2020

Introduction to MCU Example

Index

1. Preface

The document is to help developer to understand this example code. The code is not part of Parade's Linux touchscreen driver, and was created as an example only, to work without the support of any OS. The driver describes:

1) How to communicate with Parade Touch Chips

2) How to parse touch report message

3) How to update a whole firmware and a configure file

4) How to perform MFG test: Auto Shorts, CM, CP Selftest

5) How to print MFG test result in unified format

In following introduction, Parade TP driver for non-Linux system is called as “MCU example” to be not confused with “TP driver” which means for any kind of driver for TP.

2. Overview

The released example code includes following files:

pt_mcu_example_tma5xx.c
pt_mcu_example_tt7xxx.c
pt_mcu_port.c
pt_mcu_port.h

And the files for FW update are not released along with the example, they are:

fw.h
fw_conf.h
tma525.H
TT7000.H

pt_mcu_example_tma5xx.c is the reference code for following products (So-called Gen5 and Gen6 of Parade TrueTouch products):

TMA5XX
TMA448
TMA445A
TT21XXX
TT31XXX
TT4XXXX

pt_mcu_example_tt7xxx.c is the reference code for TT7XXX only.

pt_mcu_port.c is a porting file based on development environment.

pt_mcu_port.h helps to share the declaration and development header file.

fw.h is only for Gen5/6 products and it is released to customer with the

name “’project’.h” (“project” is name of customer project). It includes two arrays: cyttsp4_ver[] and cyttsp4_image[].

fw_conf.h is only for Gen5/6 products and it is released to customer with the name “driver.h”. It includes serveral arrays: ttconfig_fw_ver[], cyttsp4_param_regs[], cyttsp4_param_size[] and cyttsp4_param_addr[], while only first Two arrays are used.

tma525.H and **TT7000.H** are translated from binary file. They include an array and works same as binary file.

3. Communication and Structure Design

3.1 Communication

There are four types for register operation. For any read action, it should start after the interrupt line is asserted.

Figure 3-1 Specified write operation:

Figure 3-2 Default read operation:

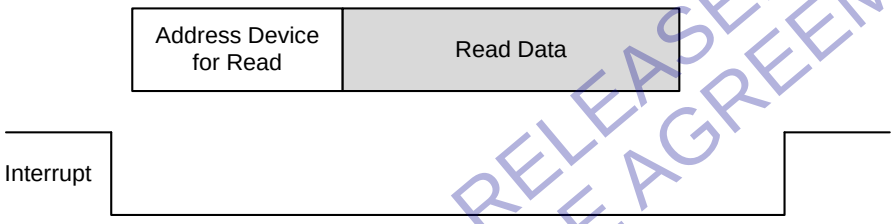


Figure 3-3 Specified read operation:

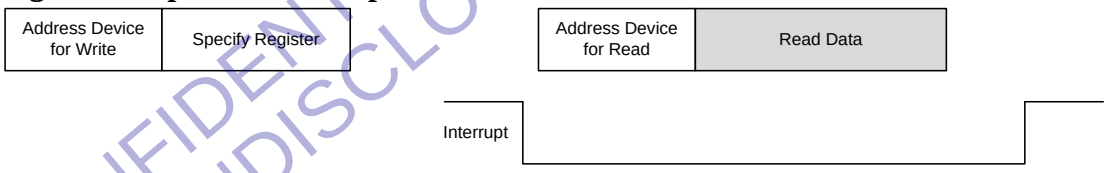
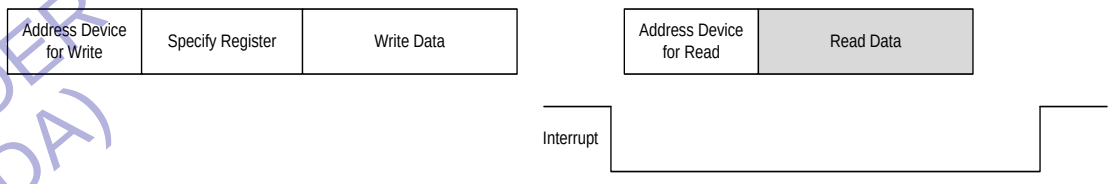
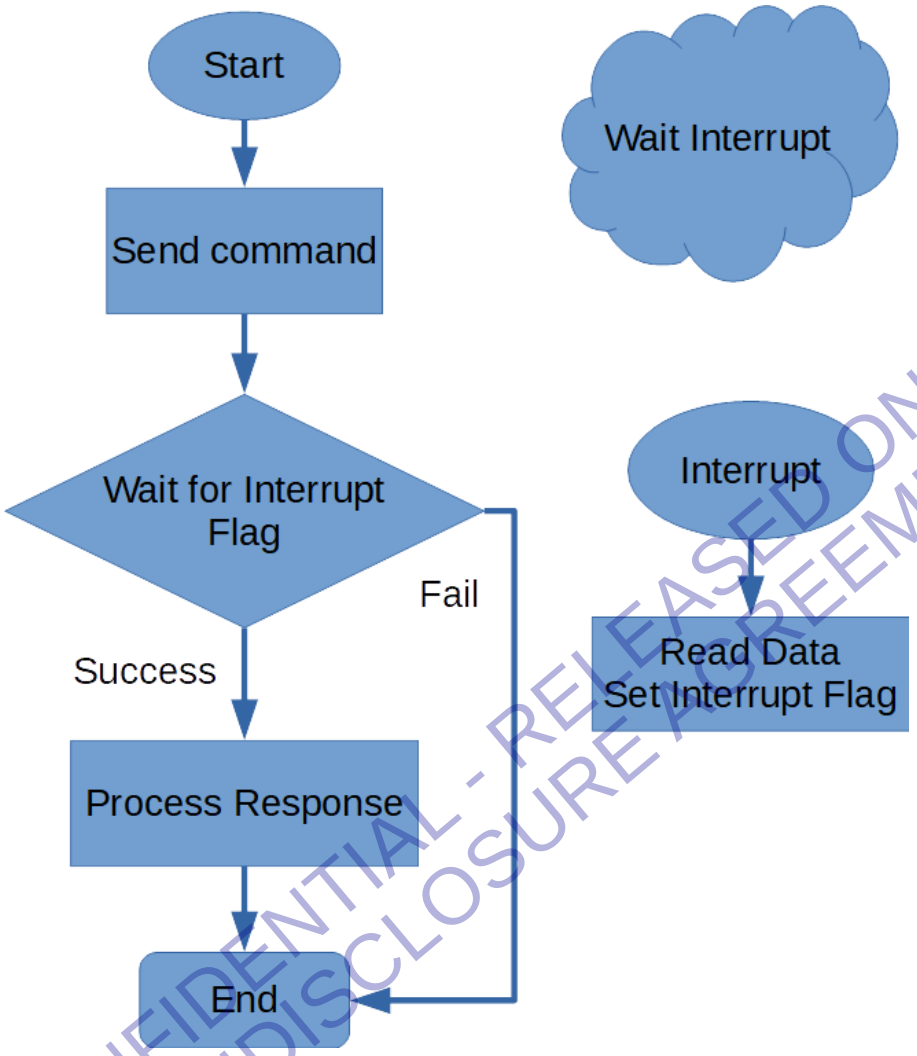


Figure 3-4 Specified write-read operation:



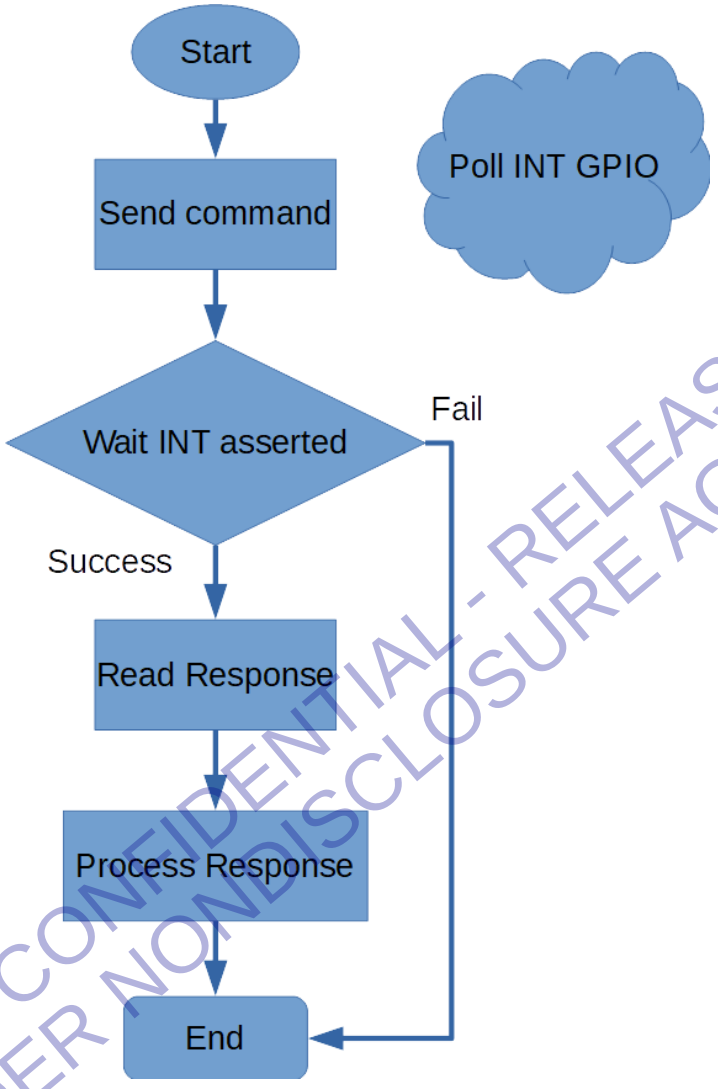
3.2 Two type of model to send and resolve response

Figure 3-5 Register interrupt to monitor the interrupt line:



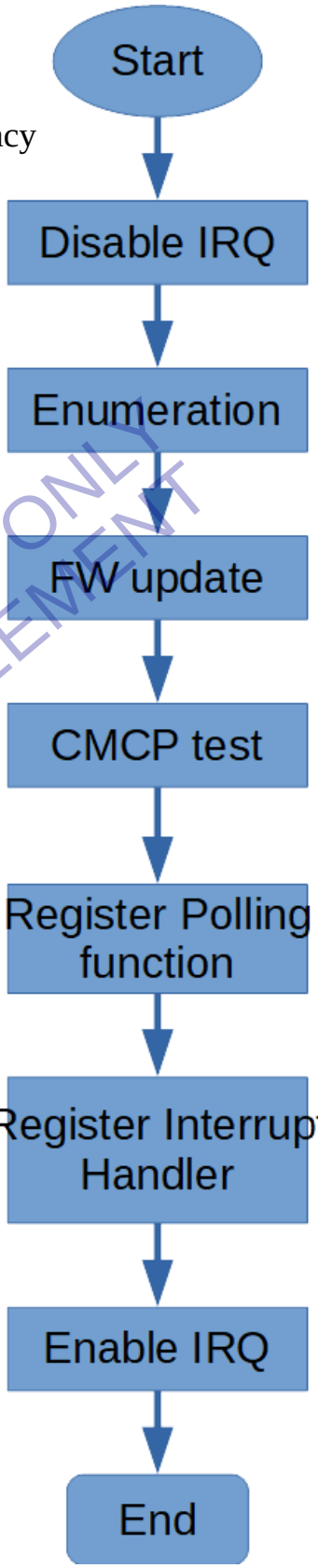
This example code provide the second way to organize functions. The reason is that the interrupt is not always in control in some platform. The polling way is easy to be ported.

Figure 3-6 Poll INT GPIO to monitor the interrupt line:



3.3 Example code workflow overview

MCU Example has to disable IRQ during polling INT GPIO. For touch report, interrupt is still an efficiency way. (**Figure 3-7 An overview for MCU Example**)

