType);
extern s32 DrvFwCtrlUpdateFirmwareBySdCard(const char *pFilePath);
```

上述的函式包含 Get Chip Type， Get Customer/Platform Firmware Version， Update Firmware by APK 及 Finger Touch Handling(報點)等功能，也都實作在 MStar_drv _fw_control.c 中.

2.5. MStar_drv _mp_test.h & MStar_drv _mp_test.c

在整個 Touch Device Driver 代碼中， 跟 MStar 的互容屏觸控 IC(MSG26xxM/MSG28xx)和自容屏觸控 IC(MSG21xxA/MSG22xx)相關的整機 MP Test 實作， 都定義在 MStar_drv _mp_test.h 和 MStar_drv _mp_test.c 中。 詳述如下：

2.5.1 Constant Value

底下針對 MStar_drv _mp_test.h 中較重要的 Constant Value 定義作進一步說明， 其餘請直接參照 MStar_drv _mp_test.h。

2.5.1.1. MP Test Retry Count

```
#define CTP_MP_TEST_RETRY_COUNT (3)
```

上述是 MP Test 的 Retry Count， 此設定值可依據客戶需求決定是否更改。

2.5.2 Function

底下針對 MStar_drv _mp_test.h 中較重要的 Function 宣告作進一步說明， 其餘請直接參照 MStar_drv _mp_test.h 和 MStar_drv _mp_test.c。

2.5.2.1. MP Test

```
extern void DrvMpTestCreateMpTestWorkQueue(void);
extern void DrvMpTestGetTestDataLog(ItoTestMode_e eItoTestMode, u8 *pDataLog, u32 *pLength);
extern void DrvMpTestGetTestFailChannel(ItoTestMode_e eItoTestMode, u8 *pFailChannel, u32
*pFailChannelCount);
extern s32 DrvMpTestGetTestResult(void);
#if defined(CONFIG_ENABLE_CHIP_TYPE_MSG26XXM) || defined(CONFIG_ENABLE_CHIP_TYPE_MSG28XX)
extern void DrvMpTestGetTestScope(TestScopeInfo_t *pInfo);
#endif //CONFIG_ENABLE_CHIP_TYPE_MSG26XXM || CONFIG_ENABLE_CHIP_TYPE_MSG28XX
#if defined(CONFIG_ENABLE_CHIP_TYPE_MSG28XX)
extern void DrvMpTestGetTestLogAll(u8 *pDataLog, u32 *pLength);
#endif //CONFIG_ENABLE_CHIP_TYPE_MSG28XX
extern void DrvMpTestScheduleMpTestWork(ItoTestMode_e eItoTestMode);
```

上述的函式是 Touch Device Driver 用來控制 MStar 互容屏觸控 IC(MSG26xxM/MSG28xx)和自容屏觸控 IC(MSG21xxA/MSG22xx)上 Mp Test 功能的常用函式， 包含顯示開路及短路測試結果、有開路或短路問題的 Channel 編號以及將開路及短路測試數據儲存到.csv 檔中。

2.6. MStar_drv_platform_porting_layer.h & MStar_drv_platform_porting_layer.c

在整個 Touch Device Driver 代碼中， 只要在各種不同的智慧型手機開發平台(EX. MTK/QCOM/SPRD)， 設定或功能實作會有差異的地方， 如 GPIO 的 RESET Pin 和 INTERRUPT Pin 的設定、TP 上 Virtual Key 的鍵值設定、 Finger Touch 的 Enable/Disable、Touch IC 的 Power On/Power Off、回報 Touch Up/Touch Down 給 Input Sub-System、Touch Device Driver 的初始化流程、設定 I2C Data Rate 的實作方式、註冊硬體中斷等， 都集中在 MStar_drv_platform_porting_layer.h 和 MStar_drv_platform_porting_layer.c 中處理。 詳述如下：

2.6.1 Constant Value

底下針對 MStar_drv_platform_porting_layer.h 中較重要的 Constant Value 定義作進一步說明， 其餘部分請直接參照 MStar_drv_platform_porting_layer.h。

2.6.1.1. SPRD Platform

```
// TODO : Please FAE colleague to confirm with customer device driver engineer about the value of RST and INT
GPIO setting
#ifndef CONFIG_ENABLE_TOUCH_PIN_CONTROL
#define MS_TS_MSG_IC_GPIO_RST     GPIO_TOUCH_RESET //53 //35
#define MS_TS_MSG_IC_GPIO_INT     GPIO_TOUCH_IRQ     //52 //37
#endif //CONFIG_ENABLE_TOUCH_PIN_CONTROL

#ifdef CONFIG_TP_HAVE_KEY
```

```
#define TOUCH_KEY_MENU (139) //229
#define TOUCH_KEY_HOME (172) //102
#define TOUCH_KEY_BACK (158)
#define TOUCH_KEY_SEARCH (217)

#define MAX_KEY_NUM (4)
#endif //CONFIG_TP_HAVE_KEY
```

如果客戶所使用的智慧型手機開發平台是 SPRD 平台， 請跟客戶確認 GPIO 的 RESET Pin 和 INTERRUPT Pin 的設定值。不過有某些 SPRD 平台的 BB Chip，GPIO 的 RESET Pin 和 INTERRUPT Pin 的設定值是透過 Pin Control 機制取得，如果是這樣的話，必須開啟 compile option CONFIG_ENABLE_TOUCH_PIN_CONTROL。另外， 如果客戶所使用的 SPRD 平台， Virtual Key 的鍵值設定跟目前 Touch Device Driver 的預設設定值不一樣， 請修改成跟客戶所使用的一致。

2.6.1.2. QCOM Platform

