聯陽半導體
ITE Tech. Inc.

iTE SDK

key/touch key/touch panel 模組開發指南

V0.91

ITE TECH. INC.

修訂記錄

修訂日期	修訂說明	頁次
2014/10/01	初建版本 V0.9	
2017/06/15	版本 V0.91 (新增同時支援多組 TP MODULE 功能)	

目錄

1. 前言 1
1.1 編寫目的 1
1.2 適用範圍 1
1.3 適用人員 1
2. 模組介紹 2
2.1 KEYPAD KCONFIG 設定 2
2.2 ITP KEYPAD DRIVER 3
2.2.1 GPIO KEYPAD(ITP_KEYPAD_GPIO.C) 3
2.2.2 ADVANCE GPIO KEYPAD(ITP_KEYPAD_CASTOR3.C) 3
2.2.3 I2C KEYPAD(TOUCH KEY) 3
2.2.4 客製化 KEYPAD DRIVER 4
2.3 TOUCH PANEL DRIVER IN TSLIB 4
2.3.1 TOUCH PANEL KCONFIG 設定 4
2.3.2 I2C INTERFACE 5
2.3.3 SPI INTERFACE 6
2.3.4 支援多組觸控模組 6
2.3.5 客製化 TOUCH PANEL DRIVER 8
3. 範例 10
3.1 範例 1 (KEYPAD) 10
3.2 範例 2(TOUCH PANEL) 11

1. 前言

1.1 編寫目的

介紹 key/touch key/touch panel 模組之功能, 說明 key/touch key/touch panel 調適工具之操作及使用.

1.2 適用範圍

應用 GPIO/SPI/I2C 等介面，對 key/touch key/touch panel 進行擷取按鍵與座標等資訊的取得。目前 key 的部分，已實作 GPIO pin 1 對 1 以及應用特殊電路達到少量 GPIO pin 對多組 key 的輸入資訊的取得。另外 touch key 與 touch panel 也已實作對 IT7230/IT7235BR/IT7236/IT7260/TSC2046/ZT2083 等控制晶片的支援。驅動程式開發者亦可自行編寫相關的驅動程式，以便支援其他相同介面的 touch key 與 touch panel。

