Touch Screen Porting Guide

Porting Guide

Customer Support

MT6000

Doc No: CS6000-AZ10A-PPD-V1.1EN
Version: V1.1
Release date: 2017-02-15
Classification: internal

Specifications are subject to change without notice.

Touch Screen

Porting Guide

MediaTek Inc.

Postal address

No. 1, Dusing 1st Rd. , Hsinchu Science
Park, Hsinchu City, Taiwan 30078

MTK support office address

No. 1, Dusing 1st Rd. , Hsinchu Science
Park, Hsinchu City, Taiwan 30078

Internet

http://www.mediatek.com/

Document Revision History

Revision	Date	Author	Description
V1.0	2017-02-15	Xj wang	Initial Release

Table of Contents

Lists of Tables

Lists of Figures

1 Introduction

1.1 Purpose

This document provides the porting guidelines for the touch screen modules.

1.2 Scope

The document provides the porting details of the touch screen.

1.3 Who Should Read This Document

This document is primarily intended for:

- Engineers with technical knowledge of the touch screen
- Customers who integrate the touch screen with user-defined applications

1.4 How to Use This Manual

This segment explains how information is distributed in this document, and presents some cues and examples to simplify finding and understanding information in this document. Table 1-1 presents an overview of the chapters and appendices in this document.

Table 1-1. Chapter Overview

#	Chapter	Contents
1	Introduction	Describes the scope and layout of this document.
2	References	The reference document of this document
3	Definition	The definition of this document
4	Abbreviations	The abbreviations if this document
5	Overview	Overview of touch screen driver
6	Touch screen	The primary structure and API of touch screen driver
7	Porting guide	Touch screen porting guide

1.4.1 Terms and Conventions

This document uses special terms and typographical conventions to help you easily identify various information types in this document. These cues are designed to simply finding and understanding the information this document contains.

Table 1-2. Conventions

Convention	Usage	Example
[1]	Serial number of a document in the order of appearance in the References topic	
void xx(zz)	Source code	int tpd_driver_add(struct tpd_driver_t *tpd_drv)
☞	Important	

2 References

[1] Linux device tree usage document:

https://android.googlesource.com/kernel/mediatek/+/android-4.4.4_r3/Documentation/devicetree/usage-model.txt

[2] Linux i2c operation:

https://android.googlesource.com/kernel/mediatek/+/android-4.4.4_r3/Documentation/i2c/writing-clients

[3] Linux regulator operation:

https://android.googlesource.com/kernel/mediatek/+/android-4.4.4_r3/Documentation/power/regulator/overview.txt

3 Definitions

For the purposes of the present document, the following terms and definitions apply:

$(platform): the platform name, example: mt6757.

$(project): the project name of you, example: evb6757_64_op01.

Device tree: Linux kernel manages the device model.

4 Abbreviations

Please note the abbreviations and their explanations provided in Table 4-1. They are used in many fundamental definitions and explanations in this document and are specific to the information that this document contains.

Table 4-1. Abbreviations

Abbreviations	Explanation
MTK	MediaTek, Asia’s largest fabless IC design company.
dts	Device tree
EINT	External interrupt
SOP	Standard Operating Procedure

5 Overview

This chapter first gives a brief description of the modules of the system and the relationship of the modules.

Mediatek implement a common driver for manage specific touch driver.

The common driver file is :

drivers/input/touchscreen/mediatek/mtk_tpd.c

it is controlled by kconfig micro : CONFIG_TOUCHSCREEN_MTK

CONFIG_TOUCHSCREEN_MTK=y

means to use common driver.

This porting guide is based on used common driver.

