Confidential B

Flashlight Driver Porting Guide base on Kernel-4.4

Flashlight Driver Porting Document

MEDIATEK

After reading the document, you would know followings

- The structure of flashlight driver
- How to port a new flashlight driver IC in kernel-4.4
- Platform & i2c Driver register step
- How to debug

Outline

MEDIATEK

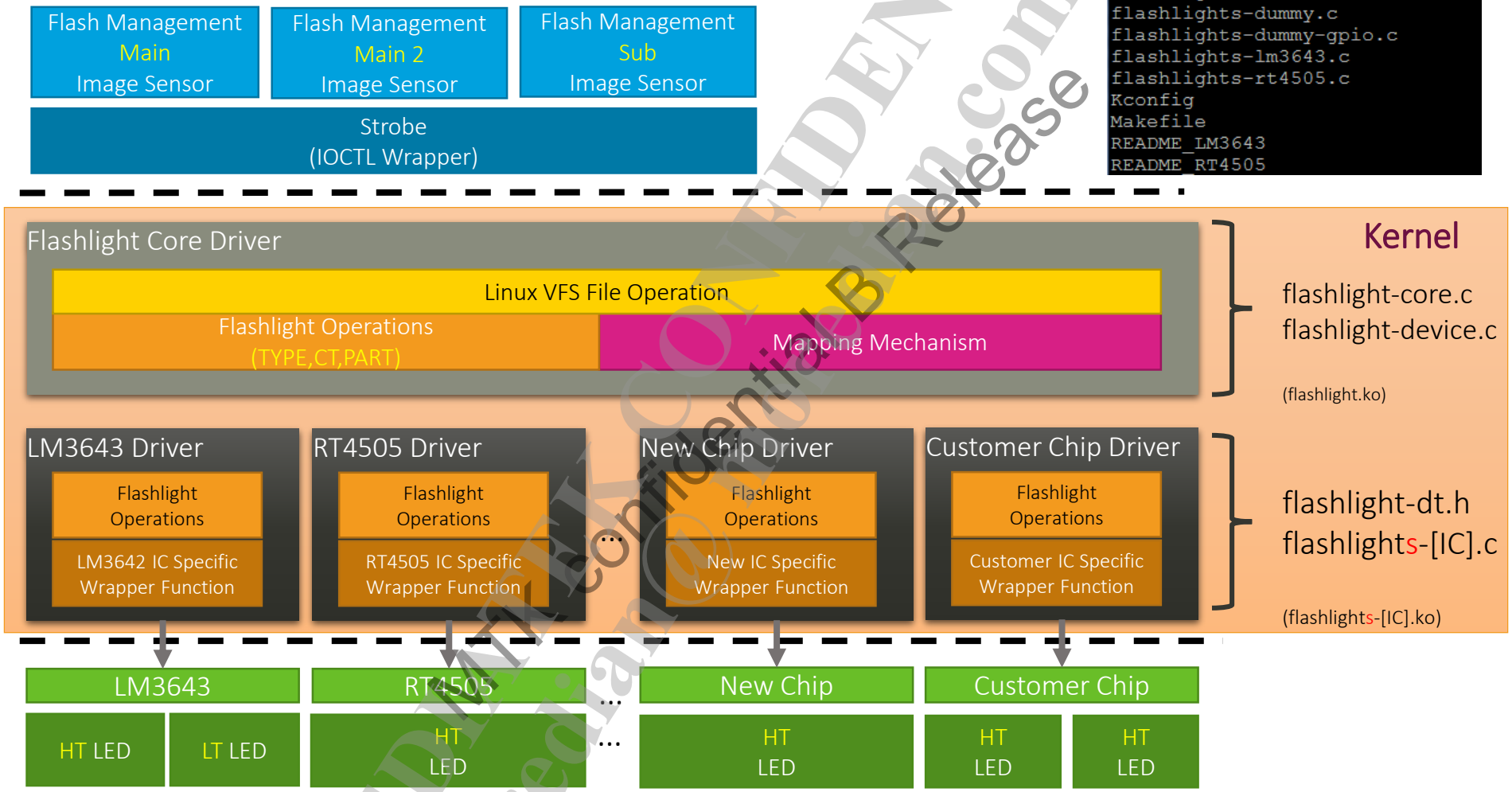
- The structure of flashlight driver
- Porting a New Driver
  - ✓ All files you should need
  - ✓ Add new platform or project in mapping table
  - ✓ Add new platform or project in device tree table
  - ✓ Copy a dummy driver and modify it
  - ✓ Wrapping the IC specific function
  - ✓ Add into kernel source
- How the flashlight driver works
  - ✓ platform driver register
  - ✓ i2c driver register
  - ✓ The meaning of flashlight-device.c
  - ✓ command transform list
  - ✓ Calling flow
- Issure share