1.3 適用人員

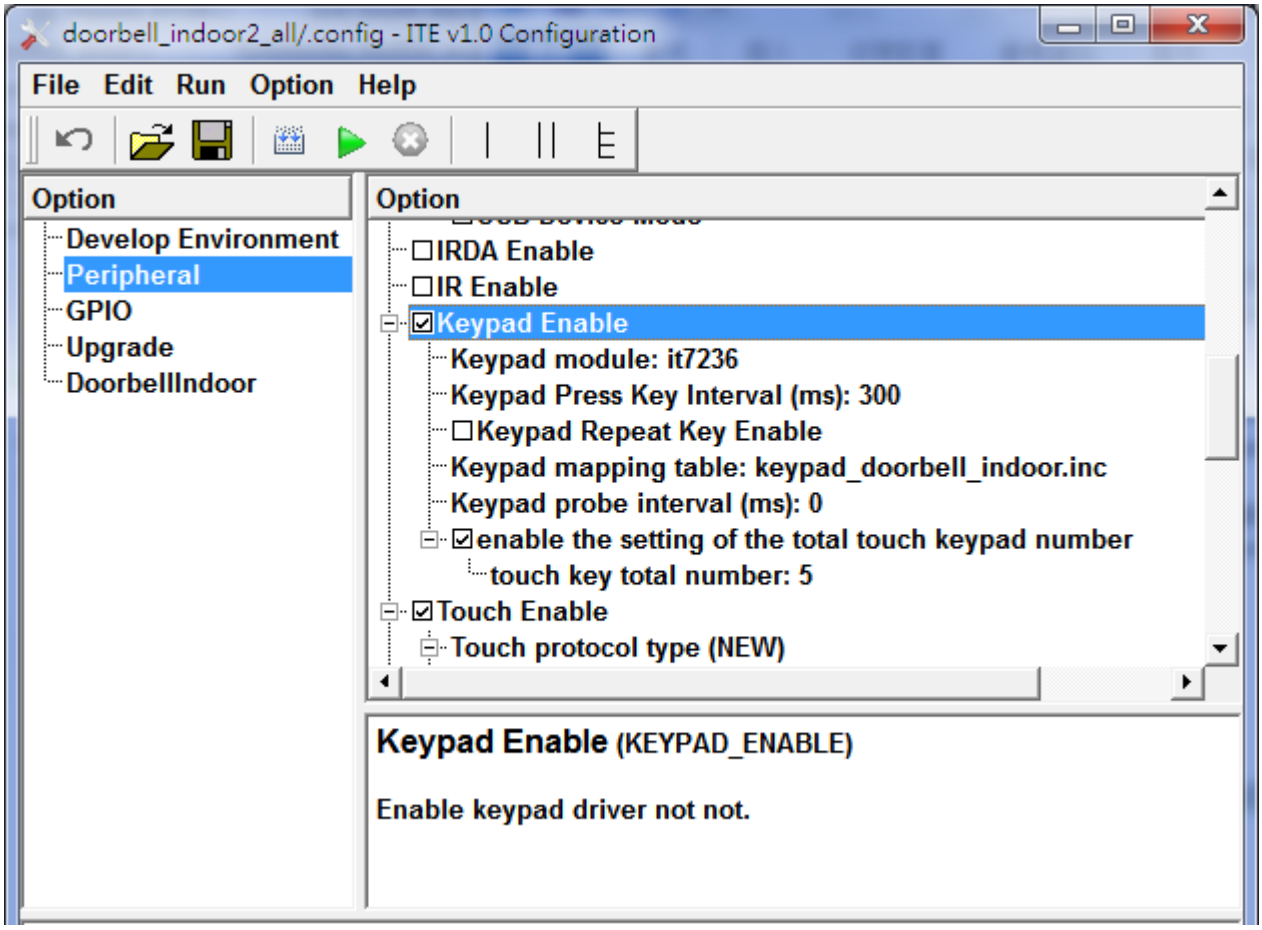
軟體應用程式，驅動程式開發者

2. 模組介紹

相關 Key/Touch Key 的 code 請參考 “sdk\driver\itp\itp_keypad.c”以及 “sdk\driver\itp\keypad\itp_keypad_xxx.c”，關於 Touch Panel 的部分，請參考 “sdk\share\tslib\plugins\xxxxxx-raw.c”。

無論是 keypad 還是 touch panel，最後都會匯入 SDL 的 event，由上層 AP 透過呼叫 SDL 的 API 來 polling key 或是 touch panel 的 event。關於 SDL 的 polling event 與 API 呼叫請參考 SDL 說明文件，或是參考 “project\doorbell_indoor\scene.c”。

2.1 Keypad KCONFIG 設定



Keypad Enable：啟動 KEYPAD 模組
Keypad module：設定所要使用的 KEYPAD 模組名稱。

Keypad Press Key interval(ms)：設定發送每個 KEY EVENT 的間隔時間。
Keypad Repeat Key Enable：啟動 repeat key 功能。
Keypad mapping table：設定 KEYPAD 的 MAPPING TABLE 檔(放在 “sdk\target\keypad”目錄中)。
Enable the setting of the total touch keypad number：啟動 KEY 總數的設定(I2C 介面)。
Touch key total number：設定 KEY 的總數(應用在 I2C 介面的 keypad)。

2.2 ITP Keypad Driver

Keypad driver 相關的 code 請參考 “sdk\driver\itp\itp_keypad.c” 以及“sdk\driver\itp\keypad\itp_keypad_xxx.c”

itp_keypad_xxx.c 是實作以下三個 API，

itpKeypadInit();
itpKeypadGetMaxLevel();
itpKeypadProbe();

