

MEDIATEK

everyday genius

Touch Screen Porting Guide

Version: 1.0
Release date: 2019-03-21

© 2015 - 2019 MediaTek Inc.

This document contains information that is proprietary to MediaTek Inc. ("MediaTek") and/or its licensor(s). MediaTek cannot grant you permission for any material that is owned by third parties. You may only use or reproduce this document if you have agreed to and been bound by the applicable license agreement with MediaTek ("License Agreement") and been granted explicit permission within the License Agreement ("Permitted User"). If you are not a Permitted User, please cease any access or use of this document immediately. Any unauthorized use, reproduction or disclosure of this document in whole or in part is strictly prohibited. THIS DOCUMENT IS PROVIDED ON AN "AS-IS" BASIS ONLY. MEDIATEK EXPRESSLY DISCLAIMS ANY AND ALL WARRANTIES OF ANY KIND AND SHALL IN NO EVENT BE LIABLE FOR ANY CLAIMS RELATING TO OR ARISING OUT OF THIS DOCUMENT OR ANY USE OR INABILITY TO USE THEREOF. Specifications contained herein are subject to change without notice.

Specifications are subject to change without notice.

Document Revision History

Revision	Date	Description
1.0	2019-3-14	Initial Draft

Table of Contents

Document Revision History.....	3
Table of Contents.....	4
1 Introduction.....	5
1.1 Purpose	5
1.2 Definitions, Acronyms and Abbreviations	5
1.3 References	5
1.4 Overview	5
2 Specific Contents	6
2.1 CTP HW and Platform Instruction	6
2.1.1 CTP HW interface	6
2.1.2 Platform instruction.....	6
2.1.3 I2C interface-Read with FIFO mode	7
2.1.4 I2C interface—read with DMA WRRD mode	8
2.1.5 I2C interface—read only with DMA mode	9
2.2 Mediatek touch driver introduction	9
2.2.1 Touch Panel Driver Architecture	10
2.2.2 driver file.....	10
2.2.3 APIs used in touch	11
2.2.4 earlyly suspend	11
2.3 touch customization.....	12
2.3.1 project configuration	12
2.3.2 Kernel config	12
2.4 touch customization.....	14
2.4.1 DWS file- I2C.....	15
2.4.2 DWS file - CTP RST/EINT Pin	15
2.4.3 dts file	15
2.4.4 device tree node	16
2.4.5 dts node--Parameters Introduction.....	17
2.4.6 device tree node	17
2.4.7 device tree node—I2C & EINT	18
2.5 Add a New Touch Driver	19
2.5.1 Touch Screen – Add New Driver (1/5)	19
2.5.2 Touch Screen – Add New Driver (2/5)	20
2.5.3 Touch Screen – Add New Driver (3/5)	20
2.5.4 Touch Screen – Add New Driver (4/5)	21
2.5.5 Touch Screen – Add New Driver (5/5)	21
2.6 Touch Panel Driver Debug	22
2.6.1 Touch Panel Driver Debug.....	22

1 Introduction

In this document, we'll introduce MT8183 touch feature and customized configuration.

1.1 Purpose

The document is for customer to configure and use Audio in MT8183 Project.

1.2 Definitions, Acronyms and Abbreviations

\$(proj)

Take MTK's turnkey solution as an example, \$(proj) correspond to tb8183m1_64_bsp.

1.3 References

N/A

1.4 Overview

N/A

2 Specific Contents

2.1 CTP HW and Platform Instruction

2.1.1 CTP HW interface

e.g. GT968 HW pad:

