Chip driver guidance document

ÿÿÿ

Date	**2021/4/14**
Project **ID**	**Xxx**
Document	**V3.1**
Version Driver Version	**V2.2**
App Version	**V1.25**
Release	**2021/4/14**
Date Author	**Steven**

Revision History:

Version	Date	Author
V2.0	2020/8/13	Steven
V3.0	2021/4/13	Steven
V3.1	2021/4/14	Steven

Change of resume:

Version **v2.0**	First Edition
	1. Compatible with the chip series, support **CST1XX/CST2XX/CST3XX/CST7XX/CST8XX /CST1XXSE/CST2XXSE/CST3XXSE** 2. Support **apk** firmware upgrade, read version number, factory test, read **raw** and **diff** functions. 3. Support **ESD** detection function. 4. Support gesture wake-up function. 5. Support proximity sensing function.
Version **V3.0**	1. Support **CST9XX** 2. Support **factory** test open and short circuit 3. Force **DTS** to match and get **rst/int/resolution**
Version **V3.1**	1. Modify the firmware configuration content

Contents1 . **Target**.....**3**

1. Goal

This document is mainly used to introduce the framework architecture of the Haiyechuang touch chip driver, the configuration and debugging of the driver, to facilitate **FAE** colleagues and solution companies to debug the driver.

Includes the main functions of the driver, driver configuration, document structure, and migration steps.

2. Applicable chip types

Basic Information	
Supported chip types	**CST1XX: CST126 CST128 CST130 CST140 CST14055 CST148** **CST1XXSE:CST128SE**

	CST2XX: CST226 CST237 CST240 **CST2XXSE: CST226SE** **CST3XX: CST326 CST328 CST340 CST348** **CST7XX: CST716 CST726 CST736** **CST8XX: CST816 CST826 CST836U** **CST9XX: CST912 CST918** **CST6XX: CST6928S**
Supported platforms	Support Android platform (MTK/Qualcomm/Allwinner/Rockchip/Spreadtrum)
ADB, apk function support	
Other Features	**GESTURE ESD PROXIMITY**

3. File structure

The driver files are stored in the **hynitron** folder, which implements functions such as driver mounting, touch point reporting, sleep wake-up, gesture wake-up, FW upgrade, and **APK** and **ADB** debugging.

The following list is a brief introduction to the functions of each file:

file name	Attribute function
Makefile	Required **Makefile**
Kconfig	Required **Kconfig** file
xxx_core.c	The main function file of the driver is required to realize the functions of driver mounting, reading and reporting of touch data, and sleep and wake-up.
xxx_core.h	The main function header file of the required driver includes project information (each project needs to be configured separately), main structure Body type.
Hynitron_config.h	Mandatory **Enable** and **Disable** header files for configurable function modules
Hynitron_common.h	Required chip type and register definition, other file function external declaration, print information definition
Hynitron_esd_check.c	Required **ESD** detection function file
Hynitron_gesture.c	Optional gesture wake-up function file
Hynitron_i2c.c	Required **IIC** communication function file
Hynitron_proximity.c	Optional proximity sensing function file
Hynitron_tool_debug.c can select Android sys and pro nodes for **adb** and **apk** debugging. It is strongly recommended to add this function.	
Hynitron_update_firmware.cRequired firmware update function file	
Hynitron_update_firmware.hRequired firmware update header file	
/firmware	Required Firmware files used for firmware upgrade
/docs	Optional **Dts** configuration

4. Compile configuration

4.1 Modify the compiled file

Package the driver files into the **hynitron** folder and copy **the hynitron** folder to **the kernel/dirvers/input/touchscreen** directory.

(1) Modify the **Kconfig** file in **the touchscreen** directory and add the following line to the end of the file:

Source “drivers/input/touchscreen/hynitron/Kconfig”

(2) Modify the **Makefile** file in **the touchscreen** directory and add the following line to the end of the file:

Obj-$(CONFIG_TOUCHSCREEN_HYNITRON_TS) +=hynitron/

or **obj-y += hynitron/**

4.2 Compile Commands

(1) Call up **menuconfig and select TOUCHSCREEN_HYNITRON_TS**

(2) Compile **bootimage**

$ make bootimage -j32

The driver file is compiled by default and there is no need to modify the above options.

5. Driver function configuration

5.1 DTS Configuration

Example:

Example:

i2c@f9927000

{ hynitron@1a{

compatible = "hynitron,hyn_ts"; reg =

<0x1a>;

hynitron,reset-gpio = <&gpio 12 0x01>;

hynitron,irq-gpio = <&gpio 13 0x02>;

hynitron,max-touch-number = <5>;

hynitron,display-coords = <1080 1920>;

hynitron,have-key;

hynitron,key-number = <3>;

hynitron,key-code = <139 172 158>;

hynitron,key-y-coord = <2000 2000 2000>; hynitron,key-

x-coord = <200 600 800>;

};

};

DTS needs to contain the following

information: **1. IIC address (reg** defaults to 0x1A, which can be changed in special

cases) **2. Property name (compatible** must be consistent with the internal definition of the driver, otherwise

the driver cannot be loaded) **3. Interrupt pin**

(hynitron, irq-gpio) 4. Reset pin (hynitron, reset-

gpio) 5. Maximum touch finger index (hynitron, max-touch-number) 6.