以供上層 itp_keypad.c 呼叫，而上層應用程式透過 keypad 提供的 ioctl & read 讀取 keypad event。

使用 ioctl() 執行 keypad 初始化，回報 key 總數以及 probe keypad event。

ITP_IOCTL_INIT：執行 keypad 初始化。
ITP_IOCTL_GET_MAX_LEVEL：回報 key 的總數給上層。
ITP_IOCTL_PROBE：偵測 key event，若偵測到有 key 輸入，則將 event 塞入 queue 中，等待 read() 取用。

使用 read() 讀取 queue 裡的 key event

read(ITP_DEVICE_KEYPAD, &ev, sizeof (ITPKeypadEvent))

目前已經有實作的 keypad driver 有 itp_keypad_gpio.c，itp_keypad_castor3.c，itp_keypad_it7230.c，itp_keypad_it7235br.c，itp_keypad_it7236.c 等

2.2.1GPIO keypad(itp_keypad_gpio.c)

有關 1 對 1GPIO Keypad driver 相關的 code 請參考 “sdk\driver\itp\keypad\itp_keypad_gpio.c”
使用一個 GPIO PIN 偵測一個 KEY EVENT，所以 KEY 的數目會等於 GPIO 佔用的數目，適合用在 KEY 數目不多，不需複雜電路設計的情況下使用。

2.2.2Advance GPIO keypad(itp_keypad_castor3.c)

應用特殊設計過的線路，可以使用較少的 GPIO pin 偵測較多的 key(參考 keypad 硬體設計指南)。
目前最多可支援 8 組 GPIO 控制 64 組 key(n 組 GPIO 最多可偵測 n^2 個 key)

2.2.3I2C keypad(touch key)

目前使用 I2C 介面的 KEYPAD 模組有 itp_keypad_it7230.c，itp_keypad_it723br.c，itp_keypad_it7236.c 等。並最多使用到 3 個 GPIO PIN(INT+SDA+SCL)，有的 touch key IC 使用 polling 方式回報 key event，因此只會使用兩個 GPIO PIN(即 I2C 的兩個 PIN)。目前所支援的 touch key IC 最多可偵測 16 組 key event(使用

IT7236)。

2.2.4客製化 keypad driver

可參考“sdk\driver\itp\keypad\itp_keypad_gpio.c”，修改三個 ITP API 即可

1. int itpKeypadGetMaxLevel(void)
回傳要偵測的 key 的總數。

2. void itpKeypadInit(void)
實作 Keypad 模組的初始化。

3. int itpKeypadProbe(void)
實作 keypad 的偵測，並回傳 keypad mapping table 相對應的值。回傳值必須是 0~(KEYPAD 總數-1)之間的整數。keypad mapping table 則是定義在 Kconfig 的 Peripheral 的 Keypad Enable 底下的 Keypad mapping table 選項中(KEYPAD_MAPPING_TABLE)。檔案則放在“\sdk\target\keypad\”目錄下。此 Keypad mapping table 檔案將會在“\sdk\share\sdl\video\castor3\SDL_castor3keypad.c”這裡被讀取，轉換成 SDL 的 scancode(參考\sdk\include\SDL\SDL_scancode.h)。SDL 的 scancode 最後會被轉成 SDL 的 keycode(參考“\sdk\share\sdl\events\SDL_keyboard.c”)，提供上層 AP 透過 SDL API 擷取 key event(請參考本文件之範例 1)。所以 Keypad mapping table 內容的編寫即是 SDL 的 scancode 與 itpKeypadProbe()回傳值的對應表。

4. static const unsigned int kpGpioTable[] = { CFG_GPIO_KEYPAD };
CFG_GPIO_KEYPAD 是 Kconfig 的環境變數，在 GPIO 型式的 keypad driver 中，代表所要用到的 GPIO pin，若在 I2C 型式的 keypad driver 中，則代表 keypad INT pin 所要使用的 GPIO pin。