The structure of flashlight driver

Flashlight Architecture

[ANDROID]/[KERNEL]/misc/mediatek/flashlight

```
mtk13205@flashlight$ ls
flashlight-core.c
flashlight-device.c
flashlight-dt.h
flashlight.h
flashlights-dummy.c
flashlights-dummy-gpio.c
flashlights-lm3643.c
flashlights-rt4505.c
Kconfig
Makefile
README_LM3643
README_RT4505
```



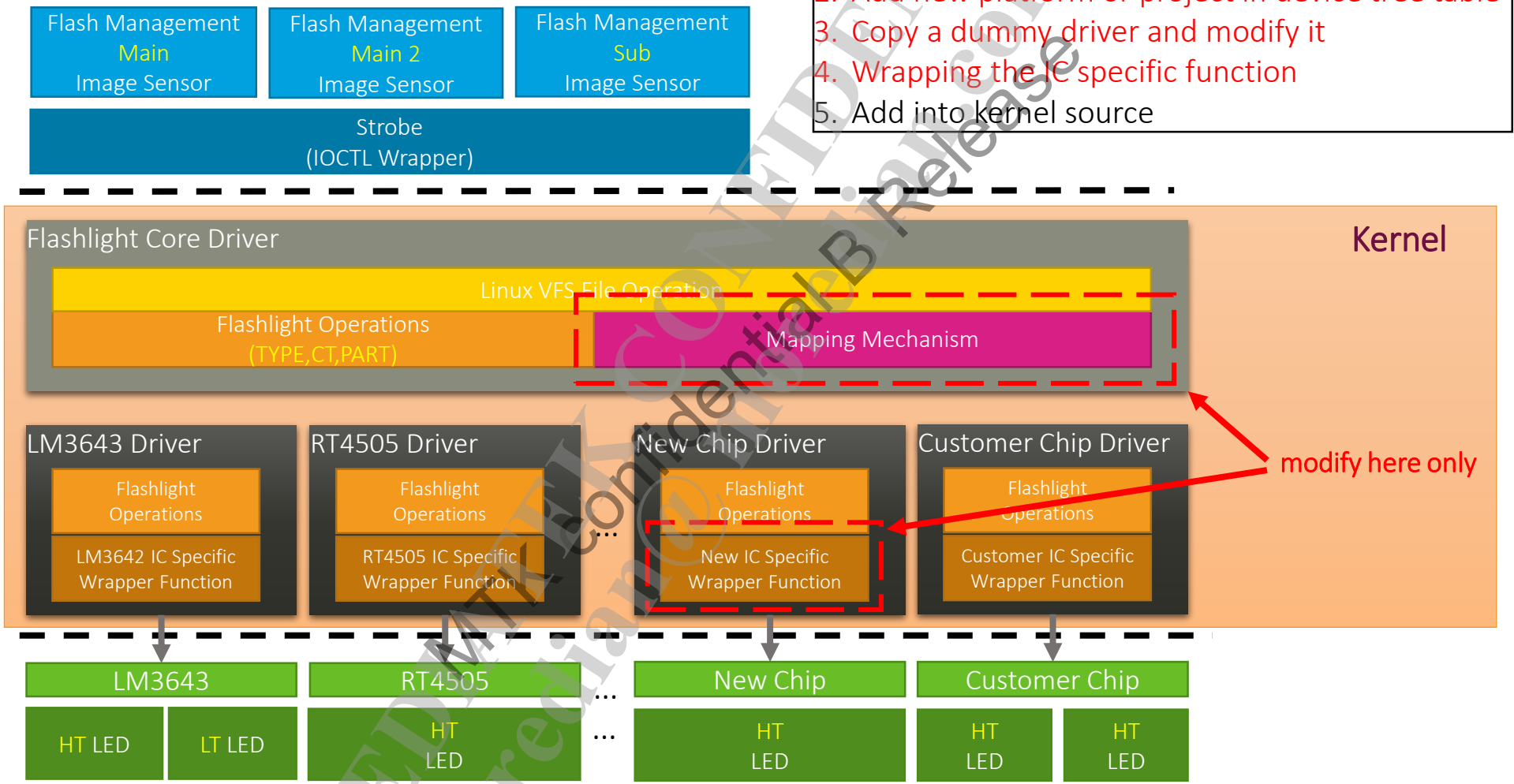
Porting a New Driver

0. All files you should need
1. Add new platform or project in mapping table
2. Add new platform or project in device tree table
3. Copy a dummy driver and modify it
4. Wrapping the IC specific function
5. Add into kernel source

Porting a New Driver

Step:

1. Add new platform or project in mapping table
2. Add new platform or project in device tree table
3. Copy a dummy driver and modify it
4. Wrapping the IC specific function
5. Add into kernel source



Step 0. All files you should need

Flashlight source code and header file:
[ANDROID]/[KERNEL]/driver/misc/mediatek/flashlight

Kernel config and Makefile:
[ANDROID]/[KERNEL]/driver/misc/mediatek/flashlight/Kconfig
[ANDROID]/[KERNEL]/driver/misc/mediatek/flashlight/Makefile
[ANDROID]/[KERNEL]/arch/arm64/configs/[PROJECT]

Device tree:
[ANDROID]/[KERNEL]/arch/arm64/boot/dts/mediatek/[PROJECT].dts
[ANDROID]/[KERNEL]/arch/arm64/boot/dts/mediatek/[PLATFORM].dtsi
[ANDROID]/[KERNEL]/drivers/misc/mediatek/dws/[PLATFORM]/[PROJECT].dws

```
mtk13205@kernel-4.4$ ls drivers/misc/mediatek/flashlight/
flashlight-core.c    flashlights-dummy-gpio.c  Makefile
flashlight-device.c  flashlights-lm3643.c      README_LM3643