5.1 Architecture

Touch screen total architecture

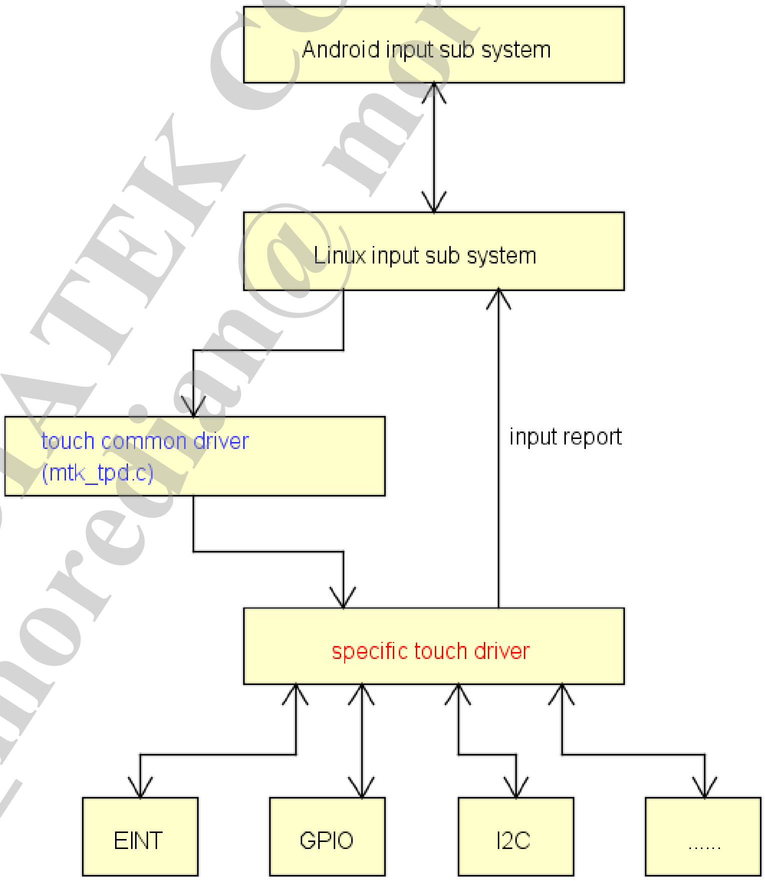
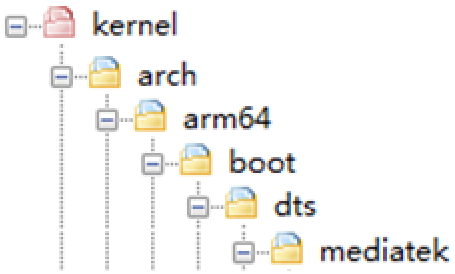


Figure 5-1. Touch screen driver architecture

5.2 Source Code Organization

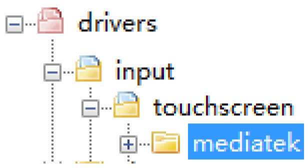
Mediatek touch screen customization parameters file folder ($(project).dts).



```
kernel
  arch
    arm64
      boot
        dts
          mediatek
```

Figure 5-2. Touch screen driver customization folder

Mediatek touch screen kernel layer code



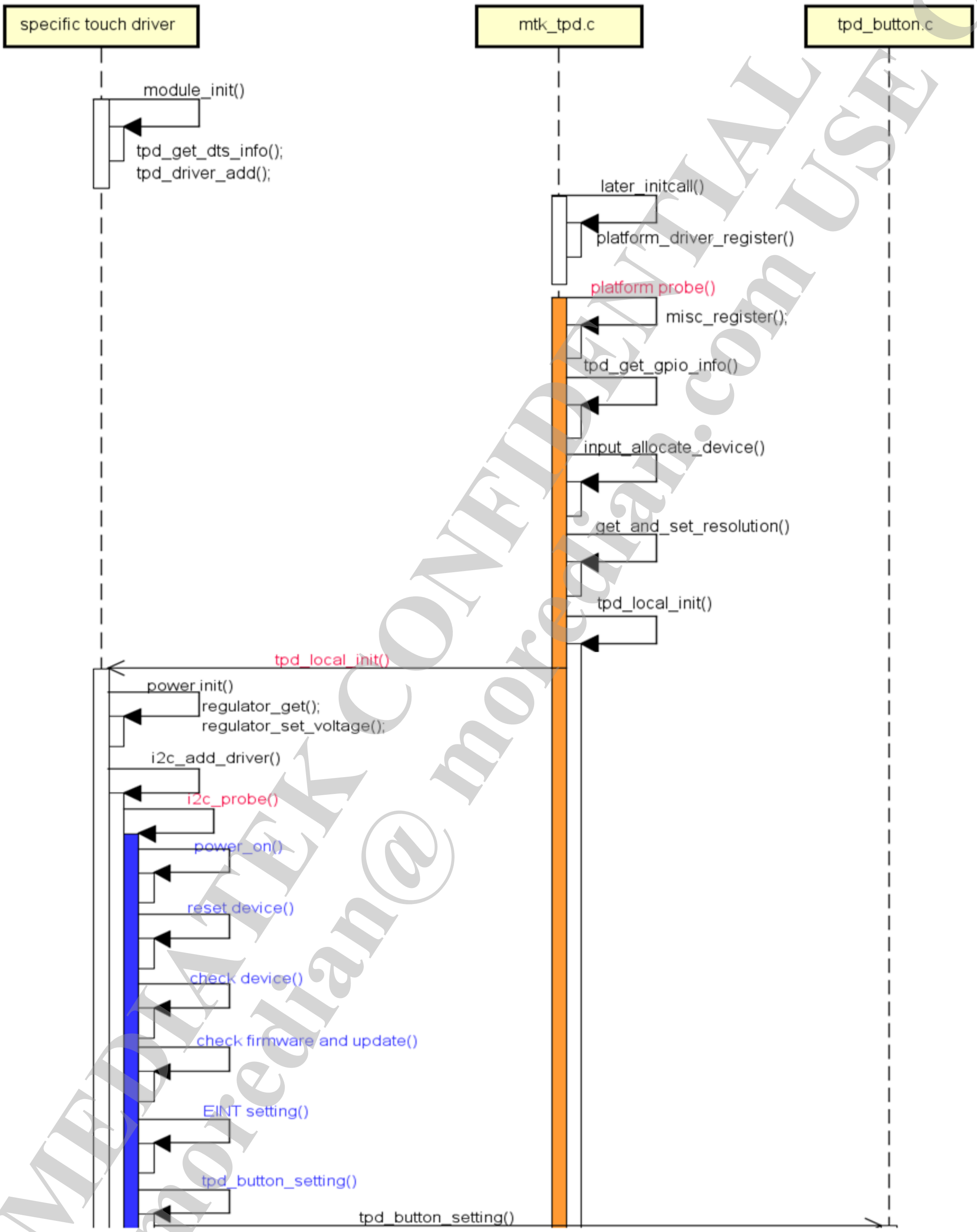
```
drivers
  input
    touchscreen
      mediatek
```