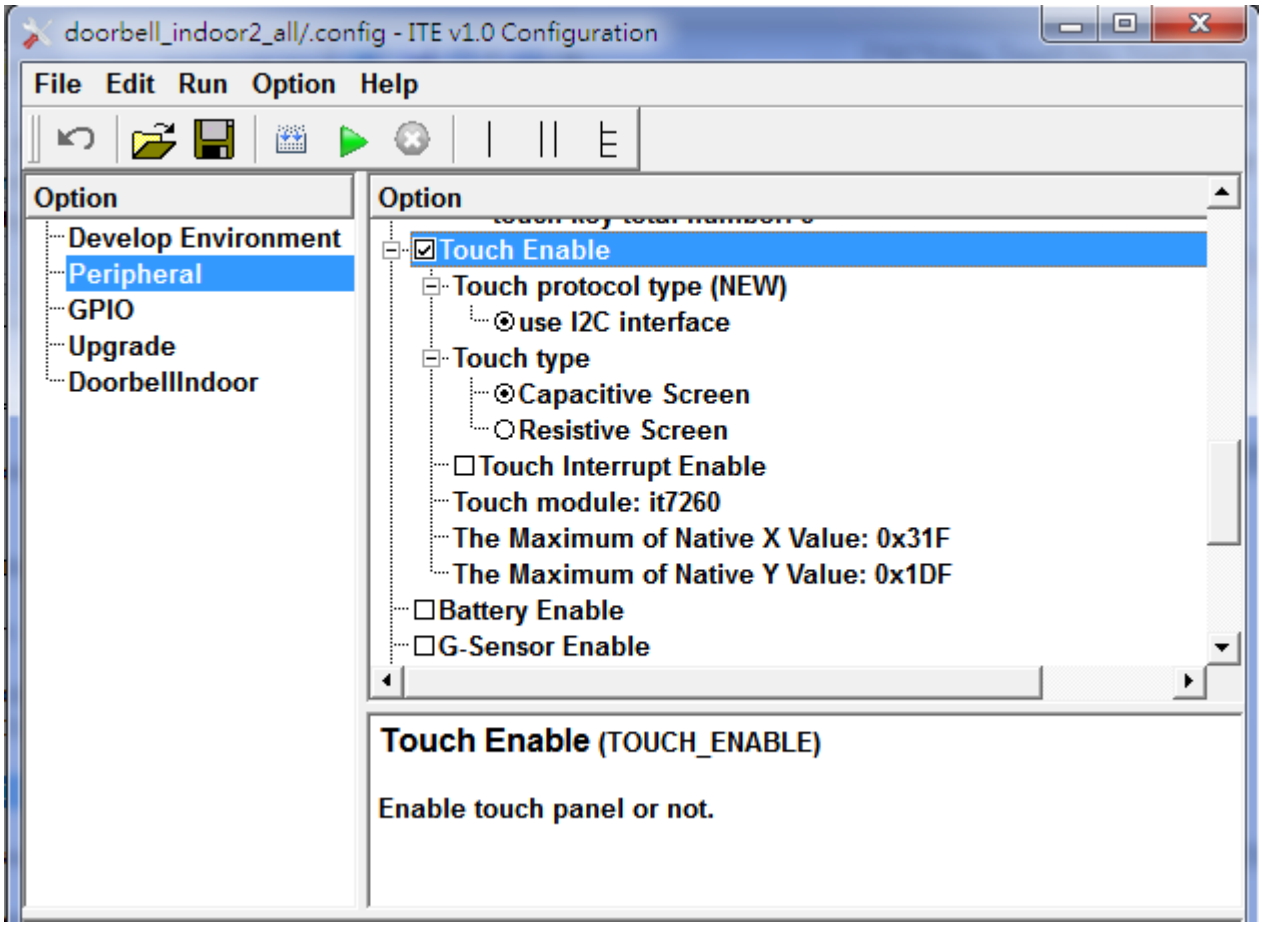
2.3 Touch Panel Driver in TSLIB

Tslib 是 OPEN SOURCE 的程式，可以之支援各種 TOUCH PANEL 並提供 FILTER，DEJITTE，CALIBRATION 等功能。TP 的 DRIVER 即是改寫 TSLIB 中 PLUGINS 目錄下的各種 TOUCH PANEL DRIVER。命名方式皆為 xxxxxx-raw.c (其中 xxxxxx 為觸控晶片名稱)，目前已支援的 touch IC 有 IT7260，ZT2083，TSC2046 等。