flashlight.h         flashlights-rt4505.c      README_RT4505
flashlights-dummy.c  Kconfig
```

```
mtk13205@kernel-4.4$ ls arch/arm64/configs/
defconfig
evb6757_64_6630_debug_defconfig
evb6757_64_6630_defconfig
evb6757_64_debug_defconfig
evb6757_64_defconfig
evb6757_64_dp_debug_defconfig
evb6757_64_dp_defconfig
evb6757_64_mhl_debug_defconfig
evb6757_64_mhl_defconfig
```

```
mtk13205@kernel-4.4$ ls arch/arm64/boot/dts/mediatek/
cust_kpd_8167.dtsi                k57v1_64_om_lwctg_dsds_4sim.dts
cust_mt6757_msdc.dtsi             k57v1_64_om_lwctg.dts
evb6757_64_6630.dts               k57v1_64_om_lwctg_lp.dts
evb6757_64_dp.dts                 k57v1_64_om_lwctg_smt.dts
evb6757_64.dts                    k57v1_64_om_lwg.dts
evb6757_64_144.dts                k57v1_64_om_qhd_ss.dts
```

```
mtk13205@kernel-4.4$ ls drivers/misc/mediatek/dws/mt6757/
evb6757_64_6630.dws  evb6757_144.dws          k57v1_64_teei.dws
evb6757_64_dp.dws    k57_64_om_lwctg_lm.dws   k57v1.dws
evb6757_64.dws       k57v1_64_180.dws         k57v1_tee.dws
evb6757_64_144.dws   k57v1_64_bif.dws         k57v1_teei.dws
evb6757_64_mhl.dws   k57v1_64_codeck.dws      tmp
evb6757_64_tee.dws   k57v1_64.dws
evb6757.dws          k57v1_64_tee.dws
```

Step 1. Add new platform or project in mapping table

[ANDROID]/[KERNEL]/driver/misc/mediatek/flashlight/Makefile

```
ccflags-$(CONFIG_MTK_FLASHLIGHT_DEBUG) += -D$(MTK_PLATFORM) -D$(MTK_PROJECT)
```

Needs to do:

- Define your platform or project
  (Platform or project also available. Makefile will bring in the parameter automatically.)