Figure 5-3. Source Code Directory Structure of driver

The description of the directories and their subdirectories is given below:

arch/arm64/boot/dts/mediatek	Contains the touch screen customization setting
Drivers/input/touchscreen/mediatek	Contains the touch screen driver code

5.3 Touch screen initialization flow


specific touch driver
mtk_tpd.c
tpd_button.c
module_init()
tpd_get_dts_info();
tpd_driver_add();
later_initcall()
platform_driver_register()
platform probe()
misc_register();
tpd_get_gpio_info()
input_allocate_device()
get_and_set_resolution()
tpd_local_init()
tpd_local_init()
power init()
regulator_get();
regulator_set_voltage();
i2c_add_driver()
i2c_probe()
power_on()
reset device()
check device()
check firmware and update()
EINT setting()
tpd_button_setting()
tpd_button_setting()

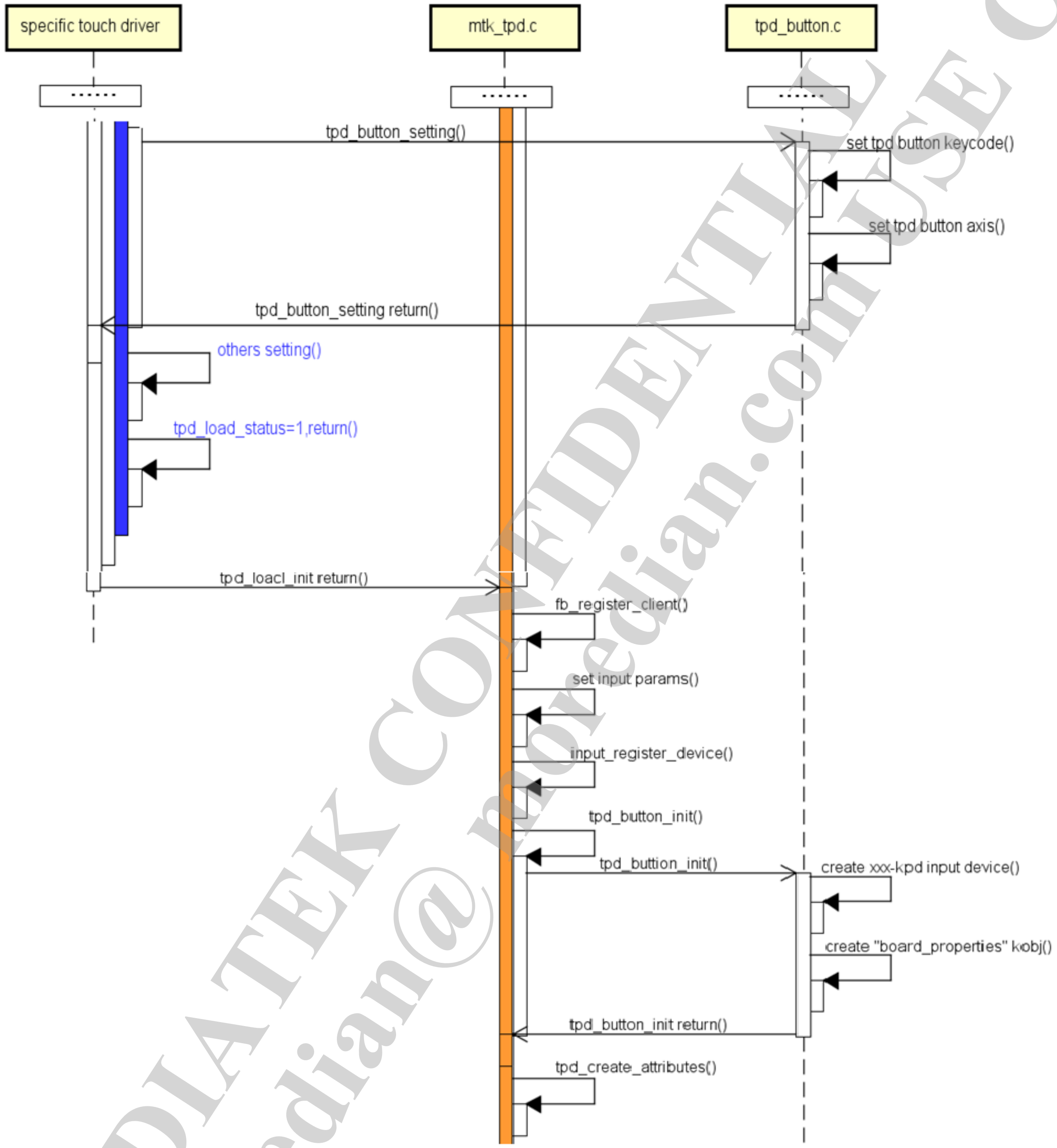


Figure 5-4. Touch screen initialization flow.

6 Touch screen

Touch screen’s primary structures and APIs are defined in file: drivers/input/touchscreen/mediatek/tpd.h

6.1 Data Structure

6.1.1 struct tpd_driver_t

The tpd_driver_t structure is used to add driver to mtk_tpd. It has the following header-file definition:

```
struct tpd_driver_t {
    char *tpd_device_name;
    int (*tpd_local_init)(void);
    void (*suspend)(struct device *h);
    void (*resume)(struct device *h);/* specific touch driver resume call back function*/
    int tpd_have_button; /* does specific touch driver support button*/
    struct tpd_attrs attrs; /* specific touch driver attributes*/
};
```

The **tpd_device_name** field is the specific touch driver name.

The **tpd_local_init** field is the specific touch driver initialization call back function.

The **suspend** field is the specific touch driver suspend call back function.

The **resume** field is the specific touch driver resume call back function.

The **tpd_have_button** field is whether specific touch driver support button.

The **attrs** field is the specific touch driver attributes.

6.1.2 struct tpd_dts_info

The tpd_dts_info structure is used to save the touch parameters from device tree. It has the following header-file definition:

```
struct tpd_dts_info {
    int tpd_resolution[2];
    int touch_max_num;
    int use_tpd_button;
    int tpd_key_num;
```

```
        int tpd_key_local[4];

        struct tpd_key_dim_local tpd_key_dim_local[4];

        struct tpd_filter_t touch_filter;

    };
```

The **tpd_resolution** field is the x, y resolution.

The **touch_max_num** field is the maximum finger number of touch.

The **use_tpd_button** field is whether touch use button or not.

The **tpd_key_num** field is the number of touch button.