Resolution (hynitron, display-coords) 7. Key information

(if there is a key, it must be configured) **Note:** In addition

to the key information, other information of DTS must be configured, otherwise **the DTS** parsing error will occur and it cannot be loaded. If it is confirmed that DTS cannot be modified, you c

Modify the function **hyn_parse_dt and assign values directly.**

5.2 Functional modules and project information configuration (mandatory)

5.2.1 Configure **Hynitron_core.h** file (project information)

Project information configuration:

Please select the corresponding **IC** type and project **ID information (Hynitron_core.h):**

```
#define HYN_CHIP_TYPE_CONFIG                CST340
#define HYN_IRQ_TRIGGER_RISING_CONFIG       0x01
#define HYN_MAIN_IIC_ADDR_CONFIG            0x1A
```

The project configuration is as above. When starting a new project, please ask the driver engineer or **FAE** engineer to configure the following information:

CHIP_TYPE chip type: **CST340** (required, if the chip type is incorrectly selected, it may cause the chip detection to fail, the firmware to fail to upgrade, and the driver to fail to load)

TRIGGER_RISING rising edge: **0x00** (optional, default falling edge triggers interrupt)

```
#define HYN_X_DISPLAY_DEFAULT       720
#define HYN_Y_DISPLAY_DEFAULT       1280
#define HYN_X_REVERT                0
#define HYN_Y_REVERT                0
#define HYN_XY_EXCHANGE             0
#define HYN_MAX_KEYS                3
#define HYN_MAX_POINTS              5
#define HYN_MAX_SELFCAP_ID          2
```

X_DISPLAY: 720 (default, if **dts** acquisition fails, use this value)

Y_DISPLAY: 1280 (Default, if **dts** acquisition fails, use this value)

X_REVERT: 0 (default, changes the coordinate direction in **the X** direction)

Y_REVERT: 0 (default, changes the coordinate direction in the **Y** direction)

XY_EXCHANGE: 0 (default, swap X and Y)

MAX_KEYS: 3 (default, 3)

MAX_POINTS: 5 (default, needs to be changed to 2)

Other macro configurations:

HYN_RESET_SOFTWARE	Enable software watchdog reset function	**Disable**
HYN_UPDATE_FIRMWARE_POEWRON_ENABLE enables power-off reset upgrade function		**Disable**
HYN_UPDATE_FIRMWARE_ENABLE	Enable firmware upgrade function	**Disable**
HYN_UPDATE_FIRMWARE_FORCE	Enable the forced firmware upgrade function	**Disable**
HYN_IIC_TRANSFER_LIMIT	Enable **IIC** communication byte length limit function	**Disable**

5.2.2 Configure the **hynitron_config.h** file (driver function module configuration)

Configure the function module:

Function module macro (hynitron_config.h) function		default
HYN_DEBUG_EN	Print **debug log** information for debugging. It is recommended to turn it off in the user version .	**Enable**
HYN_MT_PROTOCOL_B_EN	**Linux** multi-touch protocol switch, enable (B **protocol), disable (A** protocol) **Enable**	
HYN_REPORT_PRESSURE_EN	**Multi-Touch A/B** reports **pressure** value, enabled by default	**Enable**
HYN_GESTURE_EN	Gesture function switch, enable (on), disable (off)	**Disable**
HYN_PSENSOR_EN	Proximity sensor switch, enable (on), disable (off)	**Disable**
THIS_ESDCHECK_EN	The **ESD** protection mechanism detects once every **1s** and resets **the IC if an abnormality occurs.**	**Enable**
HYN_AUTO_FACTORY_TEST_EN starts the factory test verification function to detect **tp** consistency.		**Disable**
HYN_EN_AUTO_UPDATE	Automatic firmware upgrade function. Enable (on), disable (off)	**Disable**
HYN_SYS_AUTO_SEARCH_FIRMWARE firmware automatic query upgrade function. enable (on), disable (off)		**Disable**
ANDROID_TOOL_SURPORT	Android platform **proc** node generation. Used for **apk** debugging.	**Enable**
HYN_SYSFS_NODE_EN	Android platform **sys** node generation. Used for **adb** debugging.	**Enable**

Configure the chip type supported by the firmware upgrade function (corresponding to chip selection):