- Choose the (TYPE,CT,PART).
  - Naming a driver "NAME". (Refer to p.9)
  - Setup "CHANNEL" (if your driver support multi-channel)
  - Setup "DECOUPLE" (if you need to apply multi-channel to different LED set.)

- MUST keep the rest part unchanged.

[ANDROID]/[KERNEL]/driver/misc/mediatek/flashlight/flashlight-device.c

```
#elif defined(mt6799)
const struct flashlight_device_id flashlight_id[] = {
    /* {TYPE, CT, PART, "NAME", CHANNEL, DECOUPLE} */
    {0, 0, 0, "flashlights-mt6336", 0, 0},
    {0, 1, 0, "flashlights-mt6336", 1, 0},
    {1, 0, 0, "flashlights-none", -1, 0},
    {1, 1, 0, "flashlights-none", -1, 0},
    {0, 0, 1, "flashlights-none", -1, 0},
    {0, 1, 1, "flashlights-none", -1, 0},
    {1, 0, 1, "flashlights-none", -1, 0},
    {1, 1, 1, "flashlights-none", -1, 0},
};
#else
```

Note

- **TYPE**: 0(rear LED set), 1(front LED set)
- **CT**: 0(high color temp), 1(low color temp)
- **PART**: HW part, just modify part 0, if you are not sure about this.

Step 2. Add new platform or project in device tree table

Needs to do:

- Modify the compatible string
- Setup the I2C slave address which is defined in IC spec .

[ANDROID]/[KERNEL]/driver/misc/mediatek/flashlight/flashlights-dt.h

```
#define LM3643_DTNAME      "mediatek,flashlights_lm3643"
#define LM3643_DTNAME_I2C  "mediatek,strobe_main"
```

Platform Device

[ANDROID]/[KERNEL]/arch/arm64/boot/dts/mediatek/[PLATFORM].dtsi

```
flashlights_lm3643: flashlights_lm3643 {
    compatible = "mediatek,flashlights_lm3643";
};
```

GPIO Device (attached on platform device)

```
&pio {
    flashlights_lm3643_pins_default: default {
    };

    flashlights_lm3643_pins_hwen_high: hwen_high {
        pins_cmd_dat {
            pins = <PINMUX_GPIO8__FUNC_GPIO8>;
            slew-rate = <1>;
            output-high;
        };
    };

    flashlights_lm3643_pins_hwen_low: hwen_low {
        pins_cmd_dat {
            pins = <PINMUX_GPIO8__FUNC_GPIO8>;
            slew-rate = <1>;
            output-low;
        };
    };
};

&flashlights_lm3643 {
    pinctrl-names = "default", "hwen_high", "hwen_low";
    pinctrl-0 = <&flashlights_lm3643_pins_default>;
    pinctrl-1 = <&flashlights_lm3643_pins_hwen_high>;
    pinctrl-2 = <&flashlights_lm3643_pins_hwen_low>;
    status = "okay";
};
```

I2C Device

Type 1. Define DT directly. (Used in FPGA/Tablet)

[ANDROID]/[KERNEL]/arch/arm64/boot/dts/mediatek/[PROJECT].dts

```
strobe_main@63 {
    compatible = "mediatek,strobe_main";
    reg = <0x63>;
};
```

Type 2. From DCT tool. (Genera used)

[ANDROID]/[KERNEL]/drivers/misc/mediatek/dws/[PLATFORM]/[PROJECT].dws

```
<device6>
    <varName>STROBE_MAIN</varName>
    <channel>I2C_CHANNEL_1</channel>
    <address>0x63</address>
</device6>
```

Automatically generate at build time.