The **tpd_key_local** field is the key code of touch button.

The **tpd_key_dim_local** field is the button position information.

The **touch_filter** field is the parameters of touch filter.

6.1.3 struct tpd_key_dim_local

The tpd_key_dim_local structure is used to save the touch button position information. It has the following header-file definition:

```
    struct tpd_key_dim_local {

        int key_x;   /* button center x position*/

        int key_y;   /* button center y position*/

        int key_width;          /* button width*/

        int key_height;         /* button height*/

    };
```

The **key_x** field is the button center x position.

The **key_y** field is the button center y position.

The **key_width** field is the button width.

The **key_height** field is the button height.

6.1.4 struct tpd_device

The tpd_device structure is used to manager touch driver. It has the following header-file definition:

```
    struct tpd_device {

        struct device *tpd_dev;             /* touch platform device*/

        struct regulator *reg;  /* touch regulator 1(power VDD)*/

        struct regulator *io_reg;           /* touch regulator 2(power IO)*/
        struct input_dev *dev;          /* position input device*/
        struct input_dev *kpd;          /* button input device*/
        struct timer_list timer;        /* not use*/
        struct tasklet_struct tasklet;  /* not use*/
        int btn_state;          /* button status*/
    };
```

The **tpd_dev** field is the touch platform device.

The **reg** field is the touch regulator 1(power VDD).

The **io_reg** field is the touch regulator 2(power IO). It is reserved.

The **dev** field is the position input device.

The **kpd** field is the button input device.

The **timer** field is reserved.

The **tasklet** field is reserved.

The **btn_state** field is the button status.

6.2 Variables

6.2.1 tpd_driver_list

tpd_driver_list is an array variable. It is used to save the specific driver that will add to mtk_tpd. It is defined as below:

static struct tpd_driver_t **tpd_driver_list**[TP_DRV_MAX_COUNT];

6.2.2 tpd_dts_data

tpd_dts_data is used to save the customization parameters of specific driver that from device tree. It is defined as below:

struct tpd_dts_info **tpd_dts_data**；

6.2.3 tpd

tpd is a point variable. It is used to save the touch screen total information of driver. It is defined as below:

struct tpd_device ***tpd**;

6.3 Functions

These functions are specific touch driver will use.

6.3.1 tpd_driver_add

The main purpose of the tpd_driver_add is adding specific driver information to global variable tpd_driver_list.