Upgrade function macro (hynitron_config.h)	Function	default
HYN_EN_AUTO_UPDATE_CST0xxSE	Enable **the CST0XXSE** series chip upgrade function. Including **CST016SE/CST026SE/CST036SE**	**Disable**
HYN_EN_AUTO_UPDATE_CST0xx	Enable **the CST0XX** series chip upgrade function. Including **CST016/CST02E/CST036**	**Disable**
HYN_EN_AUTO_UPDATE_CST1xx	Enable **the CST1XX** series chip upgrade function. Including **CST126/CS130/CST140/CST14055/CST148**	**Disable**
HYN_EN_AUTO_UPDATE_CST1xxSE	Enable **the CST1XXSE** series chip upgrade function. Including **CST128SE/CST18858SE/CST18868SE**	**Disable**
HYN_EN_AUTO_UPDATE_CST2xx	Enable **the CST2XX** series chip upgrade function. Including **CST226/CST237/CST240**	**Disable**
HYN_EN_AUTO_UPDATE_CST2xxSE	Enable **the CST2XXSE** series chip upgrade function. Includes **CST226SE**	**Disable**
HYN_EN_AUTO_UPDATE_CST3xx	Enable **the CST3XX** series chip upgrade function. Includes **CST326/CST328/CST340/CST348**	**Disable**
HYN_EN_AUTO_UPDATE_CST3xxSE	Enable **the CST3XXSE** series chip upgrade function. Including **CST328SE**	**Disable**
HYN_EN_AUTO_UPDATE_CST78xx	Enable **the CST78XX** series chip upgrade function. Including **CST716/CST726/CST736/CST816/CST826/CST836U**	**Disable**
HYN_EN_AUTO_UPDATE_CST6xx	Enable **the CST6XX** series chip upgrade function. Including **CST6928S**	

HYN_EN_AUTO_UPDATE_CST9xx	Enable **the CST9XX** series chip upgrade function. Including **CST912 CST918**	

Note: If you cannot enter **bootloader** mode during the upgrade , please confirm the reset method:

1. Power off reset

2. Reset pin reset

3. Watchdog reset

The window period for entering **bootloader** mode is generally 5ms~20ms after resetting the chip, and commands sent within this period are valid.

5.2.3 Firmware information configuration (must be modified)

Configuration **IC** type:

#define HYN_CHIP_TYPE_CONFIG configure **CST340** **ÿhynitron_core.hÿ**

firmware (hynitron_update_firmware.c):

```
00026: #include "firmware/capacitive_hynitron_cst0xx_update.h"
00027: #include "firmware/capacitive_hynitron_cst2xx_update.h"
00028: #include "firmware/capacitive_hynitron_cst2xxse_update.h"
00029: #include "firmware/capacitive_hynitron_cst3xx_update.h"
00030: #include "firmware/capacitive_hynitron_cst3xxse_update.h"
00031: #include "firmware/capacitive_hynitron_cst6xx_update.h"
00032: #include "firmware/capacitive_hynitron_cst8xx_update.h"
00033: #include "firmware/capacitive_hynitron_cst9xx_update.h"
00034:
00035: //please config the chip series before using.
00036: struct hynitron_fw_array hynitron_fw_grp[] = {
00037:     //0-name; 1-fw; 2-project_id; 3-module_id; 4-chip_type; 5-fw_length;
00038:     { "capacitive_hynitron_cst0xx_update",   cst0xx_fw,   0x2843,0x01, CST016, (sizeof(cst0xx_fw))},
00039:     { "capacitive_hynitron_cst2xx_update",   cst2xx_fw,   0x0501,0x01, CST226, (sizeof(cst2xx_fw))},
00040:     { "capacitive_hynitron_cst2xxse_update", cst2xxse_fw, 0x0501,0x01, CST226SE, (sizeof(cst2xxse_fw))},
00041:     { "capacitive_hynitron_cst3xx_update",   cst3xx_fw,   0x2117,0x11, CST348, (sizeof(cst3xx_fw))},
00042:     { "capacitive_hynitron_cst6xx_update",   cst6xx_fw,   0x2117,0x11, CST6928S, (sizeof(cst6xx_fw))},
00043:     { "capacitive_hynitron_cst3xxse_update", cst3xxse_fw, 0x0501,0x01, CST328SE, (sizeof(cst3xxse_fw))},
00044:     { "capacitive_hynitron_cst8xx_update",   cst8xx_fw,   0x0501,0x01, CST836, (sizeof(cst8xx_fw))},
00045:     { "capacitive_hynitron_cst9xx_update",   cst9xx_fw,   0x2208,0x01, CST918, (sizeof(cst9xx_fw))},
00046:
00047: };
```

The following contents need to be modified according to the firmware:

(1) Replace the .h file of the corresponding

chip (2) Modify **the hynitron_fw_grp[ ]** corresponding firmware, project ID, module ID, chip type. (3) If a project has

multiple **TP** factories, you can add the corresponding header file and identify it according to the project **ID** and module **ID .**