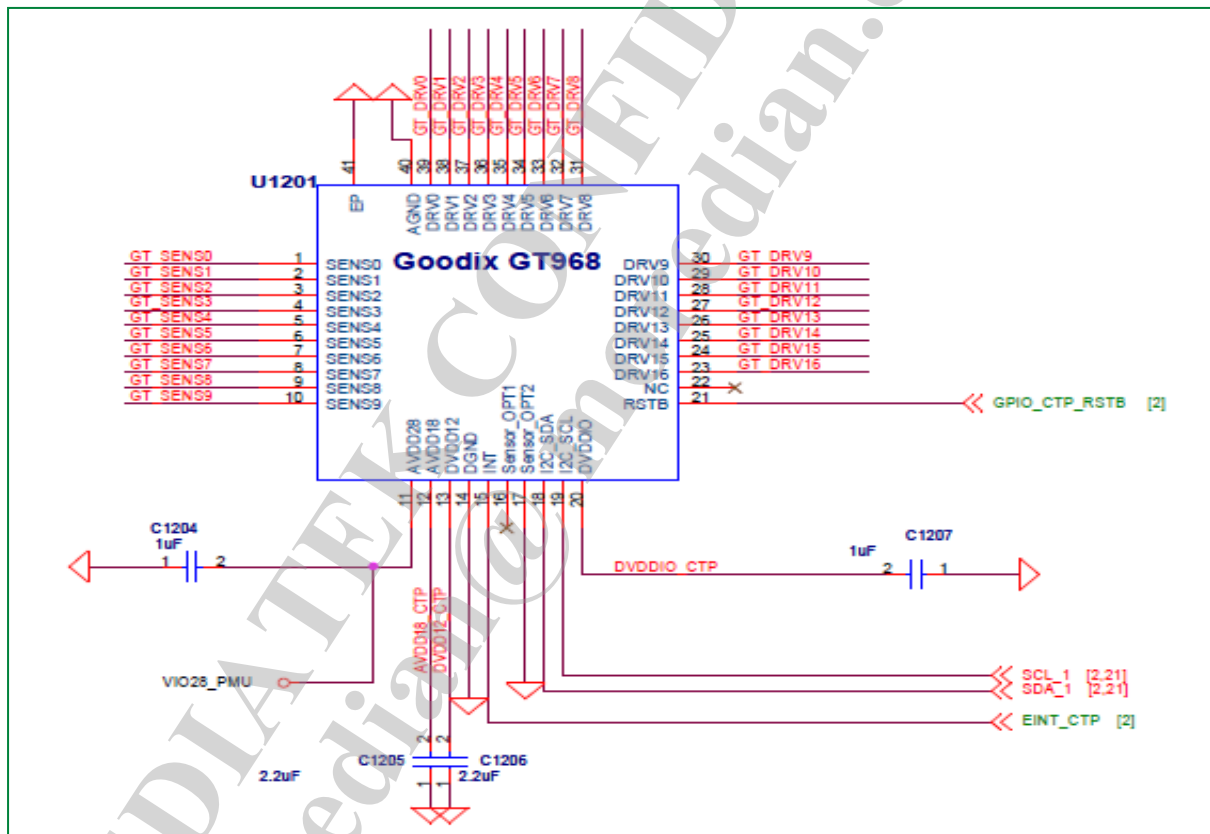
I2C

EINT

RST

AVDD/VDDIO

GND



2.1.2 Platform instruction

- GPIO /EINT/POWER
 - After got the HW information, you should apply your hardware interface to device tree(".dts"), such as EINT/RST/Power ... etc.
 - I2C
 - Mediatek platform I2C support :
 - FIFO mode : read/write 8 Bytes one time
 - DMA mode : only read/write: 65532Byte;
- write and read: write 255 Byte, read 31Byte

NOTE: For DMA 255x255Byte: The low 8-bit is "trans_len".
The high 8-bit is "trans_num"

```
trans_len = (msg->len) & 0xFF;
trans_num = (msg->len >> 8) & 0xFF;
```

2.1.3 I2C interface-Read with FIFO mode

- read with FIFO mode, max 8 bytes one time
 - write and read mode: I2C_WR_FLAG
 - without stop condition after register address write: I2C_RS_FLAG

Multiple Read



```
static s32 i2c_read_nondma(struct i2c_client *client,
    u8 addr, u8 *rxbuf, int len)
{
    int ret;
    struct i2c_msg msg;

    memset(&msg, 0, sizeof(struct i2c_msg));
    rxbuf[0] = addr;
    msg.addr = client->addr & I2C_MASK_FLAG;
    msg.flags = 0;
    msg.len = ((len & 0x1f) << 8) | 1;
    msg.buf = rxbuf;
    msg.ext_flag = I2C_WR_FLAG | I2C_RS_FLAG;
    msg.timing = GSLTP_I2C_MASTER_CLOCK;
    ret = i2c_transfer(client->adapter, &msg, 1);
    return ret;
}
```

if stop condition
need after register
addr, please
remove
I2C_RS_FLAG



- write with FIFO mode
 - max write 8 bytes one time

```

#define RPR_FIFO_MAX_RD_SIZE C_I2C_FIFO_SIZE
#define RPR_FIFO_MAX_WR_SIZE C_I2C_FIFO_SIZE - RPR_REG_ADDR_LEN

static s32 i2c_write_nondma(struct i2c_client *client, u8 addr, u8 *txbuf, int len)
{
    int ret;
    int retry = 0;
    struct i2c_msg msg;
    u8 wrBuf[RPR_FIFO_MAX_WR_SIZE + 1];

    if ((txbuf == NULL) && len > 0)
        return -1;

    memset(&msg, 0, sizeof(struct i2c_msg));
    memset(wrBuf, 0, RPR_FIFO_MAX_WR_SIZE + 1);
    wrBuf[0] = addr;
    if (txbuf)
        memcpy(wrBuf + 1, txbuf, len);

    msg.flags = 0;
    msg.buf = wrBuf;
    msg.len = 1 + len;
    msg.addr = (client->addr & I2C_MASK_FLAG);
    msg.ext_flag = (client->ext_flag | I2C_ENEXT_FLAG);
    msg.timing = RPR_I2C_MASTER_CLOCK;

    for (retry = 0; retry < 5; ++retry) {
        ret = i2c_transfer(client->adapter, &msg, 1);
        if (ret < 0)
            continue;
        return 0;
    }
    RPR0521_ERR("Dma I2C Write Error: 0x%04X, %d bytes, err-code: %d\n", addr, len, ret);
    return ret;
}
// end i2c write nondma

```

Parameter of i2c_write_nond

2.1.4 I2C interface—read with DMA WRRD mode

- read with DMA mode: I2C_DMA_FLAG
 - with write and read mode: I2C_WR_FLAG, max 31bytes
 - without stop condition after register address write: I2C_RS_FLAG

```

static s32 i2c_dma_read(struct i2c_client *client, u8 addr, u8
{
    int ret;
    struct i2c_msg msg;

    memset(&msg, 0, sizeof(struct i2c_msg));
    *g_dma_buff_va = addr;
    msg.addr = client->addr & I2C_MASK_FLAG;
    msg.flags = 0;
    msg.len = ((len & 0x1f) << 8) | 1;
    msg.buf = g_dma_buff_pa;
    msg.ext_flag = I2C_WR_FLAG | I2C_RS_FLAG | I2C_DMA_FLAG;
    msg.timing = GSLTP_I2C_MASTER_CLOCK;

    ret = i2c_transfer(client->adapter, &msg, 1);
    memcpy(rxbuf, g_dma_buff_va, len);
    return ret;
}