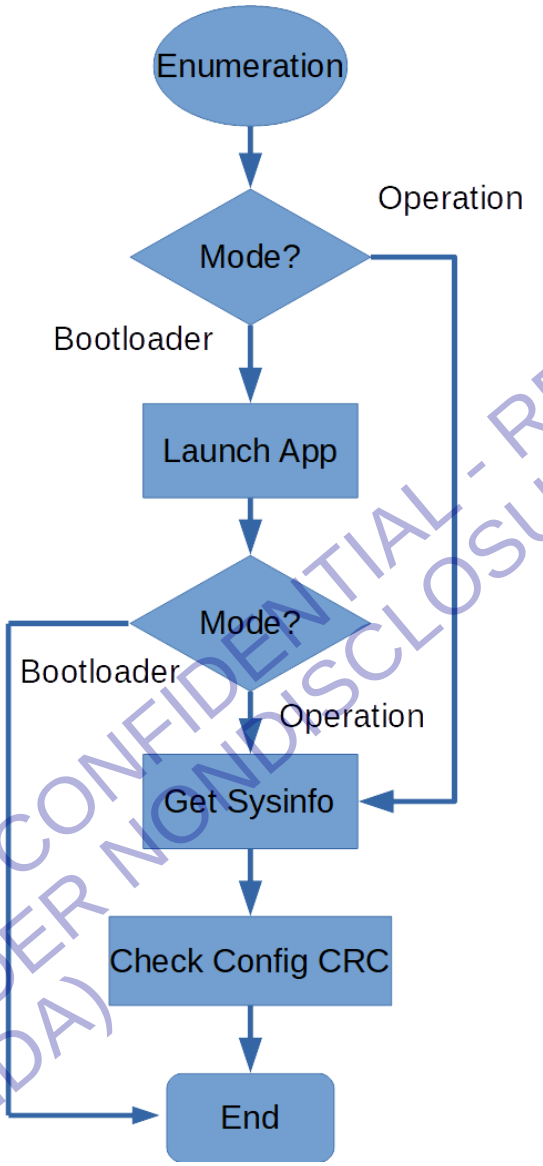


4. MCU Example Workflow Introduction

4.1. Enumeration (Gen5/6)

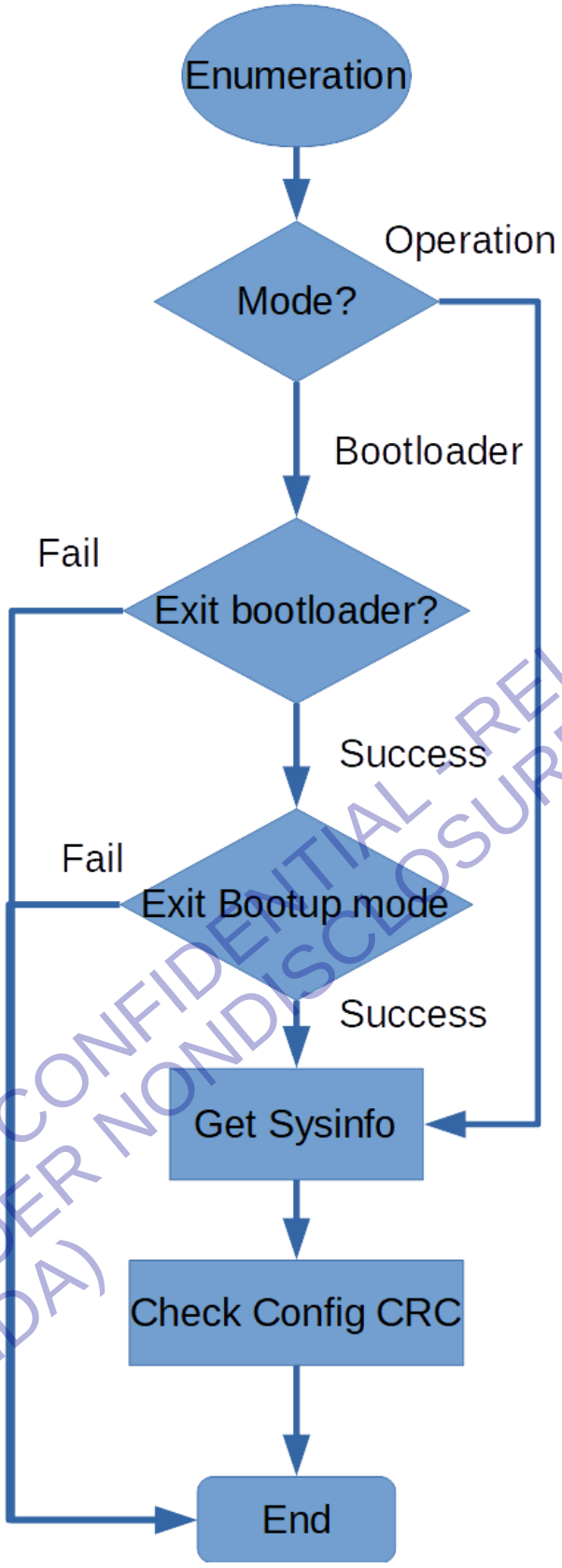
Enumeration flow means to have a basic check for IC state and try to obtain FW information and check the CRC of configure area of the chip.

Figure 4-1 Enumeration Flow of Gen5/6



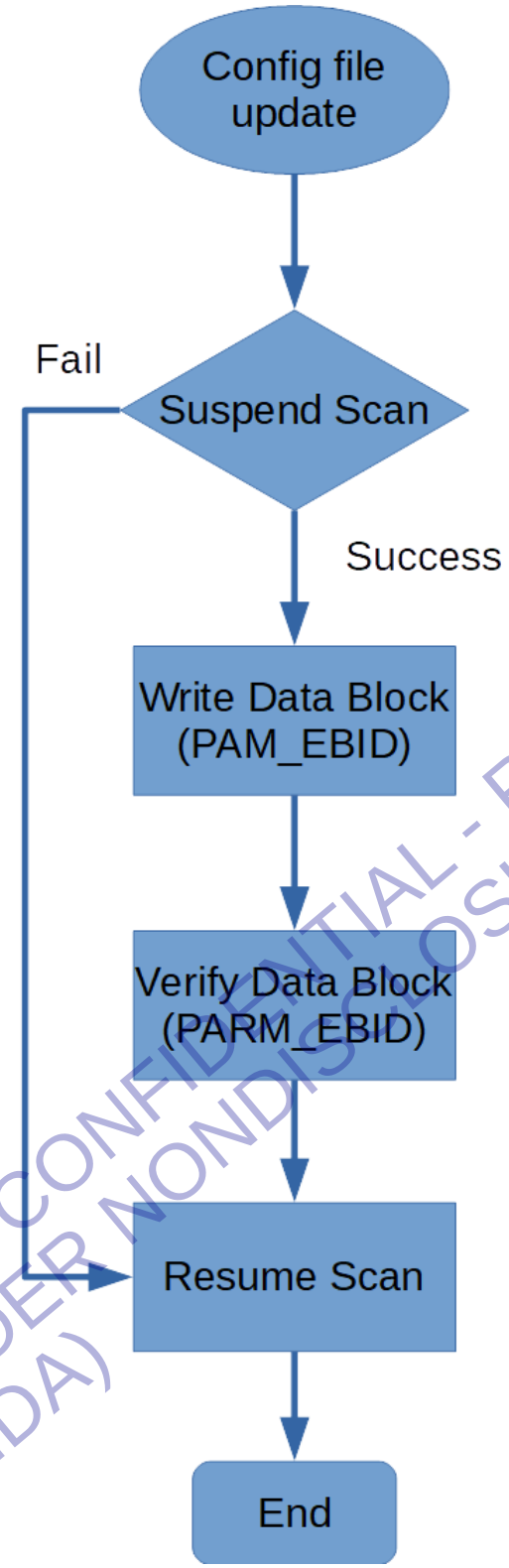
4.2. Enumeration (TT7XXX)