```
// TODO : Please FAE colleague to confirm with customer device driver engineer about the value of RST and INT GPIO setting
#ifndef CONFIG_ENABLE_TOUCH_PIN_CONTROL
#define MS_TS_MSG_IC_GPIO_RST    0
#define MS_TS_MSG_IC_GPIO_INT    1
#endif //CONFIG_ENABLE_TOUCH_PIN_CONTROL

#ifdef CONFIG_TP_HAVE_KEY
#define TOUCH_KEY_MENU (139) //229
#define TOUCH_KEY_HOME (172) //102
#define TOUCH_KEY_BACK (158)
#define TOUCH_KEY_SEARCH (217)

#define MAX_KEY_NUM (4)
#endif //CONFIG_TP_HAVE_KEY
```

如果客戶所使用的智慧型手機開發平台是 QCOM 平台， 請跟客戶確認 GPIO 的 RESET Pin 和 INTERRUPT Pin 的設定值。不過有某些 QCOM 平台的 BB Chip，GPIO 的 RESET Pin 和 INTERRUPT Pin 的設定值是透過 Pin Control 機制取得，如果是這樣的話，必須開啟 compile option CONFIG_ENABLE_TOUCH_PIN_CONTROL。另外， 如果客戶所使用的 QCOM 平台， Virtual Key 的鍵值設定跟目前 Touch Device Driver 的預設設定值不一樣， 請修改成跟客戶所使用的一致。

2.6.1.3. MTK Platform

```
#ifndef CONFIG_PLATFORM_USE_ANDROID_SDK_6_UPWARD
#define MS_TS_MSG_IC_GPIO_RST     (GPIO_CTP_RST_PIN)
#define MS_TS_MSG_IC_GPIO_INT     (GPIO_CTP_EINT_PIN)
#endif //CONFIG_PLATFORM_USE_ANDROID_SDK_6_UPWARD
```

```
#ifdef CONFIG_TP_HAVE_KEY
#define TOUCH_KEY_MENU      KEY_MENU
#define TOUCH_KEY_HOME      KEY_HOMEPAGE
#define TOUCH_KEY_BACK      KEY_BACK
#define TOUCH_KEY_SEARCH    KEY_SEARCH

#define MAX_KEY_NUM (4)
#endif //CONFIG_TP_HAVE_KEY
```

如果客戶所使用的智慧型手機開發平台是 MTK 平台，且搭配的安卓系統是 Android 5.x 以前的版本， 通常 GPIO 的 RESET Pin 和 INTERRUPT Pin 的設定值不用再另行設定。但如果 MTK 平台搭配的安卓系統是 Android 6.x 以後的版本，則必須開啟 compile option CONFIG_PLATFORM_USE_ANDROID_SDK_6_UPWARD，Touch Device Driver 會用別的方式來設定 GPIO 的 RESET Pin 和 INTERRUPT Pin 的設定值。另外， 如果客戶所使用的是 MTK 平台，Virtual Key 的鍵值設定也不用再另行設定。

2.6.2 Function

底下針對 MStar_drv_platform_porting_layer.h 中較重要的 Function 宣告作進一步說明， 其餘請直接參照 MStar_drv_platform_porting_layer.h 和 MStar_drv_platform_porting_layer.c。

2.6.2.1. Input Device Registration

```
extern s32 DrvPlatformLyrInputDeviceInitialize(struct i2c_client *pClient);
```

上述的函式針對不同的智慧型手機開發平台， Touch Device Driver 在註冊或設定 Touch Input Device 的方式會有差異， 相關的處理會用不同平台的 Compile Option 區隔。

2.6.2.2. Enable/Disable Finger Touch

```
extern void DrvPlatformLyrDisableFingerTouchReport(void);
extern void DrvPlatformLyrEnableFingerTouchReport(void);
```

上述的函式針對不同的智慧型手機開發平台， 在 Enable 或 Disable 觸控固件透過硬體中斷通知 Touch Device Driver 來讀取報點封包的方式會有差異，相關的處理會用不同平台的 Compile Option 區隔。

.

2.6.2.3. Power On/Power Off Touch IC

```
extern void DrvPlatformLyrTouchDevicePowerOff(void);
extern void DrvPlatformLyrTouchDevicePowerOn(void);
#ifdef CONFIG_ENABLE_REGULATOR_POWER_ON
extern void DrvPlatformLyrTouchDeviceRegulatorPowerOn(bool nFlag);
#endif //CONFIG_ENABLE_REGULATOR_POWER_ON
extern void DrvPlatformLyrTouchDeviceResetHw(void);
```

上述的函式針對不同的智慧型手機開發平台， Touch Device Driver 在幫 Touch IC 上電或斷電的方式會有差異， 相關的處理會用不同平台的 Compile Option 區隔。

2.6.2.4. Report Touch Up/Touch Down to Input Sub-System

```
extern void DrvPlatformLyrFingerTouchPressed(s32 nX, s32 nY, s32 nPressure, s32 nId);
extern void DrvPlatformLyrFingerTouchReleased(s32 nX, s32 nY, s32 nId);
```

上述的函式針對不同的智慧型手機開發平台， Touch Device Driver 在回報 Finger Touch 資訊給 Linux 的 Input Sub-System 的方式會有些許差異， 相關的處理會用不同平台的 Compile Option 區隔。

2.6.2.5. Interrupt Handler Registration

```
extern s32 DrvPlatformLyrTouchDeviceRegisterFingerTouchInterruptHandler(void);
```

上述的函式針對不同的智慧型手機開發平台， 在註冊觸控固件透過硬體中斷通知 Touch Device Driver 讀取報點封包的方式會有些許差異， 相關的處理會用不同平台的 Compile Option 區隔。

2.6.2.6. Request GPIO Usage