```


2.1.5 I2C interface—read only with DMA mode

- read only with DMA mode: have I2C_DMA_FLAG but no I2C_WR_FLAG
- there will be a stop condition between msg[0] and msg[1]
- max read length is 65532 bytes

```
static s32 i2c_dma_non_wrrd_read(struct i2c_client *client, u1
{
    int ret;
    struct i2c_msg msg[2];

    memset(&msg, 0, 2 * sizeof(struct i2c_msg));
    msg[0].addr = client->addr & I2C_MASK_FLAG;
    msg[0].flags = 0;
    msg[0].len = GSLTP_REG_ADDR_LEN;
    msg[0].buf = &addr;
    msg[0].ext_flag = I2C_DMA_FLAG;
    msg[0].timing = GSLTP_I2C_MASTER_CLOCK;

    msg[1].addr = client->addr & I2C_MASK_FLAG;
    msg[1].flags = 0;
    msg[1].len = len;
    msg[1].buf = g_dma_buff_pa;
    msg[1].ext_flag = I2C_DMA_FLAG;
    msg[1].timing = GSLTP_I2C_MASTER_CLOCK;

    ret = i2c_transfer(client->adapter, &msg, 2);
    memcpy(rxbuf, g_dma_buff_va, len);
    return ret;
} ? end i2c_dma_non_wrrd_read ?
```

- write with DMA mode, max 65532 bytes

```
static s32 i2c_dma_write(struct i2c_client *client, u8 addr, u8
{
    int ret;
    struct i2c_msg msg;

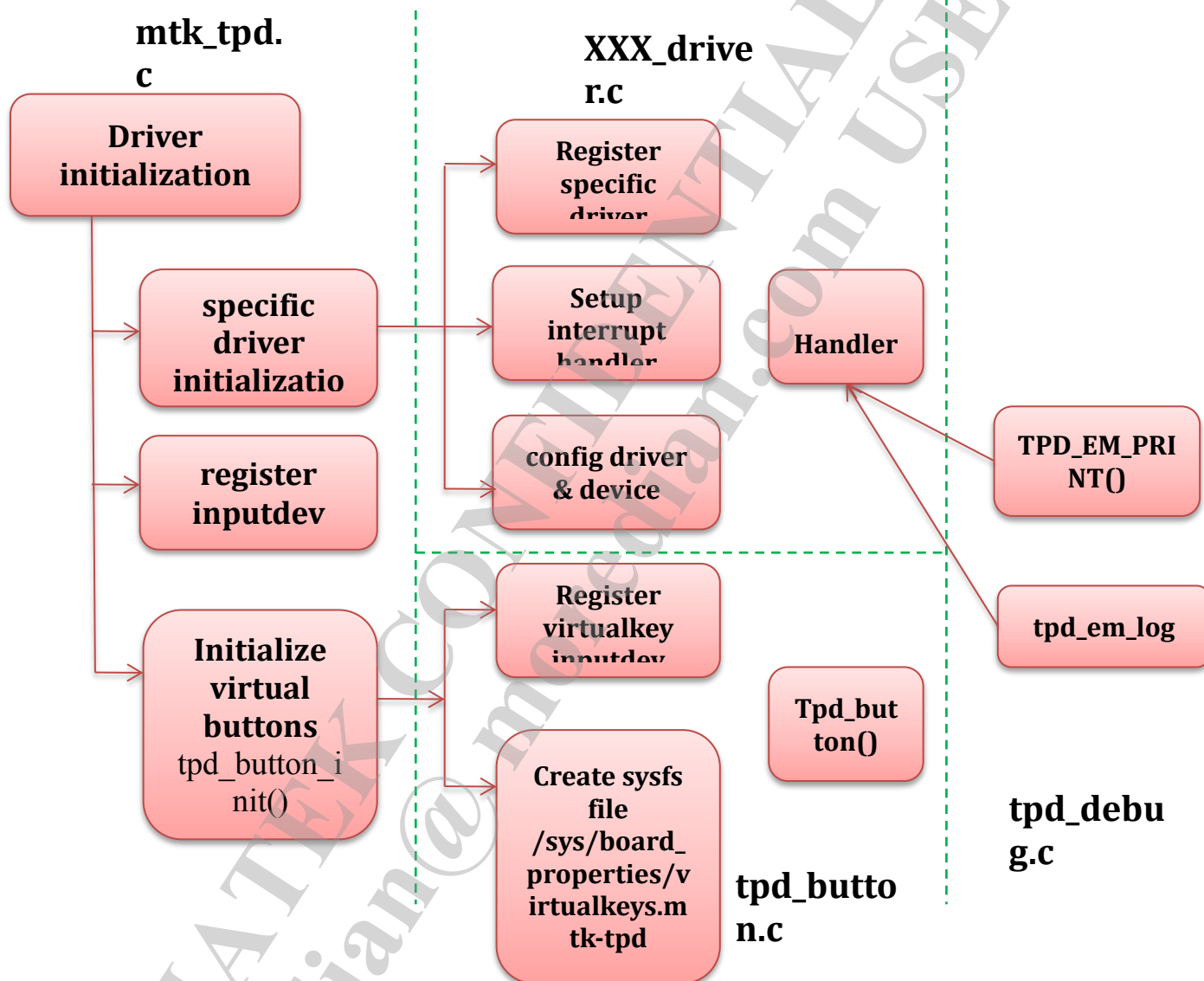
    memset(&msg, 0, sizeof(struct i2c_msg));
    *g_dma_buff_va = addr;
    msg.addr = (client->addr & I2C_MASK_FLAG);
    msg.flags = 0;
    msg.buf = g_dma_buff_pa;
    msg.len = 1 + len;
    msg.ext_flag = (client->ext_flag | I2C_ENEXT_FLAG \
        | I2C_DMA_FLAG);
    msg.timing = GSLTP_I2C_MASTER_CLOCK;

    memcpy(g_dma_buff_va + 1, txbuf, len);
    ret = i2c_transfer(client->adapter, &msg, 1);
    return ret;
}
```