5.3 Mass production test configuration (optional)

The configuration of mass production test parameters needs to be used with the

Hyntronic test **apk . Open the macro: ANDROID_TOOL_SURPORT (hynitron_config.h)**

Generate **proc nodes: /proc/cst1xx_ts/cst1x-update (mutual capacity) /proc/**

cst8xx_ts/cst8xx-update (self-capacity) Hyntronic test **APK:**

Hyntronic_TP_Tools(1.25).apk (the version cannot be lower than 1.25)

5.3.1ÿInstall test **apk**

adb install C:\Users\steven_wu\Desktop\Hyntronic_TP_Tools(1.25).apk

5.3.2, Modify selinux permissions

adb shell setenforce 0

If you need to use the **user** version, please modify the upper **te** file and set the test **apk** to the system **apk.**

Here is an example:

1. Add in file_context: /proc/cst1xx_ts/cst1xx-update u:object_r:cst1xx_ts:s0 2. Add in **File.te : type**

cst1xx-update ,fs_type,proc_type; 3. Add in **untrusted_app_25.te**

allow untrusted_app_25 cst1xx-update :file { read write getattr ioctl open};

For details, please consult Android upper-level **apk** colleagues.

5.3.3ÿ**Configure apk** factory test parameters

Factory test data save path: /sdcard/Android/data/com.hyn.tp_updatefw/cache/cstxxx/ Configuration file

name: hyn_mutualcap_testconfig.ini Configuration file

path: /system/etc/hyn_mutualcap_testconfig.ini

Adb root

Adb remount

Adb push /..hyn_mutualcap_testconfig.ini /system/etc/

Start **APK** Factory automatic testing:

Adb shell am start -n com.hyn.tp_updatefw/.FactoryActivity

6. adb debugging node

Step 1: Modify **selinux** permissions: **adb shell setenforce 0**

There are two **adb** debugging nodes generated :

1. /proc/cst1xx_ts/cst1xx-update (open the macro **ANDROID_TOOL_SURPORT)**

2. /sys/hynitron_debug (open the macro HYN_SYSFS_NODE_EN)

Check the version number: cat **hyntpfwver**

Firmware Upgrade:

Echo “1” >hynfwupdate //Automatically upgrade the firmware integrated

Echo “HYN_CST1_1.bin” >hynfwipgradeapp into the driver. //**Automatically upgrade /mnt/HYN_CST1_1.bin, and ensure that** the path of **HYN_CST1_1.bin is /mnt.**

Interrupt **enable :** **Echo “88” >hyntprwreg**

Interrupt **disable:** **Echo “99” >hyntprwreg**

Reset chip: **Echo “77” >hyntprwreg**

7. apk debugging

Step 1: Modify **selinux** permissions: **adb shell setenforce 0** Step 2:

Install the test **apk: adb install C:\Users\steven_wu\Desktop\Hyntronic_TP_Tools(1.25).apk (the version cannot be lower than 1.25)**

1. DrawLine

2. Draw a line

3. MultiTouch

4. Multi-finger touch

5. Update Firmware

6. Firmware Update

7. Data Analysis

8. Data analysis, including **rawdata** diff, mutual capacitance and self-capacitance signals

9. Manual Test

10. Manual test, mutual capacitance will obtain factory data **Delta High Short**

11. Self-Test

12. Automatic testing, as the name implies, automatically obtains factory data and then makes judgments based on data analysis. This item depends on the test configuration file.

13. About

14. Software Introduction

15. Exit

16. Exit

7.1 Data Retention

Diff and **rawdata** are not saved by default. When **Savedata** is checked on the **DataAnalysis** interface, the data will be automatically saved to **/sdcard/Android/data/com.hyn.tp_updatefw/cache/cstxxx/ and** save in .csv format.

Factory data is saved by default, the directory is the same as above.

7.2 Introduction to Rawdata/Diff Mutual Compatibility Interface

When Apk automatically identifies a mutual-capacitance chip, several semi-transparent radio buttons and check boxes will appear in the upper left corner of the DataAnalysis interface. The functions are as follows:

Radio button:

Raw: Read **Rawdata** data

Diff: Read Diff data

Checkbox:

Reversal: Swap the data left and right. Because the relative positions of TX and RX of the tablet and the phone are different, and for debugging purposes, you can select the option to adjust the data display position (yes The current position is still symmetrical)

Savedata: Save data

SelfCapData: If checked, the self-capacitance signals of **Rx** and **Tx** will be displayed on the right and bottom .

7.3 Updating the firmware

ÿ Before entering **Update Firmare ,** please push the firmware to the /sdcard/ directory via **adb .** The reference command is as follows:

adb push **.bin /sdcard/

ÿThen select the firmware **through the "OPEN** FILE" button

ÿClick "UPDATE" to update. After the progress bar ends, the interface will automatically refresh the number of **tx and rx** and the firmware version.