```
extern s32 DrvPlatformLyrTouchDeviceRequestGPIO(struct i2c_client *pClient);
```

上述的函式針對不同的智慧型手機開發平台，Touch Device Driver 在 Request RESET 和 INTERRUPT 這兩個 GPIO Pin 的使用權方式會有些許差異， 相關的處理會用不同平台的 Compile Option 區隔。

2.6.2.7. Set I2C Data Rate

```
extern void DrvPlatformLyrSetIicDataRate(struct i2c_client *pClient, u32 nIicDataRate);
```

上述的函式針對不同的智慧型手機開發平台， Touch Device Driver 在設定 I2C Data Rate 的方式會有差異， 目前只有 MTK 平台的設定方式已經確認， 至於 SPRD 和 QCOM 平台的設定方式則需要 FAE 同仁跟客戶進一步確認。

2.6.2.8. Report Finger Touch by Coordinate

```
#ifdef CONFIG_ENABLE_REPORT_KEY_WITH_COORDINATE
#if defined(CONFIG_TOUCH_DRIVER_RUN_ON_MTK_PLATFORM)
#define BUTTON_W (100)
#define BUTTON_H (100)

int g_TpVirtualKeyDimLocal[MAX_KEY_NUM][4] =
{{BUTTON_W/2*1,TOUCH_SCREEN_Y_MAX+BUTTON_H/2,BUTTON_W,BUTTON_H},{BUTTON_W/2*3,TOUCH
_SCREEN_Y_MAX+BUTTON_H/2,BUTTON_W,BUTTON_H},{BUTTON_W/2*5,TOUCH_SCREEN_Y_MAX+BUTTON
_H/2,BUTTON_W,BUTTON_H},{BUTTON_W/2*7,TOUCH_SCREEN_Y_MAX+BUTTON_H/2,BUTTON_W,BUTTON
_H}};
#endif // CONFIG_TOUCH_DRIVER_RUN_ON_MTK_PLATFORM
#endif //CONFIG_ENABLE_REPORT_KEY_WITH_COORDINATE
```

因為在 MTK 手機開發平台上， Touch Device Driver 是使用轉換後的座標資訊來回報 TP 上哪一個 Virtual Key 給 Linux 的 Input Sub-System， 而非直接用 Virtual Key 的鍵值來回報。因此每一個 Virtual Key 的座標定義需要依據不同專案所使用的 Touch Panel 尺寸大小作調整， 這部份需要 FAE 同仁跟客戶進一步確認. 相關的定義請至 MStar_drv_platform_porting_layer.c 內修改。

2.7. MStar_drv_platform_interface.h & MStar_drv_platform_interface.c

在整個 Touch Device Driver 代碼中， MStar 提供給不同的智慧型手機開發平台(EX. MTK/QCOM/SPRD)呼叫的接口， 如 Touch Device Driver 的註冊和初始化程序， Touch Device Driver 的 Suspend/Resume 處理等， 都集中在 MStar_drv_platform_interface.h 和 MStar_drv_platform_interface.c 中處理。 詳述如下：

2.7.1 Function

下列針對 MStar_drv_platform_interface.h 中較重要的 Function 宣告作進一步說明， 其餘請直接參照 MStar_drv_platform_intreface.h 和 MStar_drv_platform_interface.c。

2.7.1.1. Device Driver Initialization

```
extern s32 /*__devinit*/ MsDrvInterfaceTouchDeviceProbe(struct i2c_client *pClient, const struct i2c_device_id *pDeviceId);
```

上述的函式提供不同的智慧型手機開發平台呼叫的接口， 主要用來執行 MStar 的 Touch Device Driver 的初始化程序， 其實就是以前 Touch Device Driver 內 Probe()要作的事。

2.7.1.2. Device Driver Suspend/Resume

```
#ifdef CONFIG_ENABLE_NOTIFIER_FB
extern int MsDrvInterfaceTouchDeviceFbNotifierCallback(struct notifier_block *pSelf, unsigned long nEvent, void *pData);
#else
#ifdef CONFIG_PLATFORM_USE_ANDROID_SDK_6_UPWARD
extern void MsDrvInterfaceTouchDeviceResume(struct device *pDevice);
extern void MsDrvInterfaceTouchDeviceSuspend(struct device *pDevice);
#else
extern void MsDrvInterfaceTouchDeviceResume(struct early_suspend *pSuspend);
extern void MsDrvInterfaceTouchDeviceSuspend(struct early_suspend *pSuspend);
#endif //CONFIG_PLATFORM_USE_ANDROID_SDK_6_UPWARD
#endif //CONFIG_ENABLE_NOTIFIER_FB
```

上述的函式提供給不同智慧型手機開發平台呼叫的接口， 主要用來執行當 Touch Device Driver 收到系統發的 Suspend 或 Resume 通知時相對應的處理方式。如果是 QCOM 平台或 SPRD 平台的話，通常需開啟 compile option CONFIG_ENABLE_NOTIFIER_FB，用 Notify Callback 的方式來接收系統發的 Suspend 或 Resume 通知。但如果是 MTK 平台且搭配的安卓系統是 Android 6.x 以後的版本，則必須開啟 compile option CONFIG_PLATFORM_USE_ANDROID_SDK_6_UPWARD。如果是 MTK 平台且搭配的安卓系統是 Android 5.x 以前的版本，則不要開啟 compile option CONFIG_PLATFORM_USE_ANDROID_SDK_6_UPWARD。

2.8. MTK Hotknot

Hotknot 是由 MediaTek 開發的一種短距離資料傳輸的協定， Touch Device Driver 中要開啟 Hotknot 功能(目前只有 MSG28xx 支援 Hotknot 功能)， 請在 mstar_drv_common.h 內 Enable 下述 Compile Option：