2.2 Mediatek touch driver introduction

driver architecture and useful notes

2.2.1 Touch Panel Driver Architecture



All specific touch driver are mounted on MTK common touch driver which named mtk_tpd.c

2.2.2 driver file

- path for touch common code: (e.g. mtk_tpd.c) :
 - <kernel_ver>/drivers/input/touchscreen/mediatek
- driver for specific touch (e.g. gt1x_tpd.c) :
 - <kernel_ver>/drivers/input/touchscreen/mediatek/<tp_drv>
- customized touch driver file path(e.g. gt1x_tpd_custom.h)
 - <kernel_ver>/drivers/input/touchscreen/mediatek/<tp_drv>/include/<cust_folder>
 - set customized file path in kernel config
 - <kernel_ver>/arch/<arm_ver>/configs/

2.2.3 APIs used in touch

- Use of_find_compatible_node to get your dts node, or use the probe callback of platform drivers to get populated dts node
- please make sure that return type of interrupt handler is irqreturn_t and must return with IRQ_HANDLED

```
struct of_device_id touch_of_match[] = {
    { .compatible = "mediatek,mt6570-touch", },
    { .compatible = "mediatek,mt6735-touch", },
};

node = of_find_matching_node(NULL, touch_of_match);
if (node) {

    static irqreturn_t tpd_interrupt_handler(int irq, void *dev_id)
    {
        if (irq_enabled) {
            irq_enabled = false;
            disable_irq_nosync(touch_irq);
        }
        tpd_flag = 1;
        wake_up_interruptible(&waiter);
        return IRQ_HANDLED;
    }
}
```

- Use irq_of_parse_and_map() to get virtual irq
- Use request_irq() instead to register IRQ
 - interrupt flag should be IRQF_TRIGGER_NONE, since irq_of_parse_and_map() already set trigger type, you can also overwrite the trigger type here)
- Use enable_irq()/disable_irq() to enable and disable irq
 - use disable_irq_nosync() in irq context instead
 - please make sure balance of enable_irq and disable_irq
 - default irq is enabled after request_irq, don't enable_irq again
- Use irq_set_irq_type() to change irq trigger type
- use enable_irq_wake() and disable_irq_wake() for irq with wakeup request

2.2.4 early suspend

- there is no early suspend, please use fb_notifier instead
- touch should wakelock itself when it want to wakeup the system
- code of fb_notifier is in mtk_tpd.c, it's not need to modify

```
static struct notifier_block tpd_fb_notifier;
/* use fb_notifier */
static void touch_resume_workqueue_callback(struct work_struct *work)
{
    TPD_DEBUG("GTP touch_resume_workqueue_callback\n");
    g_tpd_drv->resume(NULL);
    tpd_suspend_flag = 0;
}

static int tpd_fb_notifier_callback(struct notifier_block *self,
```

2.3 touch customization

configuration items

2.3.1 project configuration

- ProjectConfig.mk (device/mediatek/<proj>)
- check information:
 - CUSTOM_KERNEL_TOUCHPANEL = <tp_drv>
 - for Goodix hotknot supported touch, please set <tp_drv> as "GT9XX" for GT9XX series touch, and please set as "GT1XX" for GT1XX series touch

```
CUSTOM_KERNEL_SUB_LENS = dummy_lens
CUSTOM_KERNEL_TOUCHPANEL = GT9XX
```

GT9XX for GT9XX series
that support hotknot

2.3.2 Kernel config

- kernel config file :
 - <proj>_debug_defconfig
 - <proj>_defconfig
- config file path :
 - 32 bits : <kernel_ver>/arch/arm/configs
 - 64 bits : <kernel_ver>/arch/arm64/configs
- check contents in config file
 - CONFIG_MTK_TOUCHPANEL=y
 - CONFIG_INPUT_TOUCHSCREEN=y
 - CONFIG_TOUCH_SCREEN_<tp_drv>=y
 - _<tp_drv> : shows the real touch driver you used, please refer to Kconfig file (<kernel_ver>/drivers/input/touchscreen/mediatek/Kconfig)
- Kconfig: shows which real touch driver is match to your config
 - <kernel_ver>/drivers/input/touchscreen/mediatek/Kconfig
 - e.g. CONFIG_TOUCHSCREEN_MTK_GT9XXTB_HOTKNOT=y, means the real touch driver is GT9XXTB_hotknot

```
config TOUCHSCREEN_MTK_GT9XXTB_HOTKNOT
```

```
bool "GT9XXTB hotknot for Mediatek package"
```

```
default n
```

```
help
```

```
    Say Y here if you have GT9xx touch panel.
```

```
    If unsure, say N.
```

```
    To compile this driver as a module, choose M here: the
    module will be called.
```

```
source "drivers/input/touchscreen/mediatek/GT9XXTB_hotknot/Kconfig"
```

- you should config customized file path in kernel config, please refer to Kconfig for add and update customized file
- step1 : kernel config

- CONFIG_<tp_drv>_FIRMWARE="<cust_folder>"

- CONFIG_<tp_drv>_CONFIG="<cust_folder>"

- e.g.:

- CONFIG_GT9XXTB_FIRMWARE="firmware_default"

- CONFIG_GT9XXTB_CONFIG="config_default"

- step2: check Kconfig

- path : <kernel_ver>/drivers/input/touchscreen/mediatek/<tp_drv>/Kconfig

```
config GT9XXTB_FIRMWARE
```

```
    string "GT9XXTB_hotknot for Mediatek firmware"
```

```
config GT9XXTB_CONFIG
```

```
    string "GT9XXTB_hotknot for Mediatek config"
```