TSLIB 提供 ts_read()，ts_open()，ts_config，供上層 AP 呼叫(參考“sdk\share\sdl\video\SDL_castor3touch.c” 或是 “project\test_touch\test_touch.c”)。

SDL 提供 SDL_PollEvent( )函式用來取得 SDL 的各種 EVENT，可參考 SDL 的說明文件，或是參考 “project\test_touch\test_touch.c”。

2.3.1Touch Panel KCONFIG 設定



Touch Enable：啟動 touch panel 模組
Touch protocol type：可選擇 I2C 介面(要啟動 SPI 模組)或是 SPI 介面(要啟動 SPI 模組)。
Touch type：設定 touch 機制(使用電阻式還是電容式 touch panel)
Touch module：設定要使用的模組名稱(touch IC 名稱)

2.3.2I2C interface

使用 I2C 介面的 touch panel IC 有 IT7260，ZT2083，可參考“sdk\share\tslib\it7260-raw.c”與“sdk\share\tslib\zt2083-raw.c”。

TSLIB 透過 TSLIB_MODULE_INIT( )，將 xxxxxx_read( )等 operation 註冊到 TSLIB 模組。SDL 會在適當時機自動呼叫 tslib API(即 ts_read())讀取 touch panel 的座標資訊。所以修改 TSLIB 底層的 touch panel driver 主要是實作函式 xxxxxx_read( )。至於如何透過 I2C 讀取 touch panel 的座標資訊，請參考 UART/SPI/I2C 模組開發指南，以及各 touch panel 控制 IC 的 programming guide。

2.3.3SPI interface

使用 SPI 介面的 TOUCH PANEL IC 有 TSC2046，可參考“sdk\share\tslib\plugins\tsc2046-raw.c”。

TSLIB 透過 TSLIB_MODULE_INIT( )，將 xxxxxx_read( )等 operation 註冊到 TSLIB 模組。SDL 會在適當時機自動呼叫 tslib API(即 ts_read())讀取 touch panel 的座標資訊。所以修改 TSLIB 底層的 touch panel driver 主要是實作函式 xxxxxx_read( )。至於如何透過 SPI 讀取 touch panel 的座標資訊，請參考 UART/SPI/I2C 模組開發指南，以及各 touch panel 控制 IC 的 programming guide。