```
#define CONFIG_ENABLE_HOTKNOT
```

Hotknot 傳輸中， 負責處理由 Native Layer 透過 ioctl 發送 Hotknot 指令的相關函式， 都定義在 mstar_drv_hotknot.h 和 mstar_drv_hotknot.c 中。

2.8.1 Hotknot 指令

Hotknot 指令大致上可以區分成三種， Hotknot 驗證指令、 Hotknot 控制或傳送資料指令、 Hotknot 接收資料指令。 在 Driver 中不同類型的指令會由不同的函式處理。

void _DrvHandleHotKnotAuth(DrvCmd_t *pCmd)	處理 Hotknot 驗證指令
int _DrvHandleHotKnotCmd(DrvCmd_t *pCmd, unsigned long nArg)	處理 Hotknot 控制或傳送資料指令
void _DrvHandleHotKnotRcv(DrvCmd_t *pCmd)	處理 Hotknot 接收資料指令

2.8.2 Hotknot 驗證指令

驗證指令主要是對固件發送驗證 Hotknot 的指令， 驗證整機的 Hotknot 是否符合規範。 固件接收到指令之後， 會立刻做 Hotknot 驗證運算， 再回傳給 Driver， Driver 再將資料回傳到 Native Layer。

2.8.3 Hotknot 控制或傳送資料用的指令

Driver 會對固件下各種的 Hotknot 控制指令， 固件針對各種控制指令作相對應的處理或是切換 State Machine，另外， 當整機為 Master(發送端)， Driver 會負責傳送從 Native Layer 來的 Hotknot 資料到固件， 固件再透過 Hotknot 協定發送資料到 Slave(接收端)。 Driver 對固件發送指令之後， 就會等待固件透過中斷回傳處理結果， 負責處理回傳結果的函式為：

```
void ReportHotKnotCmd(u8 *pPacket, u16 nLength)
```

再將收到的結果回傳到 Native Layer， 若固件發生錯誤而沒有回傳結果， Driver 超過 Timeout 時間就會進入處理 Timeout 相關的函式：