- Make file : check and add your customized file path

- path : <kernel_ver>/drivers/input/touchscreen/mediatek/<tp_drv>/makefile

```
ccflags-y += -I$(srctree)/drivers/input/touchscreen/mediatek/GT9XXTB_hotknot/
```

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

```
ccflags-y += -I$(srctree)/drivers/input/touchscreen/mediatek/GT9XXTB_hotknot/include/$(CONFIG_
```

```
ccflags-y += -I$(srctree)/drivers/input/touchscreen/mediatek/GT9XXTB_hotknot/include/$(CONFIG_
```

```
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 += gt9xx_driver.o
```

```
obj-y += goodix_tool.o
```

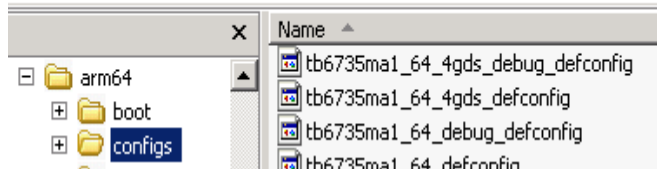
```
obj-y += gt9xx_update.o
```

make file

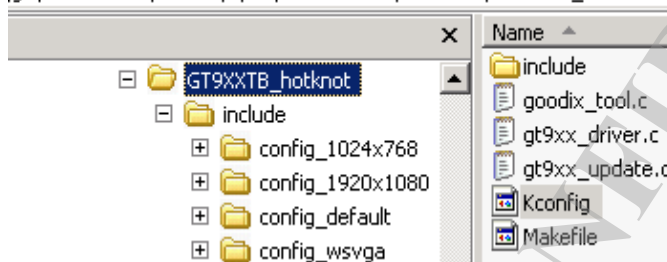
match with
the prev

- file and folder examples :

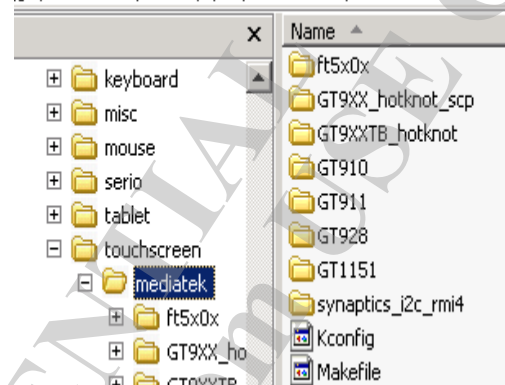
\\kernel-3.18\\arch\\arm64\\configs



\\git\\kernel-3.18\\drivers\\input\\touchscreen\\mediatek\\GT9XXTB_hotknot



\\git\\kernel-3.18\\drivers\\input\\touchscreen\\mediatek



- config for specific touch driver

```
/*
#define GTP_CUSTOM_CFG 0
#define GTP_DRIVER_SEND_CFG 1
#define GTP_HAVE_TOUCH_KEY 0

CONFIG_TOUCHSCREEN_MTK=y
CONFIG_GTP_DRIVER_SEND_CFG=y
CONFIG_GTP_AUTO_UPDATE=y
CONFIG_GTP_HEADER_FW_UPDATE=y
CONFIG_GTP_CREATE_WR_NODE=y

config GTP_DRIVER_SEND_CFG
bool "GTP_DRIVER_SEND_CFG"
default n
help
Say Y here if you have GT9XXTB_hotknot touch panel GTP_DEBUG_FUNC_ON.

If unsure, say N.

To compile this driver as a module, choose M here: the
module will be called.
```

2.4 touch customization

dws and dts

2.4.1 DWS file- I2C

- check i2c setting in dws file
 - <kernel_ver>/drivers/misc/mediatek/dws/<platform>/<proj>.dws
 - e.g. touch panel uses I2C id0
 - VarName1 should be
 - GPIO_I2C0_SDA_PIN
 - GPIO_I2C0_SCL_PIN

GPIO Setting EINT Setting ADC Setting KEYPAD Setting PMIC Setting POWER Setting MD1_EINT Setting																	
	Ei...	Def.Mode	M0	M1	M2	M3	M4	M5	M6	M7	In...	In...	D...	In	Out	O...	VarName1
GPIO84	<input type="checkbox"/>	1:SDA0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>							<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	IN	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	GPIO_I2C0_SDA_PIN
GPIO85	<input type="checkbox"/>	1:SCL0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>							<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	IN	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	GPIO_I2C0_SCL_PIN

- I2C address config

GPIO EINT ADC KEYPAD I2C PMIC ClockBuffer POWER MD1_EINT									
Slave Device			Channel			Device Address			
CAP TOUCH			I2C CHANNEL 1			0x5D			

2.4.2 DWS file - CTP RST/EINT Pin

- RST Pin
 - VarName1 should be GPIO_CTP_RST_PIN

GPIO Setting EINT Setting ADC Setting KEYPAD Setting PMIC Setting POWER Setting MD1_EINT Setting																	
	Ei...	Def.Mode	M0	M1	M2	M3	M4	M5	M6	M7	In...	In...	D...	In	Out	O...	VarName1
GPIO13	<input type="checkbox"/>	0:GPIO13	<input checked="" type="checkbox"/>								<input type="checkbox"/>	<input type="checkbox"/>	OUT	<input type="checkbox"/>	<input checked="" type="checkbox"/>		GPIO_CTP_RST_PIN

- EINT Pin
 - check EINT mode
 - VarName1 should be GPIO_CTP_EINT_PIN

GPIO Setting EINT Setting ADC Setting KEYPAD Setting PMIC Setting POWER Setting MD1_EINT Setting																	
	Ei...	Def.Mode	M0	M1	M2	M3	M4	M5	M6	M7	In...	In...	D...	In	Out	O...	VarName1
GPIO10	<input checked="" type="checkbox"/>	0:GPIO10									<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	IN				GPIO_CTP_EINT_PIN