Figure 4-2 Enumeration Flow of TT7XXX



4.3. Config update (Gen5/6)

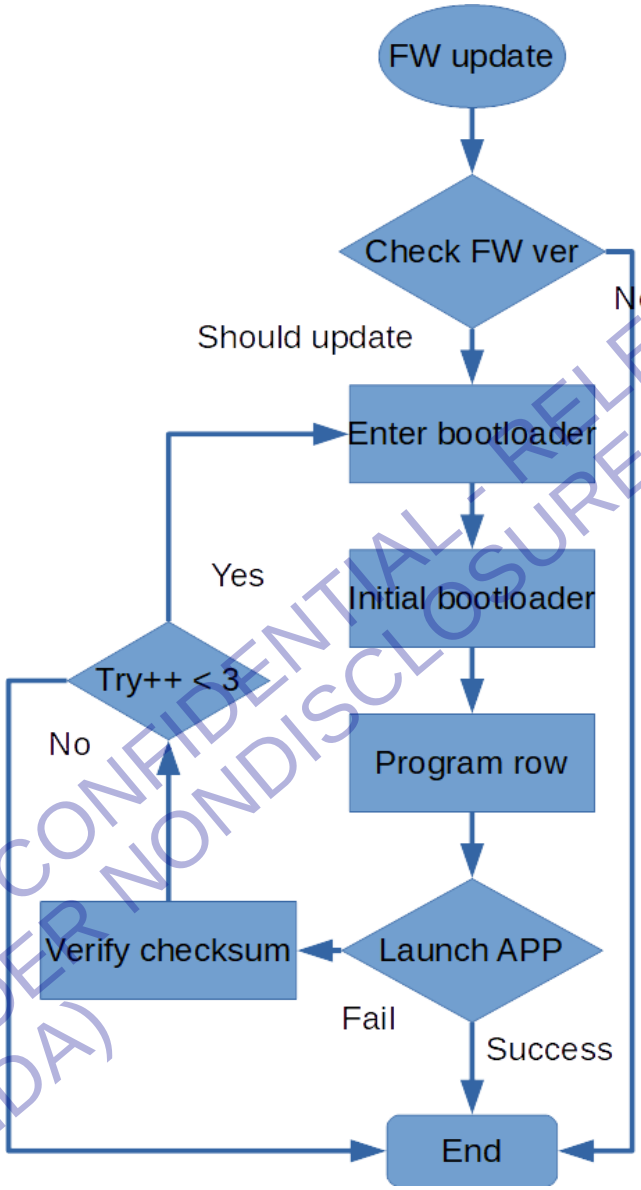
Figure 4-3 Configure file Update Flow



4.4. FW update (Gen5/6)

Here is a brief introduction about how to program Gen5/6 device. The example code provides demo both for header file (project.h) update and binary file update.

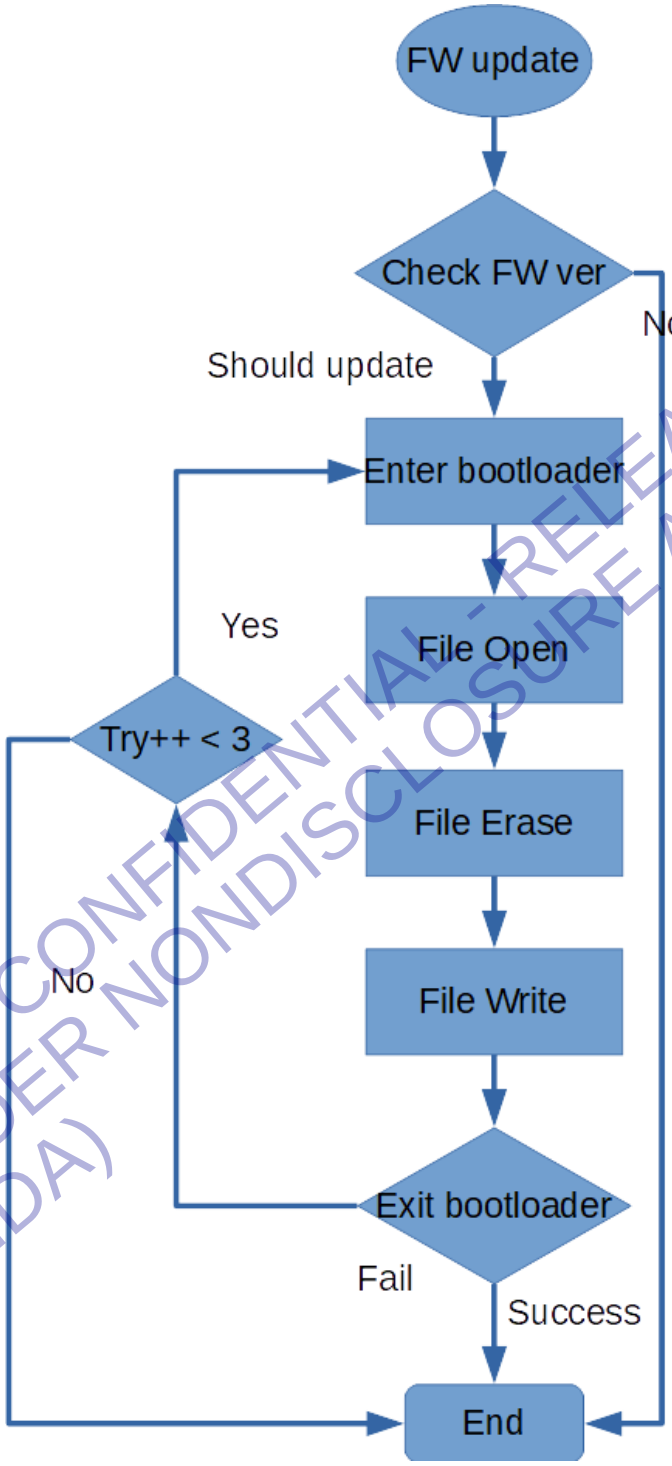
Figure 4-4 Firmware update flow for Gen5/6