2.3.4支援多組觸控模組

SDK 新增支援多組觸控模組的功能。可讓 USER 只需建置一套 PKG，即可適用於不同觸控模組。

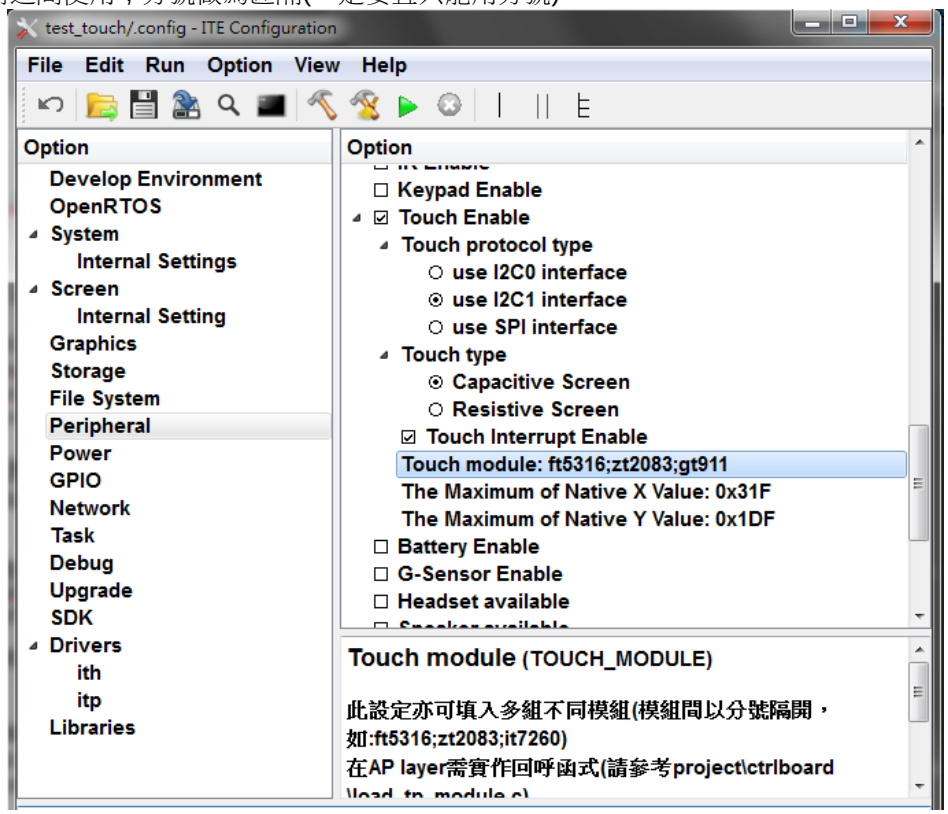
使用方式

步驟一：正確設定多組觸控模組

在 KCONFIG 的 Peripheral:TOUCH_MODULE 將其設定為多組模組

EX: ft5316;zt2083;gt911

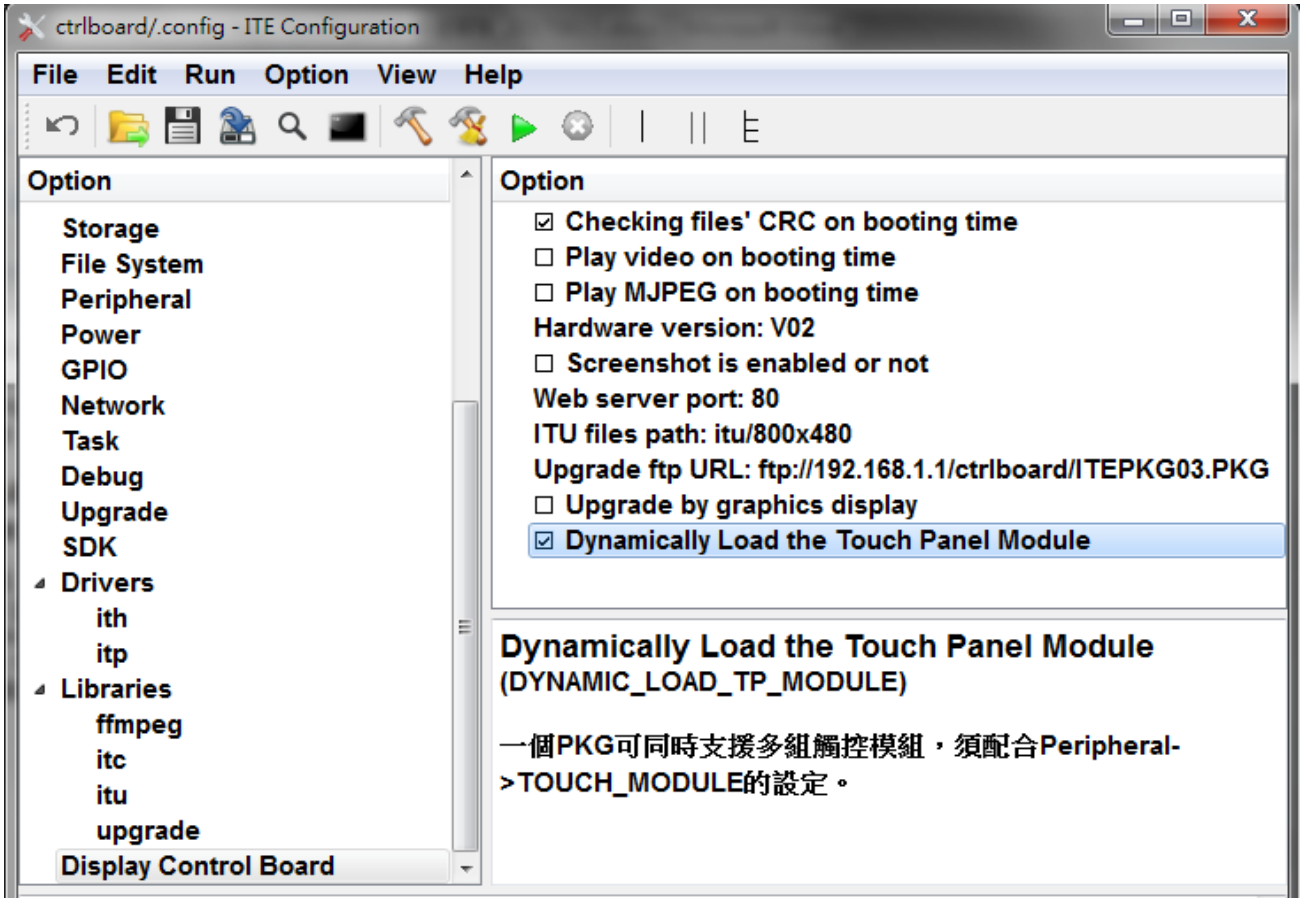
模組名稱之間使用";"分號做為區隔(一定要且只能用分號)