- EINT setting: please refer to CTP spec for setting EINT setting,
- note: please select **TOUCH** for EINT Var

GPIO EINT ADC KEYPAD I2C PMIC ClockBuffer POWER MD1_EINT									
	EINT Var			Debounce Time (ms)		Polarity	Sensitive_Level	Debounce En	
EINT9	NC			0					
EINT10	TOUCH			0		Low	Edge	Disable	

please select TOUCH

2.4.3 dts file

- file path: <kernel_ver>/arch/<arm_ver>/boot/dts/<proj>.dts
 - please include platform dts file (< platform >.dtsi)

- please include cust.dtsi (maybe had included in <platform>.dtsi , if so, doesn't need to include again in <proj>.dts)

```
#include "mt6735m.dtsi"
#include "tb6735ma1_64_bat_setting.dtsi"
#include "cust.dtsi"
```

- touch node:
 - root node: <kernel_ver>/arch/<arm_ver>/boot/dts/<platform>.dtsi
 - compatible string should match with software, SW use of find_matching_node() read device tree information from dts file (mtk_tpd.c)
- note: device tree node name and compatible string should be lowercase

```
touch: touch@ {
    compatible = "mediatek,mt6735-touch",
                "mediatek,mt6735m-touch";
    vtouch-supply = <&mt_pmic_vgpl_ldo_reg>;
};
```

```
struct of_device_id touch_of_match[] = {
    { .compatible = "mediatek,mt6735-touch", },
    { .compatible = "mediatek,mt6580-touch", },
    { .compatible = "mediatek,mt8173-touch", },
    { .compatible = "mediatek,mt6755-touch", },
    { .compatible = "mediatek,mt6797-touch", },
    { .compatible = "mediatek,mt8163-touch", },
    {},
};
```

```
void tpd_get_dts_info(void)
{
    struct device_node *node1 = NULL;
    int key_dim_local[16], i;

    node1 = of_find_matching_node(node1, touch_of_match);
}
```

2.4.4 device tree node

- attach node for touch, <proj>.dts

```
{touch {
    tpd-resolution = <800 1280>;
    use-tpd-button = <0>;
    tpd-key-num = <3>;
    tpd-key-local = <139 172 158 0>;
    tpd-key-dim-local = <90 883 100 40 230 883 100 40 370 883 100 40 0 0 0 0>;
    tpd-max-touch-num = <5>;
    tpd-filter-enable = <1>;
    tpd-filter-pixel-density = <93>;
    tpd-filter-custom-prameters = <0 0 0 0 0 0 0 0 0 0 0>;
    tpd-filter-custom-speed = <0 0 0>;
    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";
};
```


2.4.5 dts node--Parameters Introduction

Parameter	Introduction	Comments
tpd_resolution[2]	touch panel resolution info for x and y axis	tpd_resolution[0]: LCM resolution tpd_resolution[1]: LCM resolution
use_tpd_button	define whether the touch panel use virtual key	1 stands for touch panel use touch 0 stands for touch panel not use touch
tpd_key_num	The number of the touch virtual key. you can not set this parameter if use_tpd_button is 0.	The max of the key number is 4.
tpd_key_local[4]	the Linux key value if touch virtual key is used, you can not set this parameter if use_tpd_button is 0.	fill in Linux key code which will be used on touch panel, layout from left to right of the touch panel. tpd_key_local[0], tpd_key_local[1], tpd_key_local[2], tpd_key_local[3]
tpd_key_dim_local[4] (include 4 parameters tpd_key_dim_local[4].key_x, tpd_key_dim_local[4].key_y, tpd_key_dim_local[4].key_width, tpd_key_dim_local[4].key_high)	the key layout info if touch virtual key is used, you can not set this parameter if use_tpd_button is 0.	every tpd_key_dim_local[i] corresponds to a key on the touch panel. tpd_key_dim_local[i].key_x: the x-axis coordinate of the key tpd_key_dim_local[i].key_y: the y-axis coordinate of the key tpd_key_dim_local[i].key_width: the width of the key tpd_key_dim_local[i].key_high: the height of the key

2.4.6 device tree node

- attach node of touch--pinctrl
 - mainly check reset pin and EINT pin

```
&pio {
    CTP_pins_default: eint0default {
    };
};
```

```

CTP_pins_eint_as_int: eint@0 {
    pins_cmd_dat {
        pins = <PINMUX_GPIO10__FUNC_GPIO10>;
        slew-rate = <0>;
        bias-disable;
    };
};
CTP_pins_eint_output0: eintoutput0 {
    pins_cmd_dat {
        pins = <PINMUX_GPIO10__FUNC_GPIO10>;
        slew-rate = <1>;
        output-low;
    };
};
CTP_pins_eint_output1: eintoutput1 {
    pins_cmd_dat {
        pins = <PINMUX_GPIO10__FUNC_GPIO10>;
        slew-rate = <1>;
        output-high;
    };
};
CTP_pins_rst_output0: rstoutput0 {
    pins_cmd_dat {
        pins = <PINMUX_GPIO62__FUNC_GPIO62>;
        slew-rate = <1>;
        output-low;
    };
};
CTP_pins_rst_output1: rstoutput1 {
    pins_cmd_dat {
        pins = <PINMUX_GPIO62__FUNC_GPIO62>;
        slew-rate = <1>;
        output-high;
    };
};
/* TOUCH end */