[ANDROID]/out/target/product/[PROJECT]/obj/KERNEL_OBJ/arch/arm64/boot/dts/cust.dtsi

```
strobe_main@63 {
    compatible = "mediatek,strobe_main";
    reg = <0x63>;
};
```

It's a MTK proprietary data format.
It's XML actually.

Step 3. Copy a dummy driver and modify it

Needs to do:

- Rename all “dummy” to “[your_ic]”
- Rename all ”DUMMY” to ”[YOUR_IC]”
- Search string “TODO”, which give you a hint where you need to modify

```
mtk13205@flashlight$ grep -r TODO ./flashlights-dummy.c
/* TODO: modify temp device tree name */
/* TODO: define driver name */
/* TODO: define register */
        /* TODO: wrap enable function */
        /* TODO: wrap disable function */
        /* TODO: wrap set level function */
        /* TODO: wrap init function */
        /* TODO: wrap uninit function */
```

Temp device tree name:
Only used for test.

Driver name:
Used in flashlight mapping table(flashlight-device.c) and kernel device name

Register and wrapper function:
Define the specific IC behavior

Step 4. Wrapping the IC specific function

Needs to do:

- Read the IC spec and port the driver
- Search string "TODO", which give you a hint where you need to modify

[ANDROID]/[KERNEL]/driver/misc/mediatek/flashlight/flashlights-dummy.c

```
/* flashlight init */
int dummy_init(void)
{
    unsigned char reg = 0, val = 0;

    /* TODO: wrap init function */

    return dummy_write_reg(dummy_i2c_client, reg, val);
}

/* flashlight uninit */
int dummy_uninit(void)
{
    unsigned char reg = 0, val = 0;

    /* TODO: wrap uninit function */

    return dummy_write_reg(dummy_i2c_client, reg, val);
}
```

```
/* flashlight enable function */
static int dummy_enable(void)
{
    unsigned char reg = 0, val = 0;

    /* TODO: wrap enable function */

    return dummy_write_reg(dummy_i2c_client, reg, val);
}

/* flashlight disable function */
static int dummy_disable(void)
{
    unsigned char reg = 0, val = 0;

    /* TODO: wrap disable function */

    return dummy_write_reg(dummy_i2c_client, reg, val);
}

/* set flashlight level */
static int dummy_set_level(int level)
{
    unsigned char reg = 0, val = 0;

    /* TODO: wrap set level function */

    return dummy_write_reg(dummy_i2c_client, reg, val);
}
```

Step 5. Add into kernel source

Needs to do:

- Modify Kconfig and Makefile. (These files are required in build procedure.)
- Modify kernel default config which indicated the kernel modules should be compiled.

[ANDROID]/[KERNEL]/driver/misc/mediatek/flashlight/Kconfig

```
config MTK_FLASHLIGHT_RT4505
    tristate "Mediatek flashlight with driver IC (Richtek RT4505)"
    depends on MTK_FLASHLIGHT
    default n
    help
        This is for the flashlight driver IC (Richtek RT4505).

config MTK_FLASHLIGHT_DUMMY
    tristate "Mediatek flashlight dummy driver"
    depends on MTK_FLASHLIGHT
    default n
    help
        This is flashlight dummy driver for camera.
```

[ANDROID]/[KERNEL]/arch/arm64/configs/[PROJECT]

```
CONFIG_MTK_FLASHLIGHT=m
CONFIG_MTK_FLASHLIGHT_LM3643=m
CONFIG_MTK_FLASHLIGHT_RT4505=m
CONFIG_MTK_FLASHLIGHT_DUMMY=m
CONFIG_MTK_FLASHLIGHT_DUMMY_GPIO=m
CONFIG_MTK_FLASHLIGHT_DEBUG=y
```

[ANDROID]/[KERNEL]/driver/misc/mediatek/flashlight/Makefile

```
flashlight-y    := flashlight-core.o
flashlight-y    += flashlight-device.o

obj-$(CONFIG_MTK_FLASHLIGHT)          += flashlight.o
obj-$(CONFIG_MTK_FLASHLIGHT_LM3643)   += flashlights-lm3643.o
obj-$(CONFIG_MTK_FLASHLIGHT_RT4505)   += flashlights-rt4505.o
obj-$(CONFIG_MTK_FLASHLIGHT_DUMMY)    += flashlights-dummy.o
obj-$(CONFIG_MTK_FLASHLIGHT_DUMMY_GPIO) += flashlights-dummy-gpio.o
```