Start command:

am start -n com.hyn.tp_updatefw/.MainActivity

am start -n com.hyn.tp_updatefw/.DrawLineActivity

am start -n com.hyn.tp_updatefw/.UpdateFwActivity

am start -n com.hyn.tp_updatefw/.DataAnalyzeActivity

am start -n com.hyn.tp_updatefw/.ManualTestActivity

am start -n com.hyn.tp_updatefw/.FactoryActivity

Schedule:

Commonly used **adb** commands:

adb shell cat /proc/kmsg | grep "HYN"

adb shell cat /proc/kmsg > /mnt/sdcard/log

adb pull /mnt/sdcard/log C:\Users\Administrator\Desktop

adb logcat > C:\Users\twl\Desktop\log\logcat.log

Adb shell settings put system show_touches **1** Open the line drawing interface

adb shell settings put system pointer_location **1** Open pointer position

adb shell setprop debug.layout true Open Layout

adb shell ll /sys/bus/i2c/drivers View device mounts

adb shell getevent

adb shell getevent -l Reporting event

adb shell getevent -r Reporting rate

adb shell getevent -r **/dev/input/event5**

adb shell input keyevent 3 Back key

adb shell input keyevent 4 Home button

adb shell input keyevent 26 **power** button

adb shell setenforce **0** Turn off **selinux**

adb push C:\Users\steven_wu\Desktop\apk\hyn_mutualcap_testconfig.ini /system/etc/ ÿÿ **push** configuration file

Adb shell input tap x y

Adb shell Input swipe x1 y1 x2 y2 adb shell slide

settings put system screen_off_timeout 600000 Set the LCD screen off time

adb shell am start com.android.settings/com.android.settings.Settings //Open phone settings

8. Gesture wake-up function

8.1 Gesture Initialization

Supported gesture events:

#define KEY_GESTURE_U **KEY_U**

#define KEY_GESTURE_UP **KEY_UP**

#define KEY_GESTURE_DOWN **KEY_DOWN**

#define KEY_GESTURE_LEFT **KEY_LEFT**

#define KEY_GESTURE_RIGHT **KEY_RIGHT**

#define KEY_GESTURE_O **KEY_O**

#define KEY_GESTURE_E **KEY_E**

#define KEY_GESTURE_M **KEY_M**

#define KEY_GESTURE_W **KEY_W**

#define KEY_GESTURE_S **KEY_S**

#define KEY_GESTURE_V **KEY_V**

#define KEY_GESTURE_C **KEY_C**

#define KEY_GESTURE_Z **KEY_Z**

#define KEY_GESTURE_DOUBLECLICK Gesture code **KEY_POWER**

reported by gesture:

Node that generates gesture information:

/sys/hynitron_debug/hyn_gesture_mode //Gesture mode status node

/sys/hynitron_debug/hyn_gesture_buf //Gesture reporting data node

8.2 Gesture reporting

Gesture data structure:

```
struct hyn_gesture_st
{
    u8 header[HYN_GESTRUE_POINTS_HEADER];
    u16 coordinate_x[HYN_GESTRUE_POINTS];
    u16 coordinate_y[HYN_MANAGEMENT_POINTS];
     u16 report_key;            //The event code ID reported by the driver
     u8 gestrue_id;             //Gesture code ID reported by the chip
     u8 mode;                   // Gesture wake-up function switch
     u8 active;                 // Gesture detection is on, set when the screen is off
};
```

Gesture function flow:

Step 1: Register gesture wake-up supported events in **hyn_probe** and register gesture nodes.

Step 2: When the screen is off, send a gesture detection command. Set the interrupt: low level trigger, no sleep. (The low level should be maintained for **200ms)**

Step 3: Interrupt trigger, read the gesture code, report **the input** layer event code corresponding to the gesture code, and wake up the screen.

9. Proximity sensing function

9.1 Proximity Sensing Initialization

The implementation of the proximity sensing function mainly depends on the matching of the driver and the upper layer of Android to realize the opening and closing of the proximity sensing function, as well as the transmission of approaching and distant data.

In the **hyn_proximity_init** function:

(1) Initialize the input device that implements proximity sensing **and** set the support event **EV_ABS**

```
hyn_proximity_data->ps_input_dev = input_allocate_device();
__set_bit(EV_ABS, hyn_proximity_data->ps_input_dev->evbit);
```

input_set_abs_params(hyn_proximity_data->ps_input_dev, ABS_DISTANCE, 0, 1, 0, 0);

ret= input_register_device(hyn_proximity_data->ps_input_dev);

(2) Register proximity sensing nodes:

/sys/hynitron_debug/hyn_proximity_mode //Proximity sensing mode status

node /**sys/hynitron_debug/hyn_proximity_buf //**Proximity sensing reporting data node

hyn_create_proximity_sysfs(hyn_ts_data->client);

Node operation:

cat hyn_proximity_mode //View node information and proximity sensing status **echo 01 >**

hyn_proximity_mode //Write node information and turn on proximity sensing for simulation debugging

(3) Initialize the interface function for proximity sensing reporting

Transplant the corresponding proximity sensing reporting implementation method according to the platform. For details, see the reference driver.