```

2.4.7 device tree node—I2C & EINT

- i2c and interrupt info for touch
 - modify dws, nodes as below in cust.dtsi will be generated by dws file
 - note: interrupt trigger type (e.g. IRQ_TYPE_EDGE_FALLING) should be the ones that defined in the following files
 - <kernel_ver>/include/dt-bindings/interrupt-controller/arm-gic.h
 - <kernel_ver>/include/dt-bindings/interrupt-controller/irq.h

```
&i2c0 {
    pinctrl-names = "default";
    pinctrl-0 = <&i2c0_pins_a>;
    status = "okay";
```

```
cap_touch@38 {
    compatible = "mediatek,cap_touch";
    reg = <0x38>;
    interrupt-parent = <&pio>;
    interrupts = <46 IRQ_TYPE_EDGE_FALLING>;
    int-gpio = <&pio 35 0>;
    rst-gpio = <&pio 45 0>;
};

static const struct of_device_id tpd_of_match[] = {
    {.compatible = "mediatek,cap_touch"},
    {},
};
```

- modify dws file is needed, nodes as below will be generated by dws file (cust.dtsi)
- e.g. attach nodes in cust.dtsi("&touch" means this is attach information of "touch" node)

```
&touch {
    interrupt-parent = <&eintc>;
    interrupts = <10 IRQ_TYPE_EDGE_FALLING>;
    debounce = <10 0>;
    status = "okay";
};

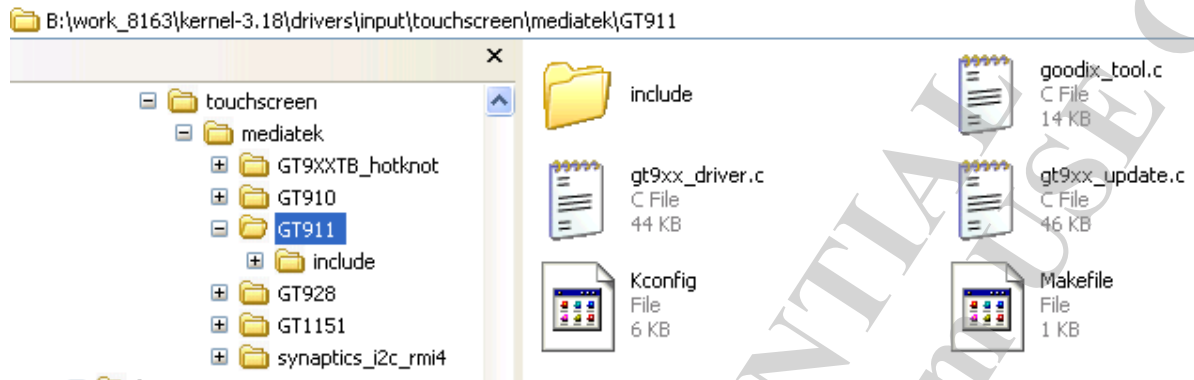
3 &touch {
4     vtouch-supply = <&mt_pmic_vgpi1_ldo_reg>;
5     status = "okay";
6 };

&i2c1 {
    #address-cells = <1>;
    #size-cells = <0>;
    cap_touch@5d {
        compatible = "mediatek,cap_touch";
        reg = <0x5d>;
        status = "okay";
    };
};
```

2.5 Add a New Touch Driver

2.5.1 Touch Screen – Add New Driver (1/5)

- How to add a new touch driver in codebase
 - Path: ./<kernel_ver>/drivers/input/touchscreen/mediatek/
 - Add your driver source code.



2.5.2 Touch Screen – Add New Driver (2/5)

- Add the Makefile and Kconfig file driver needs, can be modeled on the other driver. Some customizable option defined here, too.

```

B:\work_8163\kernel-3.18\drivers\input\touchscreen\mediatek\GT911\Makefile
0 10 20 30 40 50 60 70 80 90 100
4 ccflags-y += -I$(srctree)/drivers/input/touchscreen/mediatek/
5 ccflags-y += -I$(srctree)/drivers/input/touchscreen/mediatek/GT911/include/$(CONFIG_GT911_FIRMWARE)/
6 ccflags-y += -I$(srctree)/drivers/input/touchscreen/mediatek/GT911/include/$(CONFIG_GT911_CONFIG)/
7 ccflags-y += -I$(srctree)/drivers/misc/mediatek/include/mt-plat/
8 ccflags-y += -I$(srctree)/drivers/misc/mediatek/include/mt-plat/$(MTK_PLATFORM)/include/
9
10 obj-y += goodix_tool.o
11 obj-y += gt9xx_driver.o
12 obj-y += gt9xx_update.o
    
```

```

B:\work_8163\kernel-3.18\drivers\input\touchscreen\mediatek\GT911\Kconfig
0 10 20 30 40 50 60
1 #
2 # Touchscreen driver configuration
3 #
4 if TOUCHSCREEN_MTK_GT911
5
6 config GT911_FIRMWARE
7     string "GT911 for Mediatek firmware"
8
9 config GT911_CONFIG
10     string "GT911 for Mediatek config"
11
12 config GTP_DRIVER_SEND_CFG
13     bool "GT911 for Mediatek package"
14     default n
    
```