4.5. FW update (TT7XXX)

Here is a brief introduction about how to program TT7XXX device with a binary FW.

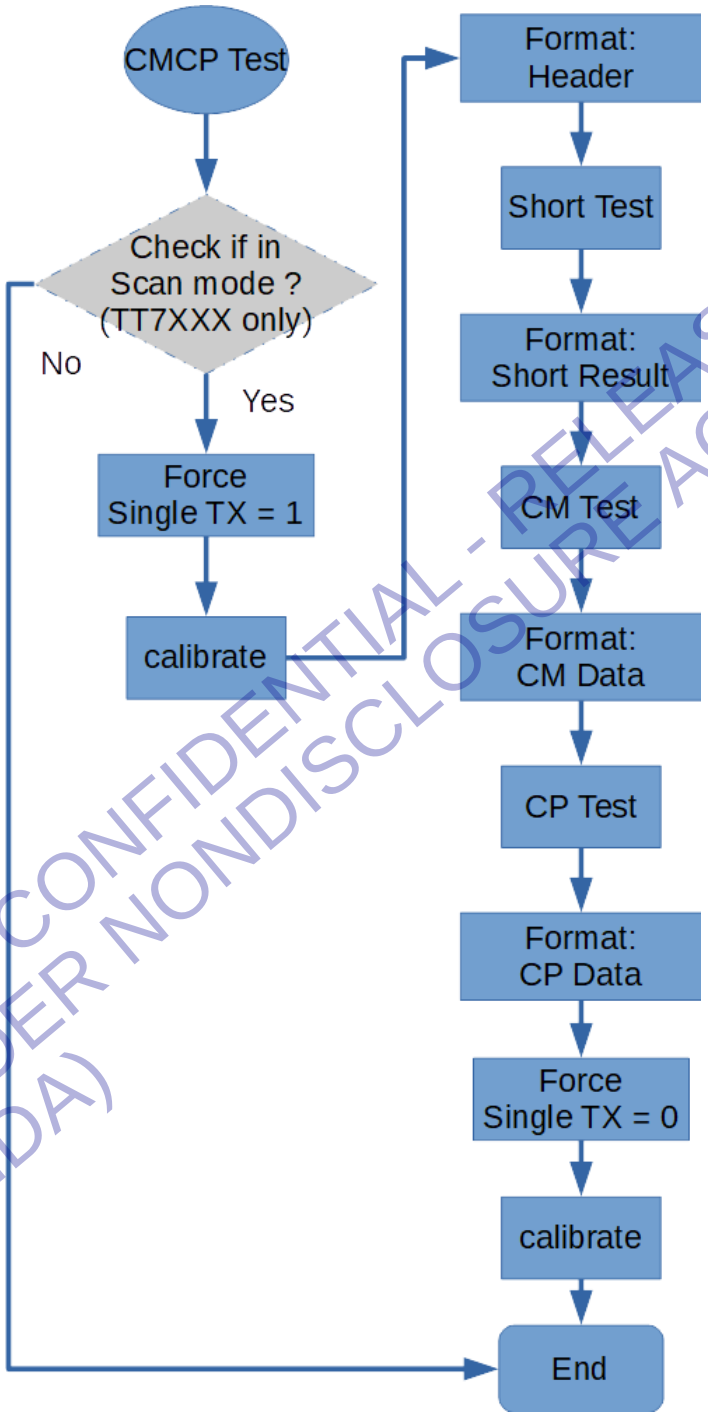
Figure 4-5 Firmware update flow of TT7XXX



4.6. CMCP test

CMCP test is to check whether the TP is in a good state and also provide calibration support. This feature is used for factory test after the device is assembled or create a test jig.

Figure 4-6 CMCP test flow



4.7. Debug Info Print

This feature is to help to FW engineer to tune touch performance. It depends on interrupt to resolve response and a timer to send query command periodically.