9.2 Proximity Sensing Report

Currently, the reporting method of proximity sensing varies depending on the platform. The mainstream reporting methods are as follows:

(1) Spreadtrum Platform

The first method: Register character miscellaneous devices:

```
err = misc_register(&tp_ps_device); //The structure tp_ps_device defines the operation interface function.
static int tp_ps_release(struct inode *inode, struct file *file);
static long tp_ps_ioctl(struct file *file, unsigned int cmd, unsigned long arg)
static struct file_operations tp_ps_fops = {
    .owner                  = THIS_MODULE,
    .open                   = tp_ps_open,
    .release                = tp_ps_release,
    .unlocked_ioctl = tp_ps_ioctl,
};
static struct miscdevice tp_ps_device = {
    .minor = MISC_DYNAMIC_MINOR,
    .name = TP_PS_DEVICE,                   //Usually the device name is ltr_558als
    .fops = &tp_ps_fops,
};
```

In the above implementation method, the upper layer will access the device: ltr_558als, and perform read and write operations through the **ioctl** operation interface function to detect approach and distance.

The second method: register **class** node:

```
firmware_class = class_create(THIS_MODULE,"sprd-tpd");//client->name
firmware_cmd_dev = device_create(firmware_class, NULL, 0, NULL, "device");//device
if(device_create_file(firmware_cmd_dev, &dev_attr_proximity) < 0)        // /sys/class/sprd-tpd/device/proximity
input_dev = input_allocate_device();
```

static DEVICE_ATTR(proximity, S_IRUGO | S_IWUSR, show_proximity_sensor, store_proximity_sensor); Register **class** device and

generate node: /**sys/class/sprd-tpd/device/proximity** Register input device and report

approach and distance events:

```
input_report_abs(tp_ps->input, ABS_DISTANCE, dps_data);
input_sync(tp_ps->input);
```

The upper layer accesses **the sys** node **proximity** to issue proximity sensing on and off

commands. This method separates write data from read data, that is, write data is issued through the node **proximity** , and read data is read through **the input** device.

(2) Mtk platform

The first method:

define the structure: struct hwmsen_object obj_ps;

obj_ps.polling = 0;//interrupt mode

obj_ps.sensor_operate = tpd_ps_operate;

ÿÿ **ID_PROXIMITYÿ if((err = hwmsen_attach(ID_PROXIMITY, &obj_ps)))**

static int tpd_ps_operate(void* self, uint32_t command, void* buff_in, int size_in,void* buff_out, int size_out, int* actualout)

This method relies on the **hwmsen** design of the **MTK** platform **and implements reading and writing data through the operation interface function: tpd_ps_operate.**

The header files that need to be included are:

#include <hwmsensor.h>

#include <hwmsen_dev.h>

#include <sensors_io.h>

Second method:

Register the light sensor driver: **alsps_driver_add(&ps_init_info);**

struct alsps_init_info ps_init_info = {

.name = "hyn_ts",

.init = ps_local_init,

.uninit = ps_local_uninit,

};

Define the interface function:

struct ps_control_path ps_ctl = { 0 };

struct ps_data_path ps_data = { 0 };

ps_ctl.open_report_data = ps_open_report_data; //Report dataps_ctl.enable_nodata **=**

ps_enable_nodata; //Send command **dataps_data.get_data = ps_get_data; //**Read

data

This method refers to the light sensor driver loading process to define the corresponding operation function, thereby realizing the interaction between the upper layer and the driver. However, this method cannot achieve compatibility, and the registration function must be placed at the driver entrance, not in **the probe .**

Header files that need to be included:

#include <alsps.h>

10. Driver loading process

10.1 Driver Entry Function

static int __init hynitron_driver_init(void) 10.2 Loading

I2C driver

ret = i2c_add_driver(&hynitron_i2c_driver); //Note to check **the dts** configuration **compatible, it must be the same as the hyn_dt_match** array.

10.3 Execute the **probe** function

1. hyn_platform_data_init(ts_data); //Initialize platform-related data and parse **dts** configuration //Initialize

2ÿhyn_gpio_configure(); **gpio_request** to apply for **IRT and RST** pins

3ÿhyn_ts_data_init(client); // Initialize **hyn_ts_data** structure data, including project ID, chip type, etc.

4. hyn_detect_bootloader(client); //Detect whether the chip **boot** mode can be entered and confirm the chip type. **5.**

hyn_input_dev_int(ts_data); //Initialize the input device input, set the reporting point **A/B** protocol, and create a reporting point work queue.

service routine. **6.** //Initialize interrupt registration, including rising edge and interrupt

hyn_irq_init(client); 7. hyn_update_firmware_init(client); //To upgrade the firmware, you must configure the correct chip type and project ID before you can upgrade.