2.5.3 Touch Screen – Add New Driver (3/5)

- Modify <kernel_ver>/drivers/input/touchscreen/mediatek/Makefile. Add specific driver build option in Makefile.

```

B:\work_8163\kernel-3.18\drivers\input\touchscreen\mediatek\Wakefile
0 10 20 30 40 50 60 70
4 obj-y += mtk_tpd.o
5 obj-y += tpd_button.o
6 obj-y += tpd_calibrate.o
7 obj-y += tpd_debug.o
8 obj-y += tpd_default.o
9 obj-y += tpd_init.o
10 obj-y += tpd_misc.o
11 obj-y += tpd_setting.o
12
13 obj-$(CONFIG_TOUCHSCREEN_MTK_GT1151) += GT1151/
14 obj-$(CONFIG_TOUCHSCREEN_MTK_GT910) += GT910/
15 obj-$(CONFIG_TOUCHSCREEN_MTK_GT9XXTB_HOTKNOT) += GT9XXTB_hotknot/
16 obj-$(CONFIG_TOUCHSCREEN_MTK_SYNAPTICS_I2C_RMI4) += synaptics_i2c_rmi4/
17 obj-$(CONFIG_TOUCHSCREEN_MTK_GT928) += GT928/
18 obj-$(CONFIG_TOUCHSCREEN_MTK_GT911) += GT911/

```

2.5.4 Touch Screen – Add New Driver (4/5)

- Modify /<kernel_ver>/drivers/input/touchscreen/mediatek/Kconfig.

Add real driver config option in Kconfig.

```

B:\work_8163\kernel-3.18\drivers\input\touchscreen\mediatek\Kconfig
0 10 20 30 40 50 60
81
82 source "drivers/input/touchscreen/mediatek/GT910/Kconfig"
83
84 config TOUCHSCREEN_MTK_GT911
85     bool "GT1151 for Mediatek package"
86     default n
87     help
88     Say Y here if you have GT911 touch panel.
89
90     If unsure, say N.
91
92     To compile this driver as a module, choose M here: the
93     module will be called.
94
95 source "drivers/input/touchscreen/mediatek/GT911/Kconfig"
96 endif

```

2.5.5 Touch Screen – Add New Driver (5/5)

- Modify <kernel_ver>/arch/<arm_ver>/configs/<proj>_defconfig
- Modify <kernel_ver>/arch/<arm_ver>/configs/<proj>_debug_defconfig

```

B:\work_8163\kernel-3.18\arch\arm64\configs\mt8163m1_64_defconfig
0 10 20 30 40 50 60
:39 # CONFIG_INPUT_KEYBOARD is not set
:40 # CONFIG_INPUT_MOUSE is not set
:41 CONFIG_INPUT_TOUCHSCREEN=y
:42 CONFIG_TOUCHSCREEN_MTK=y
:43 CONFIG_GTP_DRIVER_SEND_CFG=y
:44 CONFIG_GTP_AUTO_UPDATE=y
:45 CONFIG_GTP_HEADER_FW_UPDATE=y
:46 CONFIG_GTP_CREATE_WR_NODE=y
:47 CONFIG_GTP_POWER_CTRL_SLEEP=y
:48 CONFIG_GTP_COMPATIBLE_MODE=y
:49 CONFIG_TOUCHSCREEN_MTK_GT911=y
:50 CONFIG_GT911_FIRMWARE="firmware1"
:51 CONFIG_GT911_CONFIG="config1"

```

2.6 Touch Panel Driver Debug

2.6.1 Touch Panel Driver Debug

- confirm touch driver has been build in load
 - if no, check project config, kernel config, Makefile and Kconfig file
- check touch log
 - enable touch log : use "pr_err" for all log macro in driver, e.g.
 - #define GTP_DEBUG(fmt, arg...) pr_err("<<-GTP-DEBUG->> [%d]"fmt"\n", __LINE__, ##arg)
- check touch input event
 - use "adb shell getevent -i" to get input event number of "mtk-tpd".
 - e.g. if "mtk-tpd" is event3, use "adb shell getevent /dev/input/event3" get touch input event.

```

[tpd_em_log] raw_x = 300, raw_y = 656, cal_x = 190, cal_y = 306, z1 = 154, z2 = 681, state = down (+10 ms)
[tpd_em_log] raw_x = 317, raw_y = 623, cal_x = 187, cal_y = 300, z1 = 162, z2 = 653, state = down (+10 ms)

```

- touch input event type/code are shown as below table.

type	code	value
1=EV_KEY	0x14a(330) = BTN_TOUCH	Down=1, Up=0
3=EV_ABS	0x30(48) = ABS_MT_TOUCH_MAJOR	1
3=EV_ABS	0x35(53) = ABS_MT_POSITION_X	x=284
3=EV_ABS	0x36(54) = ABS_MT_POSITION_Y	y=366
3=EV_ABS	0x39(57) = ABS_MT_TRACKING_ID	point index=1
0=EV_SYN	0x2(2) = SYN_MT_REPORT	0
0=EV_SYN	0x0(0) = SYN_REPORT	0

- check dws and dts setting with schematics
 - AVDD regulator configuration

- check dws & dts for I2C info and interrupt information

```
cap_touch@38 {
    compatible = "mediatek,cap_touch";
    reg = <0x38>;
    interrupt-parent = <&pio>;
    interrupts = <46 IRQ_TYPE_EDGE_FALLING>;
    int-gpio = <&pio 35 0>;
    rst-gpio = <&pio 45 0>;
};

touch: touch {
    compatible = "mediatek,mt6735-touch";
    vtouch-supply = <&mt_pmic_vgp1_ldo_reg>;
};
```

software check

regulator control should balance bwtween enable and disable

regulator_set_voltage(), regulator_enable().....

use irq_of_parse_and_map() to get virtual irq

use request_irq() to register IRQ

enable_irq() should balance with disable_irq()

- check the waveform of VDD/SCL/SDA/RESET/INT pins by scope to make sure touch IC works normally.
- For coordinate not match with LCM coordinate issue
 - check touch FW and customization table
 - check TPD_CALIBRATION_MATRIX
 - #define TPD_CALIBRATION_MATRIX_ROTATION_NORMAL
 - #define TPD_CALIBRATION_MATRIX_ROTATION_FACTORY