How the flashlight driver works

- platform driver register
- i2c driver register
- The meaning of flashlight-device.c
- command transform list

platform driver register

/kernel-4.4/drivers/misc/mediatek/flashlight/flashlight-core.c

```
module_init(flashlight_init);

static int __init flashlight_init(void)
{
  int ret;

  fl_dbg("Init start.\n");

#ifndef CONFIG_OF
  ret = platform_device_register(&flashlight_platform_device);
  if (ret) {
          fl_err("Failed to register platform device\n");
          return ret;
  }
#endif

  ret = platform_driver_register(&flashlight_platform_driver);
  if (ret) {
          fl_err("Failed to register platform driver\n");
          return ret;
  }

          ......
}
```

```
static struct platform_driver flashlight_platform_driver = {
   .probe = flashlight_probe,
   .remove = flashlight_remove,
   .shutdown = flashlight_shutdown,
   .driver = {
              .name = FLASHLIGHT_DEVNAME,
              .owner = THIS_MODULE,
#ifdef CONFIG_OF
              .of_match_table = flashlight_of_match,
#endif
   },
};

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

Match device tree:
/kernel-4.4/arch/arm64/boot/dts/mediatek/mtxxxx.dtsi

```
flashlight_core: flashlight_core {
        compatible = "mediatek,flashlight_core"
};
```

```
static int flashlight_probe(struct platform_device *dev)
{
     flashlight_class = class_create(THIS_MODULE, FLASHLIGHT_CORE);

     flashlight_device =
         device_create(flashlight_class, NULL, flashlight_devno, NULL, FLASHLIGHT_DEVNAME);

          ......
}
```

i2c driver register

/kernel-4.4/drivers/misc/mediatek/flashlight/flashlights-lm3643.c

```
module_init(flashlight_lm3643_init);

static int __init flashlight_lm3643_init(void)
{
   ret = platform_driver_register(&lm3643_platform_driver);
     ......
}


static struct platform_driver lm3643_platform_driver = {
   .probe = lm3643_probe,
   .remove = lm3643_remove,
   .driver = {
            .name = LM3643_NAME,
            .owner = THIS_MODULE,
#ifdef CONFIG_OF
            .of_match_table = lm3643_of_match,
#endif
   },
};


static const struct of_device_id lm3643_of_match[] = {
   {.compatible = LM3643_DTNAME},
   {},
};

#define LM3643_DTNAME      "mediatek,flashlights_lm3643"
```

Match device tree:
/kernel-4.4/arch/arm64/boot/dts/mediatek/mtxxxx.dtsi

```
flashlights_lm3643: flashlights_lm3643 {
        compatible = "mediatek,flashlights_lm3643";
};
```

```
static int lm3643_probe(struct platform_device *dev)
{
  /* init pinctrl */
  lm3643_pinctrl_init(dev)

  i2c_add_driver(&lm3643_i2c_driver)
}

static struct i2c_driver lm3643_i2c_driver = {
  .driver = {
            .name = LM3643_NAME,
#ifdef CONFIG_OF
            .of_match_table = lm3643_i2c_of_match,
#endif
            },
  .probe = lm3643_i2c_probe,
  .remove = lm3643_i2c_remove,
  .id_table = lm3643_i2c_id,
};

static const struct of_device_id lm3643_i2c_of_match[] = {
  {.compatible = LM3643_DTNAME_I2C},
  {},
};


 #define LM3643_DTNAME_I2C "mediatek,strobe_main"
```

查看 i2c device

```
/sys/class/i2c-adapter/i2c-1/1-0063 # cat name
cat name
strobe_main
```

Match device tree:

I2C Device

Type 1. Define DT directly. (Used in FPGA/Tablet)

[ANDROID]/[KERNEL]/arch/arm64/boot/dts/mediatek/[PROJECT].dts