```
void _HotKnotTimeOutHandler()
```

然後回報錯誤給 Native Layer。

2.8.4 Hotknot 接收資料指令

當整機為 Slave 時， 在傳輸模式下(Transfer Mode)， 隨時有可能接收到來自 Master 的 Hotknot 資料. 固件收到資料之後，會透過中斷傳送到 Driver， Driver 裡面建立的一個 Queue 會儲存這些資料， 這部分可以參考 mstar_drv_hotknot_queue.h， mstar_drv_hotknot_queue.c， 裡面有幾個主要函式， 說明如下：

void CreateQueue()	建立 Queue
PushQueue(u8 * pBuf, u16 nLength)	將資料放入 Queue
PopQueue(u8 * pBuf, u16 nLength)	從 Queue 取出資料
DeleteQueue(void);	刪除 Queue

Native Layer 每隔一段固定的時間就會下指令到 Driver 檢查 Queue 是否有 Hotknot 資料， 有資料就取回去， 如果在指定的 Timeout 時間內都沒有收到任何資料， 就會回報錯誤。

2.8.5 Hotknot Shared library

Hotknot 除了 Driver 部分， 還必須搭配 Native Layer 的 Shared Library(.so 檔)， 主要有 libhotknot.so、libhotknot_dev.so 和 libhotknot_vendor.so 三個 Shared Library， 其中 libhotknot.so、 libhotknot_dev.so 由 MediaTek 提供， 主要是處理 Framework 端的系統呼叫， 而 libhotknot_vendor.so 會呼叫 Driver、 Driver 和固件溝通及實現 Hotknot Interface。

2.8.6 如何驗證 Hotknot libhotknot_vendor.so 和 Driver 功能

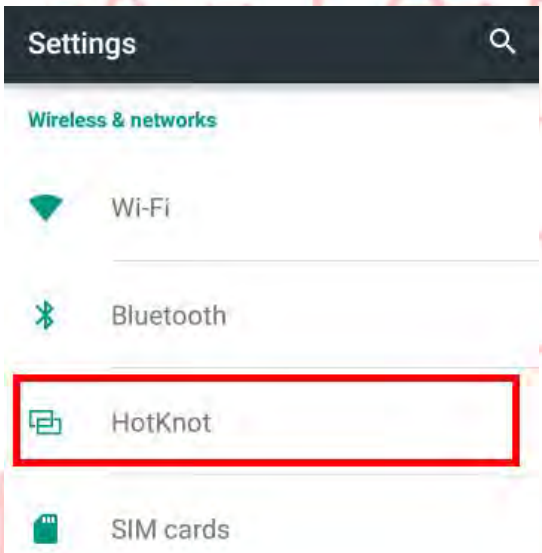
驗證方法是安裝 libhotknot_vendor.so 和一個測試用的程式 hotknot_test， libhotknot.so 和 libhotknot_dev.so 不需要安裝， 這部分驗證成功代表 Hotknot 呼叫流程 "libhotknot_vendor.so => Driver => 固件" 是正常的。
安裝方法如下:

1. 以管理者(Administrator)權限打開 Windows console 程式
2. 安裝 libhotknot_vendor.so 和 hotknot_test
   (1) adb root (獲取 root 權限)
   (2) adb remount
   (3) 若整機為 64bit 系統， 則 so 檔有分 64bit 和 32bit
   adb push [libhotknot_vender.so (32bit) 路徑] /system/lib
   adb push [libhotknot_vender.so (64bit) 路徑] /system/lib64
   (4) adb push [hotknot_test 路徑] /system/bin/
   (5) adb shell chmod 755 /system/bin/hotknot_test (修改 hotknot_test 屬性)
   (6) adb reboot (重新開機)
3. 執行測試程式
   (1) adb shell (進入 android shell)
   (2) ./system/bin/hotknot_test
4. HotKnot 驗證測試
   select "2. Authentication Test"
5. HotKnot 傳輸測試 master(sender)
   master(sender): select "3. Send Test"
   執行程式之後就將測試的 master、slave 相互貼屏
6. HotKnot 傳輸測試 slave(receiver)
   slave(receiver): select "4. Receive Test"
   執行程式之後就將測試的 master、slave 相互貼屏
7. 測試程式中，每一個步驟都會顯示訊息。如果出現"success"就代表這個步驟成功；如果出現"failed"就代表這個步驟測試失敗。 如果發現"failed"訊息，可以將整個測試顯示的訊息傳回給相關的同仁幫忙分析。

2.8.7 Hotknot 完整驗證

2.10.6 所描述的是驗證 Hotknot libhotknot_vendor.so 和 Driver 功能， 若要進行完整驗證則必須將 libhotknot.so 和 libhotknot_dev.so 也安裝進整機， 安裝方法如下:

1. 以管理者(Administrator)權限打開 Windows consle 程式
2. 安裝 libhotknot_vendor.so 和 hotknot_test
   (1) adb root (獲取 root 權限)
   (2) adb remount
   (3) 若整機為 64bit 系統， 則 so 檔有分 64bit 和 32bit
   adb push [libhotknot.so (32bit) 路徑] /system/lib
   adb push [libhotknot.so (64bit) 路徑] /system/lib64
   (4) adb push [libhotknot_dev.so (32bit) 路徑] /system/lib
   adb push [libhotknot_dev.so (64bit) 路徑] /system/lib64
   (5) adb reboot (重新開機)
3. 若 hotknot 功能正常工作則會在整機的設定(Settings)裡面看到 hotknot 選項， 如下:



2.8.8 Hotknot 驗證階段

Hotknot 功能的驗證會因為固件的開發階段而有所不同， 大致上可以區分為三個階段：

1. Touch IC 固件還未有 Hotknot 功能
   這個階段 Porting Driver 只能查看 Shared Library 呼叫 Driver 是否成功， 但因為固件沒有 Hotknot 功能，所以 Driver 等不到固件的回應， 只能進入 Timeout 函式。
2. Touch IC 固件有 Pseudo Hotknot 功能
   這個階段 Porting Driver， 當 Driver 呼叫固件程式，從固件回傳的資料為固件預設的資料並不是真實 Hotknot 傳輸產生的資料， 目的只是要驗證從 Native Layer 到固件呼叫流程是否正常。
3. Touch IC 固件有完整 Hotknot 功能
   這個階段 Porting Driver 從固件回傳的資料為真實 Hotknot 傳輸產生的資料， 可驗證完整的 Hotknot 功能。

2.8.9 Pseudo Hoknot 固件

使用 Pseudo Hotknot 固件須注意下面幾個事項:

1. Pseudo Hotknot 固件主要是為了 Hotknot 功能的測試， 可以在各種 TP 上面作測試， 但是觸控報點並不一定正常， 所以在整機上更新 Pseudo Hotknot 固件之前， 先確定有原來可以正常觸控報點的固件檔案， 以便做完 Hotknot 功能測試之後， 可以更新回原來的固件。
2. 更新 Pseudo Hotknot 固件之後， 若整機上的觸控不正常， 可以用 usb 接整機連回 PC， 利用第三方軟件工具(ex. Android Screen Monitor)控制整機上面的畫面來進行操作。

2.9. MStar_drv_sprd.c

在整個 Touch Device Driver 代碼中， MStar 提供給 SPRD 智慧型手機開發平台使用的 Sample Code， 可以參照 MStar_drv_sprd.c。 這份 Sample Code 已經跟 MStar 的 Touch Device Driver 所提供的接口完成串接， 可直接使用即可。

另外， 在 SPRD 平台要成功編譯 MStar 的 Touch Device Driver 代碼以及要成功註冊使用 MStar 的 Touch IC， 必須注意下列兩點：

2.9.1 How to Register MStar Touch IC

在 SPRD 平台要成功註冊使用 MStar 的 Touch IC， 必須注意在 MStar_drv_sprd.c 內的下述設定值(EX. msg2xxx).

```
#define MSG_TP_IC_NAME "msg2xxx"
```

必須和客戶的 SPRD 平台 Codebase 中， 用來放置下述代碼的另一個檔案， 在跟 I2C BUS 註冊使用時的設定值一樣， 舉例如下：

```
static struct i2c_board_info __initdata i2c1_boardinfo[] = {
    { I2C_BOARD_INFO("msg2xxx",  (0x4C>>1)), },  //SLAVE_I2C_ID_DWI2C
};
```

2.9.2 How to Compile MStar Touch Device Driver Code

下列 Makefile 的寫法是 MStar 的 Touch Device Driver 代碼如何放進 SPRD 平台來 Build Code 的作法， 舉例如下：

```
obj-$(CONFIG_TOUCHSCREEN_MSG2238) += msg2238_ts.o

msg2238_ts-objs := mstar_drv_sprd.o
msg2238_ts-objs += mstar_drv_common.o
msg2238_ts-objs += mstar_drv_utility_adaption.o
msg2238_ts-objs += mstar_drv_fw_control.o
msg2238_ts-objs += mstar_drv_mp_test.o
msg2238_ts-objs += mstar_drv_ic_fw_porting_layer.o
msg2238_ts-objs += mstar_drv_main.o
msg2238_ts-objs += mstar_drv_platform_interface.o
msg2238_ts-objs += mstar_drv_platform_porting_layer.o
msg2238_ts-objs += mstar_drv_jni_interface.o
```

2.10. Mstar_drv_qcom.c

在整個 Touch Device Driver 代碼中， MStar 提供給 QCOM 智慧型手機開發平台使用的 Sample Code， 可以參照 mstar_drv_qcom.c。這份 Sample Code 已經跟 MStar 的 Touch Device Driver 所提供的接口完成串接， 可直接使用即可。

另外， 在 QCOM 平台要成功編譯 MStar 的 Touch Device Driver 代碼以及要成功註冊使用 MStar 的 Touch IC， 必須注意底下兩點：

2.10.1 How to Register MStar Touch IC

在 QCOM 平台要成功註冊使用 MStar 的 Touch IC， 必須注意在 mstar_drv_qcom.c 內的下述設定值(EX. msg2xxx) 。