- Definition

  int tpd_driver_add(struct tpd_driver_t *tpd_drv);

- Parameters

Table 6-1. Parameters of tpd_driver_add

Parameters	Direction (IN/OUT)	Description
tpd_drv	IN	Specific touch driver information

- Return Values

  On success, it returns a 0 that is a specific driver has added into global variable tpd_driver_list successfully. On errors, the return value is as below table.

Table 6-2. Return Values of tpd_driver_add

Return Values	Description
0	On success, that is a specific driver has added into global variable tpd_driver_list successfully.
-1	On error, that is a specific driver has added into global variable tpd_driver_list unsuccessfully.
1	On error, the driver information has existed in tod_driver_list.

6.3.2 tpd_get_dts_info

The main purpose of the tpd_get_dts_info is used to get parameters from device tree and save to global variable : tpd_dts_data

- Definition

  void tpd_get_dts_info(void);

- Parameters

Table 6-3. Parameters of tpd_get_dts_info

Parameters	Direction (IN/OUT)	Description
N/A	N/A	N/A

- Return Values

No return value.

Table 6-4. Return Values of tpd_get_dts_info

Return Values	Description
N/A	N/A

6.3.3 tpd_gpio_as_int

The main purpose of the tpd_gpio_as_int is used to set eint pin as EINT mode.

- Definition

void tpd_gpio_as_int(int pin);

- Parameters

Table 6-5. Parameters of tpd_gpio_as_int

Parameters	Direction (IN/OUT)	Description
pin	IN	It must be set as 1

- Return Values

No return value.

Table 6-6. Return Values of tpd_gpio_as_int

Return Values	Description
N/A	N/A

6.3.4 tpd_gpio_output

The main purpose of the tpd_gpio_output is used to set eint pin or reset pin output value.

- Definition

void tpd_gpio_output(int pin, int level);

- Parameters

Table 6-7. Parameters of tpd_gpio_output

Parameters	Direction (IN/OUT)	Description
pin	IN	PIN type, 1: EINT pin; 0: reset pin;
level	IN	Output value, 1: output high; 0: output low

- Return Values

No return value.

Table 6-8. Return Values of tpd_gpio_output

Return Values	Description

Return Values	Description
N/A	N/A

7 Porting guide

7.1 Parameters setting

Touch parameters are set in device tree file :

kernel-4.4/arch/arm64/boot/dts/mediatek/$(project).dts.

For example: evb6757_64.dts

```
&touch {
        tpd-resolution = <1080 1920>;    /* resolution*/
        use-tpd-button = <0>;
        tpd-key-num = <3>;
        tpd-key-local= <139 172 158 0>;  /* button key code*/
        tpd-key-dim-local = <90 883 100 40 230 883 100 40 370 883 100 40 0 0 0 0>; /*button position*/
        tpd-max-touch-num = <5>;
        tpd-filter-enable = <1>;
        tpd-filter-pixel-density = <146>;
        tpd-filter-custom-prameters = <0 0 0 0 0 0 0 0 0 0 0 0>;
        tpd-filter-custom-speed = <0 0 0>;
        status = "okay";
};
```

The description of parameters is as following table:

Table 7-1. Parameters of touch driver

Parameters	Description
tpd-resolution	Touch resolution
use-tpd-button	Whether touch have button
tpd-key-num	Touch button number
tpd-key-local	Touch button Linux key code
tpd-key-dim-local	Touch button position information.
tpd-max-touch-num	The maximum finger count of touch
tpd-filter-enable	Whether use touch filter feature
tpd-filter-pixel-density	Touch filter parameter
tpd-filter-custom-prameters	Touch filter parameter
tpd-filter-custom-speed	Touch filter parameter

7.2 GPIO control

Touch driver use Linux pinctrl method to control GPIO.

It is configured in file:

kernel-4.4/arch/arm64/boot/dts/mediatek/$(project).dts.

For example: evb6757_64.dts

```
&touch {
        pinctrl-names = "default", "state_eint_as_int", "state_eint_output0", "state_eint_output1",
                "state_rst_output0", "state_rst_output1";
        pinctrl-0 = <&ctp_pins_default>;
        pinctrl-1 = <&ctp_pins_eint_as_int>;
        pinctrl-2 = <&ctp_pins_eint_output0>;
        pinctrl-3 = <&ctp_pins_eint_output1>;
        pinctrl-4 = <&ctp_pins_rst_output0>;
        pinctrl-5 = <&ctp_pins_rst_output1>;
        status = "okay";
};
&pio {
        ctp_pins_default: eint0default {
        };
        ctp_pins_eint_as_int: eint@0 {
                pins_cmd_dat {
                        pins = <PINMUX_GPIO1__FUNC_GPIO1>;
                        slew-rate = <0>;
                        bias-disable;
                };
        };
        ctp_pins_eint_output0: eintoutput0 {
                pins_cmd_dat {
                        pins = <PINMUX_GPIO1__FUNC_GPIO1>;
                        slew-rate = <1>;
                        output-low;
                };
        };
        ctp_pins_eint_output1: eintoutput1 {
                pins_cmd_dat {
                        pins = <PINMUX_GPIO1__FUNC_GPIO1>;
                        slew-rate = <1>;
                        output-high;
                };
        };
        ctp_pins_rst_output0: rstoutput0 {
                pins_cmd_dat {
                        pins = <PINMUX_GPIO10__FUNC_GPIO10>;
                        slew-rate = <1>;
                        output-low;
                };
        };
        ctp_pins_rst_output1: rstoutput1 {
                pins_cmd_dat {
                        pins = <PINMUX_GPIO10__FUNC_GPIO10>;
                        slew-rate = <1>;
                        output-high;
                };
        };
};
```