8ÿhynitron_proc_fs_init(); **//** Generate proc/node information for **apk** debugging.

9. hyn_create_sysfs(client); 10. //Generate **sys/hynitron_debug** node for **debugging .**

hyn_gesture_init(hyn_ts_data->input_dev, client); // Initialize the gesture wake-up function and generate gesture nodes.

11ÿthis_proximity_init(); // Initialize the proximity sensing function and generate a proximity sensing node.

12. hyn_init_esd_protect(); 10.4 **//ESD** protection function initialization, time period **1s.**

Touch information reporting

1. The touch chip pulls an interrupt pulse.

2. Trigger the driver's interrupt service routine.

3. Add the report **work** to the corresponding work queue.

4. Report touch data to **the input** layer.

5. The Android layer processes and displays the coordinates.

11. Register Description

11.1 Mutual Compatibility Product Register:

Touch information register (ENUM_MODE_NORMAL mode)

(1) The touch information must be in **normal** mode, otherwise the data is abnormal (write **0xD109** to enter).

(2) Data reading must read **the 7** bytes of the first finger according to **0xD000 ,** including the number of fingers and the number of keys.

(3) Send the read data to complete the synchronization data (write **0xD000AB).**

Here is an example:

0x1A	IN	0xD0	0x00					
0x1A	R	0x06	0x33	0x56	0x68	0x8F	0x01	0xAB
0x1A	IN	0xD0	0x00	0xAB				

(4) To read the subsequent multi-finger data, allocate **5** bytes to each finger and read according to the address.

For details, please refer to the reporting function **cst3xx_touch_report .**

register address	High four				Lower four bits			
	bit7	**bit6**	**bit5**	**bit4**	**bit3 bit2**	**bit1 bit0**		
0xD000 1st finger **ID**					**1st finger status: pressed (0x06) or lifted**			
0xD001 The high eight bits of the **X** coordinate value of the 1st finger :					**X_Position>>4**			
0xD002 The high eight bits of the **Y** coordinate value of the 1st finger :					**Y_Position>>4**			
0xD003 1st finger's **X** coordinate value **X_Position&0x0F**					**1st** finger's **Y** coordinate value **Y_Position&0x0F**			
0xD004 1st finger pressure value								
0xD005 Report key flag (0x80)					Report the number of fingers			
0xD006 Fixed **0xAB**								
0xD007 2nd finger **ID**					**2nd finger status: pressed (0x06) or lifted**			
0xD008 The high eight bits of the **X** coordinate value of the 2nd finger :					**X_Position>>4**			

0xD009 The high eight bits of the **Y** coordinate value of the 2nd finger :	**Y_Position>>4**	
0xD00A 2nd finger's **X** coordinate value **X_Position&0x0F**	**2nd** finger's **Y** coordinate value **Y_Position&0x0F**	
0xD00B 2nd finger pressure value		
0xD00C 3rd finger **ID**	**3rd finger status: pressed (0x06) or lifted**	
0xD00D The high eight bits of the **X** coordinate value of the 3rd finger :	**X_Position>>4**	
0xD00E The high eight bits of the **Y** coordinate value of the 3rd finger :	**Y_Position>>4**	
0xD00F 3rd finger's **X** coordinate value **X_Position&0x0F**	**3rd** finger's **Y** coordinate value **Y_Position&0x0F**	
0xD010 3rd finger pressure value		
0xD011 4th finger **ID**	**4th finger status: pressed (0x06) or lifted**	
0xD012 The high eight bits of the **X** coordinate value of the 4th finger :	**X_Position>>4**	
0xD013 The high eight bits of the **Y** coordinate value of the 4th finger :	**Y_Position>>4**	
0xD014 4th finger's **X** coordinate value **X_Position&0x0F**	**4th** finger's **Y** coordinate value **Y_Position&0x0F**	
0xD015 4th finger pressure value		
0xD016 5th finger **ID**	**5th finger status: pressed (0x06) or lifted**	
0xD017 The high eight bits of the **X** coordinate value of the 5th finger :	**X_Position>>4**	
0xD018 The high eight bits of the **Y** coordinate value of the 5th finger :	**Y_Position>>4**	
0xD019 5th finger's **X** coordinate value **X_Position&0x0F**	The **Y** coordinate value of **the 5th** finger is **Y_Position&0x0F**	
0xD01A 5th finger pressure value		

Version information register (ENUM_MODE_DEBUG_INFO mode)