```
#define MSG_TP_IC_NAME "msg2xxx"
```

必須和客戶的 QCOM 平台 Codebase 中， 用來放置下述代碼的另一個檔案， 在跟 I2C BUS 註冊使用時的設定值一樣， 舉例如下：

```
static struct i2c_board_info __initdata i2c1_boardinfo[] = {
    { I2C_BOARD_INFO("msg2xxx",  (0x4C>>1)), },  //SLAVE_I2C_ID_DWI2C
};
```

2.10.2 How to Compile MStar Touch Device Driver Code

至於如何將 MStar 的 Touch Device Driver 代碼放進 QCOM 平台來 Build Code 的作法，據了解和 SPRD 平台 Makefile 的寫法類似， 可先參照 SPRD 平台的方式來處理。

2.11. Mstar_drv_mtk.c

在整個 Touch Device Driver 代碼中， MStar 提供給 MTK 智慧型手機開發平台使用的 Sample Code， 可以參照 mstar_drv_mtk.c。這份 Sample Code 已經跟 MStar 的 Touch Device Driver 所提供的接口完成串接， 可直接使用即可。

另外， 在 MTK 平台要成功編譯 MStar 的 Touch Device Driver 代碼以及要成功註冊使用 MStar 的 Touch IC， 必須注意下列兩點：

2.11.1 How to Register MStar Touch IC

在 MTK 平台要成功註冊使用 MStar 的 Touch IC， 必須注意在 MStar_drv_mtk.c 內的下述設定值(EX. msg2xxx)。

```
#define MSG_TP_IC_NAME "msg2xxx"
```

必須和同樣在 MStar_drv_mtk.c 內， 跟 I2C BUS 註冊使用時的設定值一樣， 舉例如下：

```
static struct i2c_board_info __initdata i2c_tpd = {I2C_BOARD_INFO(MSG_TP_IC_NAME, (0x4C>>1))};
//SLAVE_I2C_ID_DW I2C
```

在 MTK 平台上， 上述的 I2C BUS 註冊動作通常是也擺在 Touch Device Driver 的代碼內， 不會放在 Codebase 的其它檔案中。

2.11.2 How to Compile MStar Touch Device Driver Code

在舊的 MTK 平台上(EX. MT6582/MT6589/..)， 只需在放置 Device Driver 的 touch panel 資料夾下， 另外新增一資料夾(EX. msg2238)， 然後將 MStar 的 Touch Device Driver 代碼放進上述新增的資料夾。 如果有需要， 可能還得特別在 Makefile 中指名要編譯的資料夾(EX. msg2238)， 即可在 MTK 平台成功編譯 MStar 的 Touch Device Driver 代碼。

但是在新的 MTK 平台上(EX. MT6735/MT6753/..)， 如何將 MStar 的 Touch Device Driver 代碼放進 MTK 平台來 Build Code 的作法， 據了解已經改成和 SPRD 平台 Makefile 的寫法類似， 可先參照 SPRD 平台的方式來處理。

3. TROUBLE SHOOTING

3.1. MTK 平台 I2C 讀取或寫入的長度限制

在 MTK 手機開發平台上， 某些型號的 BB(Base Band) Chip(EX. MT6572/MT6582/MT6589)有 I2C 的限制: read/write 一次最多 8 Byte，因此 Touch Device Driver 和 TP Firmware 或 Touch IC 間的資料傳輸必須另外 Enable 一個 DMA Mode 的機制才能跨越此 I2C 的 read/write 限制。 要 Enable 針對 I2C 的 DMA Mode 機制， 請在 mstar_drv_common.h 內 Enable 下述 Compile Option 即可。

#define CONFIG_ENABLE_DMA_IIC

另外， 在 MTK 平台的 Codebase 內(EX. MT6582)， 也可以到下述路徑的 mt_i2c.h 中， 找到跟 I2C 存取相關的參數設定， 藉此確認客戶專案所使用 BB Chip 的 I2C 相關限制為何。

../mediatek/platform/mt6582/kernel/driver/i2c/mt_i2c.h

3.2. 整機上報點效能不好

在手機開發階段， 如果要偵錯 Touch Device Driver 的問題， 通常會把 MStar_drv_common.h 內的 Flag CONFIG_TOUCH_DRIVER_DEBUG_LOG_LEVEL 給 Enable(設值為 1)， 這樣子才有辦法看到 Linux 吐的 Kernel Log 進行偵錯。 但如果開啟 Linux 吐 Kernel Log 的機制， 在整機上進行觸控操作時可能會覺得反應變慢。 因此在手機進入量產出貨階段， 客戶編譯的 Target Load 一定會把 Linux 吐 Kernel Log 的機制給關掉。

如果在整機上進行觸控操作時有覺得反應較慢的現象， 可以先到 Touch Device Driver 代碼中檢查 MStar_drv_common.h 內的 Flag CONFIG_TOUCH_DRIVER_DEBUG_LOG_LEVEL 是否有被 Enable(設值為 1)。 如果有， 可以改成 Disable(設值為 0)後， 再重新編譯一版 Target Load 試試看。 如果 Disable 後， 觸控操作的反應沒有改善， 就得從 TP Firmware 那邊來查問題了。

3.3. 整機上無法觸控操作

在客戶端合入 MStar 的 Touch Device Driver 後， 如果在整機開機後去進行觸控操作都沒反應， 可以先從下述的幾個方向來作檢查。

3.3.1 觸控 IC 是否有上電

在 SPRD 平台的某些特定 BB Chip(EX. SC7715)或 QCOM 平台的某些特定 BB Chip(EX. MSM8610)或 MTK 平台的某些特定 BB Chip(EX. MT6582)， 必須多執行下述的函式才能讓觸控 IC 可以正常上電。