7.3 I2C setting

Method one: I2c is configured in dws file by DCT.

The DCT path is :

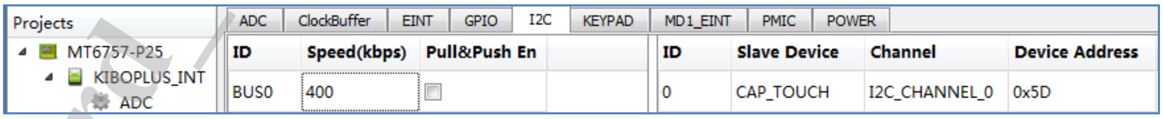
/vendor/mediatek/proprietary/scripts/dct/DrvGen.exe

The dws file path is :

drivers/misc/mediatek/dws/mt6757/$(project).dws

Example: evb6757_64.dws

We can set touch I2C information in I2C part of dws file.



ID	Speed(kbps)	Pull&Push En	ID	Slave Device	Channel	Device Address
BUS0	400	□	0	CAP_TOUCH	I2C_CHANNEL_0	0x5D

Figure 7-1. I2C setting in dws file.

Table 7-2. Parameters of I2C setting in dws file

Parameters	Description
Speed	Set the I2C bus speed
Pull&Push En	Whether this I2C bus use pull&push mode
ID	The device configuration number
Slave Device	The slave device name
Channel	I2C bus number that device connected
Device Address	I2C address of slave device

CAP_TOUCH is used to set touch I2C.

The compatible of i2c driver in specific touch driver must is “mediatek,cap_touch”.

For example:

```
static const struct of_device_id tpd_of_match[] = {
        {.compatible = "mediatek,cap_touch"},
        {},
};
static struct i2c_driver tpd_i2c_driver = {
        .driver = {
                .of_match_table = tpd_of_match,
//...
```

Method two: I2C is configured in device tree.

The device tree file is:

kernel-4.4/arch/arm64/boot/dts/mediatek/$(project).dts.

For example: evb6757_64.dts

```
&i2c0 {
        #address-cells = <1>;
        #size-cells = <0>;
        clock-frequency = <400000>;
        mediatek,use-open-drain;
        cap_touch@5d {
                compatible = "mediatek,cap_touch";
                reg = <0x5d>;
                status = "okay";
        };
};
```

7.4 Regulator setting

Method one: regulator is configured in dws file by DCT.

The DCT path is :

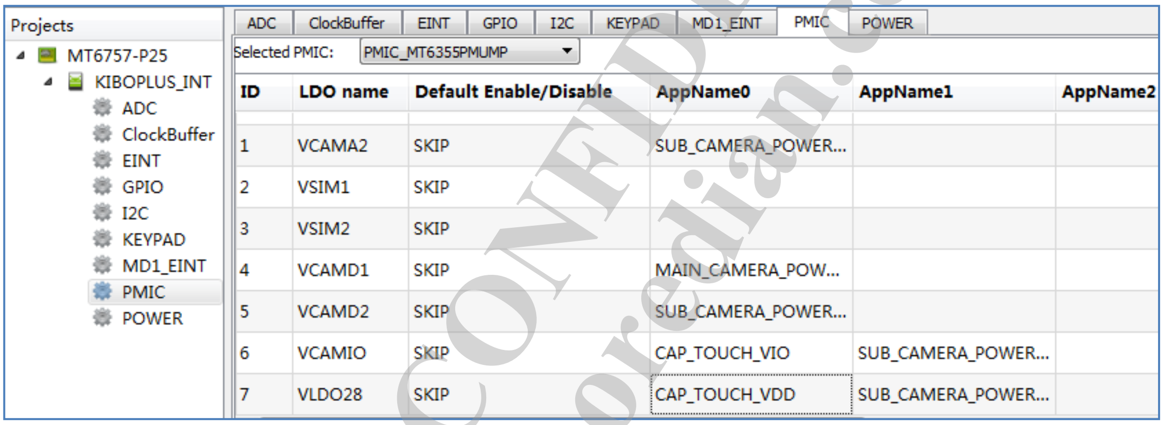
/vendor/mediatek/proprietary/scripts/dct/DrvGen.exe

The dws file path is :

drivers/misc/mediatek/dws/mt6757/$(project).dws

Example: evb6757_64.dws

We can set touch power information in PMIC part of dws file.