步驟二：實作 callback function

在 project 中實作一個 callback function。此 function 主要任務是辨識目前板端所使用的 TP module，並將 module name 回傳給 SDL。辨識 TP module 的機制、Slave Address 的 table，或是其他 callback function 需要的初始設定，都須由 USER 自行 porting，以便應付各種 project 的環境

目前 ctrlboard 以及 indoor2 這兩個 project 有提供此 callback function 的實作範例。並且由 project 中的 KCONFIG 的 **DYNAMIC_LOAD_TP_MODULE** 來決定是否開啟此功能(如下圖)。USER 亦可以此作為關鍵字搜尋相關的修改。在範例中，辨識機制是用 I2C slave address 下命令，觀察是否「slave device 會回 ACK 訊號」來判斷目前板端使用的觸控模組。此辨識方法已經在 it9854 觸控板 demo board 環境下驗證可行。只是此方法並非標準正規的辨識方式，無法百分之百保證都能辨識成功。必須視狀況自行修改辨識機制。



注意事項：
1.如果 Kconfig 的 TOUCH_MODULE 只設一組 MODULE，無論是否開啟 DYNAMIC_LOAD_TP_MODULE，SDL 都不會執行 callback function，且會載入 TOUCH_MODULE 所設定的 module

2.如果 Kconfig 的 TOUCH_MODULE 設定為多組 module，則一定會執行 callback function，所以必須正確實作 callback function，觸控才會正常運作

3.承 2，若 callback function 發生辨識錯誤，回傳錯誤的 module name，則 SDL 無防呆或 error handle 機制，觸控也一定不會正常運作。請務必確保 callback function 的辨識正確性。

2.3.5客製化 Touch Panel Driver

關於客製化 TOUCH PANEL DRIVER，可以任意新增 "\sdk\share\tslib\plugins\xxxxxx-raw.c (xxxxxx 要新增的 touch IC 名稱)，其模板程式如下所示。

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <alloca.h>
#include <pthread.h>
#include "ite/ith.h"
#include "ite/itp.h"
#include "config.h"
#include "tslib-private.h"

#define TP_GPIO_PIN         CFG_GPIO_TOUCH_INT

#ifdef       CFG_GPIO_TOUCH_WAKE
#define TP_GPIO_WAKE_PIN CFG_GPIO_TOUCH_WAKE
#endif

#ifdef       CFG_TOUCH_INTR
#define ENABLE_TP_INTR_MODE
#endif
/*#############################################################################################
 *                    the private function implementation
 ############################################################################################*/