Figure 4-7 Use a timer to query noise metric periodically

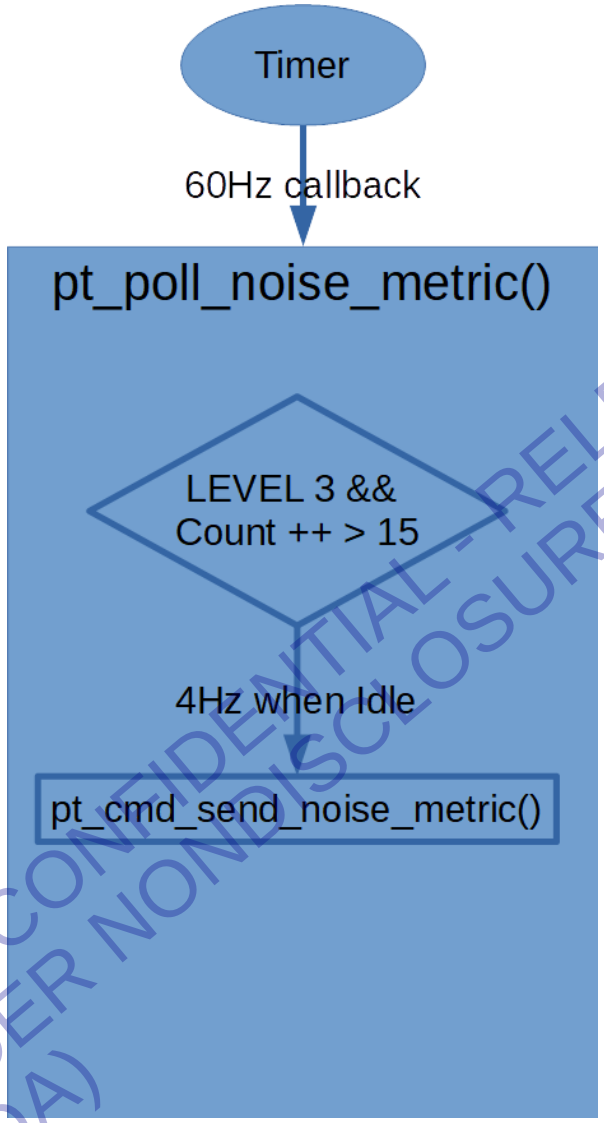
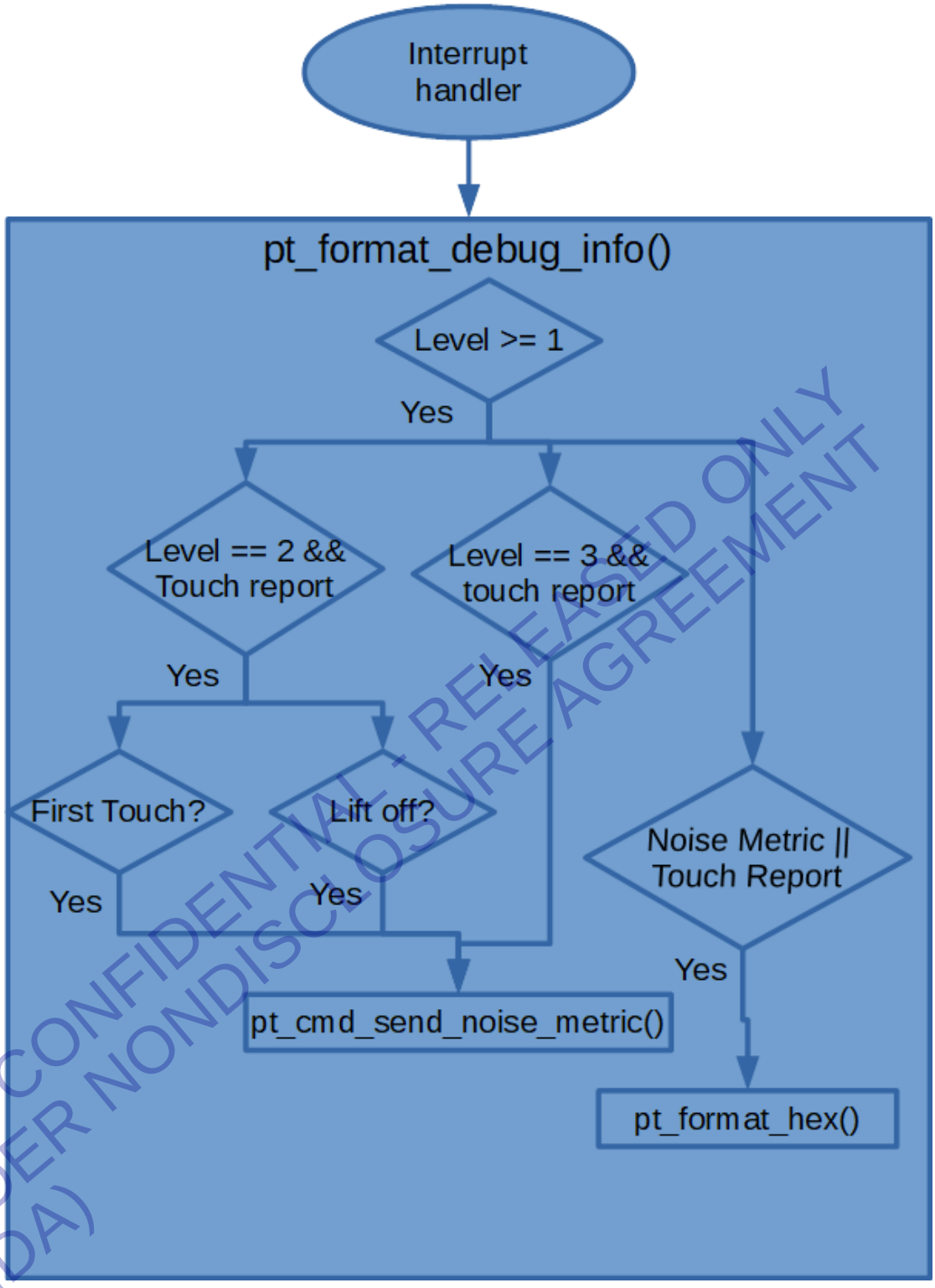