Selected PMIC: PMIC_MT6355PMUMP

ID	LDO name	Default Enable/Disable	AppName0	AppName1	AppName2
1	VCAMA2	SKIP	SUB_CAMERA_POWER...		
2	VSIM1	SKIP			
3	VSIM2	SKIP			
4	VCAMD1	SKIP	MAIN_CAMERA_POW...		
5	VCAMD2	SKIP	SUB_CAMERA_POWER...		
6	VCAMIO	SKIP	CAP_TOUCH_VIO	SUB_CAMERA_POWER...	
7	VLDO28	SKIP	CAP_TOUCH_VDD	SUB_CAMERA_POWER...	

Figure 7-2. regulator setting in dws file.

Table 7-3. Parameters of regulator setting in dws file

Parameters	Description
LDO name	LDO name of PMIC , can't setting
Default Enable/Disable	LDO default status: enabled, disabled or skip
AppName0,1,2...	Device name that will use this LDO

CAP_TOUCH_VIO is used to set touch VIO power.

CAP_TOUCH_VDD is used to set touch VDD power.

Method two: regulator is configured in device tree.

The device tree file is:

kernel-4.4/arch/arm64/boot/dts/mediatek/$(project).dts.

For example: evb6757_64.dts

```
mt_pmic_vldo28_ldo_reg: ldo_vldo28 {
        regulator-name = "vldo28";
        regulator-min-microvolt = <2800000>;
        regulator-max-microvolt = <2800000>;
```

```
    regulator-enable-ramp-delay = <264>;
    regulator-default-on = <1>; /* 0:skip, 1: off, 2:on */
    status = "okay";
};
&touch {
    vtouch-supply = <&mt_pmic_vldo28_ldo_reg>;
    status = "okay";
};
```

There are three steps to use regulator in driver.

Step 1: regulator get;

Step 2: regulator set voltage;

Step 3: regulator enable.

For example:

```
/* step1: regulator get*/
tpd->reg = regulator_get(tpd->tpd_dev, "vtouch");
if (IS_ERR_OR_NULL(tpd->reg)) {
    pr_err("%s get regulator failed!\n", __func__);
    return -EFAULT;
}
/* step2: set voltage*/
ret = regulator_set_voltage(tpd->reg, 2800000, 2800000); /*set 2.8v*/
if (ret) {
    pr_err("%s regulator_set_voltage(%d) failed!\n", __func__, ret);
    return -EFAULT;
}
/* step3: regulator enable*/
ret = regulator_enable(tpd->reg);/*enable regulator*/
if (ret) {
    pr_err("%s regulator_enable() failed!\n", __func__);
    return -EFAULT;
}
```

Notice :

1. The regulator id must “vtouch” when use API: regulator_get() if touch VDD is configured by DCT .

2. Regulator control is not necessary. It is according to the project design.

7.5 EINT setting

Method one: EINT is configured in dws file by DCT.

The DCT path is :

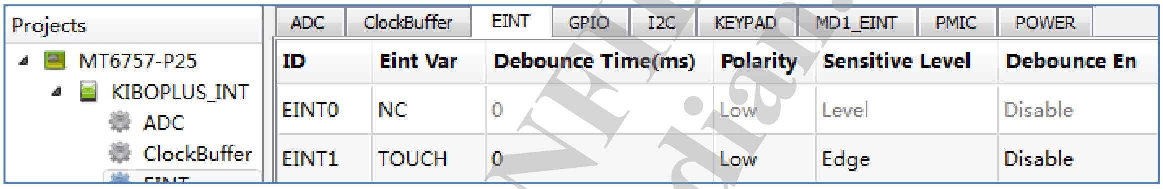
/vendor/mediatek/proprietary/scripts/dct/DrvGen.exe

The dws file path is :

drivers/misc/mediatek/dws/mt6757/$(project).dws

Example: evb6757_64.dws

We can set touch power information in EINT part of dws file.



ID	Eint Var	Debounce Time(ms)	Polarity	Sensitive Level	Debounce En
EINT0	NC	0	Low	Level	Disable
EINT1	TOUCH	0	Low	Edge	Disable

Figure 7-3. EINT setting in dws file.

Table 7-4. Parameters of EINT setting in dws file

Parameters	Description
ID	Eint number of chip, can't set.
Eint variable	Device name that will use this eint.
Debounce Time	Debounce time of this eint , its unit is ms. It must be set 0 when Debounce En is Disable.
Polarity	The eint trigger polarity that you want to set. High or Low.
Sensitive Level	The eint trigger sensitive that you want to set. Edge or Level.
Debounce En	Whether enable the eint debounce function.