/*#############################################################################################
 #                    public function implementation
 ############################################################################################*/
static int xxxxxx_read(struct tslib_module_info *inf, struct ts_sample *samp, int nr)
{
       //TO DO:get touch sampe and return sample & nr(the total sample count)
       return nr;
}

static const struct tslib_ops xxxxxx_ops =
{
       xxxxxx_read,
};
```

```
TSAPI struct tslib_module_info *castor3_mod_init(struct tsdev *dev, const char *params)
{
	struct tslib_module_info *m;

	m = malloc(sizeof(struct tslib_module_info));
	if (m == NULL)
		return NULL;

	m->ops = &xxxxxx_ops;
	return m;
}

#ifndef TSLIB_STATIC_CASTOR3_MODULE
	TSLIB_MODULE_INIT(castor3_mod_init);
#endif
```

客製 TOUCH PANEL 驅動程式之注意事項

- CFG_GPIO_TOUCH_INT 是 KCONFIG 所設定的“Touch INT Pin”使用的 GPIO PIN(視需要決定是否使用)
- CFG_GPIO_TOUCH_WAKE 是 KCONFIG 設定“Touch Wake Pin”使用的 GPIO PIN(視需要決定是否使用)
- CFG_TOUCH_INTR 是 KCONFIG 中“Touch interrupt Enable”是否使用中斷的設定(視需要決定是否使用)
- 模板程式中 xxxxxx 需跟檔名一致
- xxxxxx_read()需要自行實作，傳入的參數有 struct tslib_module_info *inf、struct ts_sample*samp、int nr。struct tslib_module_info 請參考 “sdk\share\talib\src\tslib-filter.h”， struct ts_sample 請參考 “sdk\include\talib.h”。nr 是指可支援的 sample 數，回傳值是此次回傳的 sample 數。相關代碼的實作與應用可參考其他 xxxxxx-raw.c 檔。

3. 範例

3.1 範例 1 (keypad)

```
#include <sys/ioctl.h>
#include <stdio.h>
#include <unistd.h>
#include <stdbool.h>
#include "SDL/SDL.h"
#include "ite/itp.h"

int TouchEvent_test(void)
{
    SDL_Event ev;

        /* SDL initial */
    if (SDL_Init(SDL_INIT_VIDEO) < 0)
        printf("Couldn't initialize SDL: %s\n", SDL_GetError());

        while (SDL_PollEvent(&ev))
        {
            switch (ev.type)
            {
          case SDL_KEYDOWN::
              switch (ev.key.keysym.sym)
              {
              case SDLK_UP:
                    printf("key direction up\n");
                 break;

              case SDLK_DOWN:
                    printf("key direction up\n");
                 break;

              case SDLK_LEFT:
                    printf("key direction up\n");
                 break;

              case SDLK_RIGHT:
                    printf("key direction up\n");
                 break;
           }
          case SDL_KEYUP:
              printf("key direction up\n");
               break;
           }
```

```
        SDL_Delay(1);
        }
}
```

3.2 範例 2(touch panel)

```
#include <sys/ioctl.h>
#include <stdio.h>
#include <unistd.h>
#include <stdbool.h>
#include "SDL/SDL.h"
#include "ite/itp.h"

int TouchEvent_test(void)
{
    SDL_Event ev;

        /* SDL initial */
    if (SDL_Init(SDL_INIT_VIDEO) < 0)
        printf("Couldn't initialize SDL: %s\n", SDL_GetError());

    for (;;)
    {
        bool result = false;

        while (SDL_PollEvent(&ev))
        {
            switch (ev.type)
            {
            case SDL_FINGERDOWN:
                printf("touch: down %d, %d\n", ev.tfinger.x, ev.tfinger.y);
                break;

            case SDL_FINGERUP:
                printf("touch: up %d, %d\n", ev.tfinger.x, ev.tfinger.y);
                break;

            case SDL_SLIDEGESTURE:
                switch (ev.dgesture.gestureId)
                {
                case SDL_TG_LEFT:
                    printf("touch: slide to left\n");
                    break;

                case SDL_TG_UP:
                    printf("touch: slide to up\n");
```

```
                break;

            case SDL_TG_RIGHT:
                printf("touch: slide to right\n");
                break;

            case SDL_TG_DOWN:
                printf("touch: slide to down\n");
                break;
            }
            break;
        }
      }
      SDL_Delay(1);
   }
}
```