```
#ifdef CONFIG_ENABLE_REGULATOR_POWER_ON
void DrvPlatformLyrTouchDeviceRegulatorPowerOn(bool nFlag)
{
```

```
#if defined(CONFIG_TOUCH_DRIVER_RUN_ON_SPRD_PLATFORM) ||
defined(CONFIG_TOUCH_DRIVER_RUN_ON_QCOM_PLATFORM)
    s32 nRetVal = 0;

    DBG(&g_I2cClient->dev, "*** %s() ***\n", __func__);

    if (nFlag == true)
    {
        nRetVal = regulator_enable(g_ReguVdd);
        if (nRetVal)
        {
            DBG(&g_I2cClient->dev, "regulator_enable(g_ReguVdd) failed. nRetVal=%d\n", nRetVal);
        }
        mdelay(20);

        nRetVal = regulator_enable(g_ReguVcc_i2c);
        if (nRetVal)
        {
            DBG(&g_I2cClient->dev, "regulator_enable(g_ReguVcc_i2c) failed. nRetVal=%d\n", nRetVal);
        }
        mdelay(20);
    }
    else
    {
        nRetVal = regulator_disable(g_ReguVdd);
        if (nRetVal)
        {
            DBG(&g_I2cClient->dev, "regulator_disable(g_ReguVdd) failed. nRetVal=%d\n", nRetVal);
        }
        mdelay(20);

        nRetVal = regulator_disable(g_ReguVcc_i2c);
        if (nRetVal)
        {
            DBG(&g_I2cClient->dev, "regulator_disable(g_ReguVcc_i2c) failed. nRetVal=%d\n", nRetVal);
        }
        mdelay(20);
    }
#elif defined(CONFIG_TOUCH_DRIVER_RUN_ON_MTK_PLATFORM)

#ifdef CONFIG_PLATFORM_USE_ANDROID_SDK_6_UPWARD
    s32 nRetVal = 0;

    DBG(&g_I2cClient->dev, "*** %s() ***\n", __func__);
```

```
    if (nFlag == true)
    {
        nRetVal = regulator_enable(g_ReguVdd);
        if (nRetVal)
        {
            DBG(&g_I2cClient->dev, "regulator_enable(g_ReguVdd) failed. nRetVal=%d\n", nRetVal);
        }
        mdelay(20);
    }
    else
    {
        nRetVal = regulator_disable(g_ReguVdd);
        if (nRetVal)
        {
            DBG(&g_I2cClient->dev, "regulator_disable(g_ReguVdd) failed. nRetVal=%d\n", nRetVal);
        }
        mdelay(20);
    }
#else
    DBG(&g_I2cClient->dev, "*** %s() ***\n", __func__);

//      hwPowerOn(MT6323_POWER_LDO_VGP1, VOL_2800, "TP"); // For specific MTK BB chip(ex. MT6582), need to enable this function call for correctly power on Touch IC.
    hwPowerOn(PMIC_APP_CAP_TOUCH_VDD, VOL_2800, "TP"); // For specific MTK BB chip(ex. MT6735), need to enable this function call for correctly power on Touch IC.
#endif //CONFIG_PLATFORM_USE_ANDROID_SDK_6_UPWARD
#endif
}
#endif //CONFIG_ENABLE_REGULATOR_POWER_ON
```

因此在客戶端合入 MStar 的 Touch Device Driver 後， 如果有觸控 IC 無法正常上電的狀況， 需要 FAE 同仁跟客戶的驅動工程師確認該專案所使用的 BB Chip 針對觸控 IC 的上電是否有特別不一樣的地方。

3.3.2 I2C Bus ID 或 I2C Slave Address 設定是否正確

如在整機開機後進行觸控操作都沒反應， 也有可能是 I2C 根本沒有通。
以 MTK 平台為例， 請檢查 mstar_drv_mtk.c 內的 I2C Bus ID 設定是否正確或者是走 SMBUS 的 I2C Slave Address 設定是否正確， 需要 FAE 同仁跟客戶的驅動工程師作確認。

```
#define I2C_BUS_ID     (1)          // i2c bus id : 0 or 1

static struct i2c_board_info __initdata i2c_tpd = {I2C_BOARD_INFO(MSG_TP_IC_NAME, (0x4C>>1))};
//SLAVE_I2C_ID_DW I2C
```

3.3.3 GPIO 的 RESET Pin 或 INTERRUPT Pin 設定是否正確

如果 RESET Pin 的設定值不正確， Touch IC 無法正常上電， TP Firmware 就無法被重置並初始化， 也就無法傳送觸控的報點資料給 Touch Device Driver。
如果 INTERRUPT Pin 的設定值不正確，TP Firmware 就無法透過硬體中斷觸發 Interrupt 通知 Touch Device Driver 收取觸控的報點資料。

因此請檢查 mstar_drv_platform_porting_layer.h 中的 MS_TS_MSG_IC_GPIO_RST 和 MS_TS_MSG_IC_GPIO_INT 這兩個設定值是否正確， 需要 FAE 同仁跟客戶的驅動工程師作確認。

3.4. 無法透過 I2C 和觸控 IC 正常溝通

在客戶端合入 MStar 的 Touch Device Driver 後， 如果 Touch Device Driver 無法透過 I2C 和 Touch IC 正常溝通，可以先從下述的幾個方向來作檢查。

3.4.1 觸控 IC 是否有上電

在 SPRD 平台的某些特定 BB Chip(EX. SC7715)或 QCOM 平台的某些特定 BB Chip(EX. MSM8610)或 MTK 平台的某些特定 BB Chip(EX. MT6582)， 必須多執行下述的函式才能讓觸控 IC 可以正常上電。