TOUCH is used to set EINT of touch.

Method two: EINT is configured in device tree.

The device tree file is:

kernel-4.4/arch/arm64/boot/dts/mediatek/$(project).dts.

For example: evb6757_64.dts

```
&touch {
        interrupt-parent = <&eintc>;
        interrupts = <1 IRQ_TYPE_EDGE_FALLING>;
        debounce = <1 0>;
        status = "okay";
};
```

There are two steps to use EINT in driver.

Step 1: get irq number;

Step 2: request irq;

For example:

```
static unsigned int touch_irq;

static int tpd_irq_registration(void)

{

        struct device_node *node = NULL;

        int ret = 0;

        u32 ints[2] = {0, 0};


        if (!tpd || !(tpd->tpd_dev) || !(tpd->tpd_dev->of_node))

                return -EINVAL;


        node = tpd->tpd_dev->of_node;

        of_property_read_u32_array(node, "debounce", ints,

                ARRAY_SIZE(ints));

        gpio_set_debounce(ints[0], ints[1]);

        /* get irq number*/

        touch_irq = irq_of_parse_and_map(node, 0);

        if (touch_irq == 0)

                return -EINVAL;

        /* request irq*/

        ret = request_threaded_irq(touch_irq, NULL,

                 (irq_handler_t)tpd_eint_interrupt_handler,

                IRQF_TRIGGER_NONE, "TOUCH_PANEL-eint", NULL);

        return ret;

}
```

7.6 Porting SOP

Step 1:

Put touch driver folder into path:

drivers/input/touchscreen/mediatek/

For example: GT1151

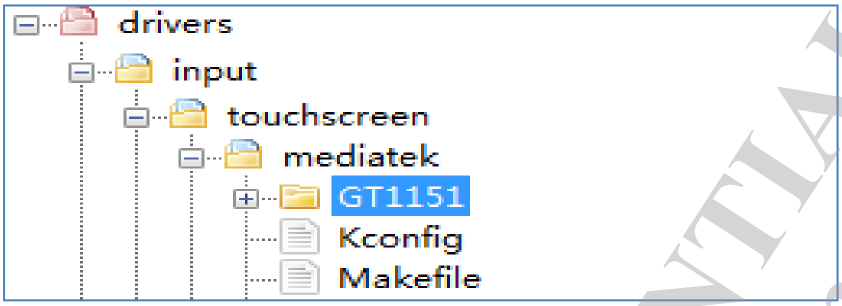


Figure 7-4. GT1151 driver folder path.

Step 2:

Add Kconfig and build information in file :

drivers/input/touchscreen/mediatek/Kconfig

and

drivers/input/touchscreen/mediatek/Makefile

For example: GT1151

Kconfig file:

```
source "drivers/input/touchscreen/mediatek/GT1151/Kconfig"
```

Makefile file :

```
obj-$(CONFIG_TOUCHSCREEN_MTK_GT1151) +=  GT1151/
```

Step 3:

Create Kconfig and Makefile file in touch driver folder.

For example: GT1151

Kconfig file: drivers/input/touchscreen/mediatek/GT1151/Kconfig

```
config TOUCHSCREEN_MTK_GT1151
        bool "GT1151 for Mediatek package"
        default n
        help
                Say Y here if you have GT1151 touch panel.
                If unsure, say N.
```

Makefile file: drivers/input/touchscreen/mediatek/GT1151/Makefile

```
ccflags-y += -I$(srctree)/drivers/input/touchscreen/mediatek/GT1151/
ccflags-y += -I$(srctree)/drivers/input/touchscreen/mediatek/
```

```
                ccflags-y += -I$(srctree)/drivers/misc/mediatek/include/mt-plat/

                ccflags-y += -I$(srctree)/drivers/misc/mediatek/include/mt-plat/$(MTK_PLATFORM)/include/


                obj-y        +=  gt1x_tpd.o
```

Step 4:

Choose the touch driver for compile in file:

arch/arm64/configs/$(project)_debug_defconfig

and

arch/arm64/configs/$(project)_defconfig

For example: evb6757_64

arch/arm64/configs/evb6757_64_debug_defconfig

arch/arm64/configs/evb6757_64_defconfig

```
                CONFIG_TOUCHSCREEN_MTK_GT1151=y
```

Step 5:

Configure parameters, GPIO, I2C, regulator and EINT in dts and dws file:

arch/arm64/boot/dts/mediatek/$(project).dts

and

drivers/misc/mediatek/dws/mt6757/$(project).dws

For example: evb6757_64

arch/arm64/boot/dts/mediatek/evb6757_64.dts

drivers/misc/mediatek/dws/mt6757/evb6757_64.dws

Step 6:

Edit touch driver code file and debug.

8 Examples

The GT1151 is the sample driver for reference. Its path is:

kernel-4.4/drivers/input/touchscreen/mediatek/GT1151/