Figure 4-8 Parse the touch report and noise metric in interrupt handler



5. Porting Interface

5.1. GPIO and I2C operation

For GPIO and I2C operation, users only need to finish following macro definitions.

```
#define I2C_READ(read_buf, read_len) …
#define I2C_WRITE(write_buf, write_len) …

#define POWER_ON_2V8() …
#define POWER_ON_1V8() …
#define GPIO_SET_RST_MODE_OUTPUT() …
#define GPIO_SET_RST_HIGH() …
#define GPIO_SET_RST_LOW() …
#define GPIO_SET_INT_MODE_INPUT() …
#define GPIO_READ_INT_STATE() …
```

Example:

```c
/* Low-level funcitons */
/* Delay functions */
#define DELAY_MS(ms) pt_delay_ms(ms)
#define DELAY_US(us) pt_delay_us(us)

/* Power functions */
#define POWER_ON_2V8() do { pt_set_vdda_gpio(1); } while(0)
#define POWER_ON_1V8() do { pt_set_vddd_gpio(1); } while(0)

/* I2C function*/
#define I2C_READ(read_buf, read_len) \
    pt_i2c_read(I2C_ADDR, read_buf, read_len, PT_MAX_PIP_MSG_SIZE)
#define I2C_WRITE(write_buf, write_len) \
    pt_i2c_write(I2C_ADDR, write_buf, write_len)

/* GPIO functions */
#define GPIO_HIGH (1)
#define GPIO_LOW (0)

#define GPIO_SET_RST_MODE_OUTPUT()
```

```
#define GPIO_SET_RST_HIGH() do { pt_set_rst_gpio(1); } while(0)
#define GPIO_SET_RST_LOW() do { pt_set_rst_gpio(0); } while(0)

#define GPIO_SET_INT_MODE_INPUT() \
    do { \
        pt_set_int_gpio_mode(); \
    } while (0)
#define GPIO_READ_INT_STATE() pt_get_irq_state()
#define REGISTER_INT_HANDLER(handler) do
{ add_touch_handler(handler); } while(0)
#define REGISTER_60HZ_HANDLER(handler) do
{ add_touch_poller_60hz(handler); } while(0)

#define DISABLE_IRQ()  do { pt_disable_irq(); } while(0)
#define ENABLE_IRQ() do { pt_enable_irq(); } while(0)
#define GET_SYSTEM_TICK() (pt_get_time_stamp())
```


5.2. Interrupt Handler and Polling Handler

The device should provide a way to register these two kind of handler. Here is an example only, and the developer is free to call the handler directly in their platform by need:

```
#define REGISTER_INT_HANDLER(handler) do
{ add_touch_handler(handler); } while(0)
#define REGISTER_60HZ_HANDLER(handler) do
{ add_touch_poller_60hz(handler); } while(0)

/* Enumeration - To detect and check the state of device */
void pt_init(void)
{
    PT_RC_CHECK rc = RC_PASS;
    u8 force_update = 0;

    DISABLE_IRQ();
    rc = pt_enumeration();
    if (rc == RC_PASS) {
```

```
        PT_DBG("%s: Enumeration is successful\n", __func__);
    } else
        force_update = 1;

    rc = pt_update(force_update);
    if (rc == RC_PASS) {
        PT_DBG("%s: update is successful\n", __func__);
    }
    rc = pt_cmcp_test();
    if (rc == RC_PASS) {
        PT_DBG("%s: cmcp_test is successful\n", __func__);
    }

    cd.debug_level = DEBUG_LEVEL_3;
    REGISTER_INT_HANDLER(pt_parse_cmd_and_interrupt);
    REGISTER_60HZ_HANDLER(pt_poll_noise_metric);
    ENABLE_IRQ();

    PT_DBG("%s: Exit\n", __func__);
}
```