(1) Version information must be read in **debug info mode (write 0xD101)**

(2) Read the information of the corresponding register address

(3) Return to normal **mode (write 0xD109)**

For details, please refer to the reporting function **cst3xx_firmware_info .**

Register Address	Register Description	Register (4 bytes)			
0xD1F4	Number of **buttons, TX, and RX** channels	**KEY_NUM**	**TP_NRX**	**NC**	**TP_NTX**
0xD1F8	**X/Y** resolution	**TP_LOSS**		**TP_RESX**	
0xD1FC	**Firmware verification code, Bootloader** time	**0xCACA**		**BOOT_TIMER**	
0xD204	Chip type, firmware project **ID**	**IC_TYPE**		**PROJECT_ID**	
0xD208	Chip firmware version number	**FW_MAJOR**	**FW_MINOR**	**FW_BUILD**	
0xD20C	Chip firmware **checksum**	**checksum_H**	**checksum_H**	**checksum_L**	**checksum_L**

Mode Command Register

The mode command is used to enter different working modes, usually for internal debugging. The client usually uses the **normal** mode **0xD109.**

Order	Command Description	Format
0xD101	**ENUM_MODE_DEBUG_INFO** mode, enter the mode of reading firmware information.	**Write 0xD1 0x01**
0xD102	**System_Reset** flag, reset the chip.	**Write 0xD1 0x02**
0xD104	**Redo_Calibration** flag, reinitializes the algorithm.	**Write 0xD1 0x04**
0xD105	**Deep** sleep, enter sleep mode.	**Write 0xD1 0x05**
0xD108	**ENUM_MODE_DEBUG_POINTS, enter debug** point reporting mode.	**Write 0xD1 0x08**
0xD109	ENUM_MODE_NORMAL, enter the normal reporting mode, which is the default mode.	**Write 0xD1 0x09**
0xD10A	**ENUM_MODE_DEBUG_RAWDATA, enter the mode of reading rawdata .**	**Write 0xD1 0x0A**
0xD10B	**ENUM_MODE_DEBUG_WRITE, enter debug write** mode.	**Write 0xD1 0x0B**
0xD10C	**ENUM_MODE_DEBUG_CALIBRATION, enter redo** debugging mode.	**Write 0xD1 0x0C**
0xD10D	**ENUM_MODE_DEBUG_DIFF**	**Write 0xD1 0x0D**
0xD119	**ENUM_MODE_FACTORY**	**Write 0xD1 0x19**

11.2 Self-contained product CST8XX/CST7XX register

The working mode switching command is as follows

Working Mode	Switch Command	describe
NORMAL	0000	Normal reporting and gesture reporting
DBG_IDAC	0004	Factory test data acquisition
DBG_POS	00E0	Factory test buttons and coordinate acquisition
DBG_RAW	0006	Original value acquisition
DBG_SIG	0007	differ value acquisition

NOMAL Register Description

Address	Name	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0				illustrate	Access
00h	**Work_mode**												**Write:** **00: NORMAL** **04: DBG_IDAC** **E0: DBG_POS** **06: DBG_RAW** **07: DBG_SIG**	**R/W**

01h Proximity ID		[7:0]								defaultÿ00 far awayÿC0 nearÿE0	R
02h touch num						touch points[3:0]					R
03h touch1_XH		event_flg			X_position[11:8]						R
04h touch1_XL		X_position[7:0]									R
05h touch1_YH		touch_ID[3:0]			Y_position[11:8]						R
06h touch1_YL		Y_position[7:0]									R
07h										defaultÿ00	R
08h										defaultÿ00	R
9h touch2_XH		event_flg			X_position[11:8]						R
10h touch2_XL		X_position[7:0]									R
11h touch2_YH		touch_ID[3:0]			Y_position[11:8]						R
12h touch2_YL		Y_position[7:0]									R
13h										defaultÿ00	R
14h										defaultÿ00	R
…	…											
A5h sleep		deepsleep[7:0]								write 03 Enter deepsleep	IN
A6h fw_version		fw_version[7:0]								Firmware version number	R
A7h fw_version		fw_version[15:8]									R
A8h module_ID		module_version[7:0]								Module ID	R
A9h project_name		project_name[7:0]								defaultÿ00	R
AAh chip_type		chip_type[7:0]								Chip Model	R
ABh chip_type		chip_type[15:8]									R
ACh checksum		checksum[7:0]								Firmware checksum	R
ADh checksum		checksum[15:8]									R
…	…											
B0h Prox_state		Prox_state[7:0]								write 01H Enter Proximity mode 00H Exit Proximity mode	IN
…	…											
D0h ges_state		ges_state[7:0]								write 01H Enter gesture recognition model 00H exit gesture mode	IN
…	…											

D3h gesture ID		gesture[7:0]								Gesture mode is enabled to be effective double click:0x24 up:0x22 down:0x23 left:0x20 rain:0x21 C:0x34 e:0x33 m:0x32 O:0x30 S:0x46 V:0x54 ln:0x31 From:0x65	R
D4h	gesture data									Reserved for compatibility with other drivers	R
D5h											R
D6h											R
D7h											R
D8h											R
D9h											R
Yes											R