```
#ifdef CONFIG_ENABLE_REGULATOR_POWER_ONvoid DrvPlatformLyrTouchDeviceRegulatorPowerOn(bool
nFlag)
{
#if defined(CONFIG_TOUCH_DRIVER_RUN_ON_SPRD_PLATFORM) ||
defined(CONFIG_TOUCH_DRIVER_RUN_ON_QCOM_PLATFORM)
    s32 nRetVal = 0;

    DBG(&g_I2cClient->dev, "*** %s() ***\n", __func__);

    if (nFlag == true)
    {
        nRetVal = regulator_enable(g_ReguVdd);
        if (nRetVal)
        {
            DBG(&g_I2cClient->dev, "regulator_enable(g_ReguVdd) failed. nRetVal=%d\n", nRetVal);
        }
        mdelay(20);

        nRetVal = regulator_enable(g_ReguVcc_i2c);
        if (nRetVal)
        {
            DBG(&g_I2cClient->dev, "regulator_enable(g_ReguVcc_i2c) failed. nRetVal=%d\n", nRetVal);
        }
        mdelay(20);
```

```
    }
    else
    {
        nRetVal = regulator_disable(g_ReguVdd);
        if (nRetVal)
        {
            DBG(&g_I2cClient->dev, "regulator_disable(g_ReguVdd) failed. nRetVal=%d\n", nRetVal);
        }
        mdelay(20);

        nRetVal = regulator_disable(g_ReguVcc_i2c);
        if (nRetVal)
        {
            DBG(&g_I2cClient->dev, "regulator_disable(g_ReguVcc_i2c) failed. nRetVal=%d\n", nRetVal);
        }
        mdelay(20);
    }
#elif defined(CONFIG_TOUCH_DRIVER_RUN_ON_MTK_PLATFORM)

#ifdef CONFIG_PLATFORM_USE_ANDROID_SDK_6_UPWARD
    s32 nRetVal = 0;

    DBG(&g_I2cClient->dev, "*** %s() ***\n", __func__);

    if (nFlag == true)
    {
        nRetVal = regulator_enable(g_ReguVdd);
        if (nRetVal)
        {
            DBG(&g_I2cClient->dev, "regulator_enable(g_ReguVdd) failed. nRetVal=%d\n", nRetVal);
        }
        mdelay(20);
    }
    else
    {
        nRetVal = regulator_disable(g_ReguVdd);
        if (nRetVal)
        {
            DBG(&g_I2cClient->dev, "regulator_disable(g_ReguVdd) failed. nRetVal=%d\n", nRetVal);
        }
        mdelay(20);
    }
#else
    DBG(&g_I2cClient->dev, "*** %s() ***\n", __func__);
```

```
//      hwPowerOn(MT6323_POWER_LDO_VGP1, VOL_2800, "TP"); // For specific MTK BB chip(ex. MT6582), need to enable this function call for correctly power on Touch IC.
    hwPowerOn(PMIC_APP_CAP_TOUCH_VDD, VOL_2800, "TP"); // For specific MTK BB chip(ex. MT6735), need to enable this function call for correctly power on Touch IC.
#endif //CONFIG_PLATFORM_USE_ANDROID_SDK_6_UPWARD
#endif
}
#endif //CONFIG_ENABLE_REGULATOR_POWER_ON
```

因此在客戶端合入 MStar 的 Touch Device Driver 後， 如果有觸控 IC 無法正常上電的狀況， 需要 FAE 同仁跟客戶的驅動工程師確認該專案使用的 BB Chip 針對觸控 IC 的上電是否有特別不一樣的地方。

3.4.2 DBBUS 的 I2C Slave Address 設定是否正確

MStar 的 Touch Device Driver 透過走 DBBUS 使用 I2C 跟觸控 IC 溝通的 I2C Slave Address 因不同型號的觸控 IC 有所差異， 但此 Slave Address 不需工程師另外手動設定， Touch Device Driver 會在手機開機時觸控驅動初始化的過程中， 自動設定 DBBUS 的 Slave Address 應設成 0x59 或 0x62。

```
SLAVE_I2C_ID_DBBUS :
(0xB2>>1) //0x59 //for MSG22xx
(0xC4>>1) //0x62 //for MSG21xxA/MSG26xxM/MSG28xx/MSG58xx
```

3.5. MTK 平台 build kernel 需注意的問題

在某些 MTK 平台(EX. MT6582) build kernel 時， build 流程會先將 kernel 的程式碼複製到 kernel 資料夾相對應的路徑下， 再對該資料夾下的程式碼進行 build code 的動作。 以觸控 IC 裝置驅動程式為例， kernel 資料夾相對應的路徑為 "kernel\mediatek\custom\out\kernel\touchpanel" 。 而在複製的過程中， build 流程只會對檔案日期較新的原始碼檔案進行複製， 日期較舊的將不會被複製。 這樣的機制可能會產生一個問題， 當開發者因故要將程式碼回復到較舊的版本， 可能會因為檔案日期較舊而沒有複製到 kernel 資料夾相對應的路徑， 產生不是開發者預期的結果。 解決的方法為在每次 build kernel 的時候， 先清空該路徑下的檔案， 可確保 build 流程中複製原始碼檔案的正確性。