```
strobe_main@63 {
    compatible = "mediatek,strobe_main";
    reg = <0x63>;
};
```

Type 2. From DCT tool. (Genera used)

[ANDROID]/[KERNEL]/drivers/misc/mediatek/dws/[PLATFORM]/[PROJECT].dws

```
<device6>
    <varName>STROBE_MAIN</varName>
    <channel>I2C_CHANNEL_1</channel>
    <address>0x63</address>
</device6>
```

Automatically generate at build time.

[ANDROID]/out/target/product/[PROJECT]/obj/KERNEL_OBJ/arch/arm64/boot/dts/cust.dtsi

```
strobe_main@63 {
    compatible = "mediatek,strobe_main";
    reg = <0x63>;
};
```

```
static int lm3643_i2c_probe(struct i2c_client *client, const struct i2c_device_id *id)
{
  ......
    /* register flashlight operations */
    if (flashlight_dev_register(LM3643_NAME, &lm3643_ops)) {
            fl_err("Failed to register flashlight device.\n");
            err = -EFAULT;
            goto err_free;
    }
  ......
}
```

```
#define LM3643_NAME "flashlights-lm3643"
```

```
static struct flashlight_operations lm3643_ops = {
  lm3643_open,
  lm3643_release,
  lm3643_ioctl,
  lm3643_strobe_store,
  lm3643_set_driver
};
```

```c
int flashlight_dev_register(const char *name, struct flashlight_operations *dev_ops)
{
  for (i = 0; i < FLASHLIGHT_DEVICE_NUM; i++) {
        if (!strncmp(name, flashlight_id[i].name, FLASHLIGHT_NAME_SIZE)) {
                type_index = flashlight_id[i].type;
                ct_index = flashlight_id[i].ct;
                part_index = flashlight_id[i].part;

                if (flashlight_index_verify(type_index, ct_index, part_index)) {
                        fl_err("Failed to register device (%s)\n", flashlight_id[i].name);
                        continue;
                }

                fl_dbg("%s %d %d %d\n",
                                flashlight_id[i].name, type_index, ct_index, part_index);

                mutex_lock(&fl_mutex);
                fl_ops[type_index][ct_index][part_index] = dev_ops;
                fl_channel[type_index][ct_index][part_index] = flashlight_id[i].channel;
                fl_decouple[type_index][ct_index][part_index] = flashlight_id[i].decouple;
                mutex_unlock(&fl_mutex);
        }
  }
}
```

获取operation
和参数

/kernel-4.4/drivers/misc/mediatek/flashlight/flashlight-device.c

```c
const struct flashlight_device_id flashlight_id[] = {
        /* {TYPE, CT, PART, "NAME", CHANNEL, DECOUPLE} */
        {0, 0, 0, "flashlights-lm3643", 0, 0},
        {0, 1, 0, "flashlights-lm3643", 1, 0},
        {1, 0, 0, "flashlights-none", -1, 0},
        {1, 1, 0, "flashlights-none", -1, 0},
        {0, 0, 1, "flashlights-none", -1, 0},
        {0, 1, 1, "flashlights-none", -1, 0},
        {1, 0, 1, "flashlights-none", -1, 0},
        {1, 1, 1, "flashlights-none", -1, 0},
};
```

The meaning of flashlight-device.c

参数	含义
TYPE	区分main/sub flashlight
CT	区分High temperature/Low temperature
PART	作为driver backup
NAME	此处有定义时，表示有此IC，默认值none
CHANNEL	自定义的参数，通过type, ct拿到channel
DECOUPLE	有些flash driver IC只能同时flash/torch mode，driver需要修改等到两个channel都设置下来才打闪。这时如果需要两个channel只操作1个，就可以使用decouple这个参数。

```
const struct flashlight_device_id flashlight_id[] = {
        /* {TYPE, CT, PART, "NAME", CHANNEL, DECOUPLE} */
        {0, 0, 0, "flashlights-lm3643", 0, 0},
        {0, 1, 0, "flashlights-lm3643", 1, 0},
        {1, 0, 0, "flashlights-none", -1, 0},
        {1, 1, 0, "flashlights-none", -1, 0},
        {0, 0, 1, "flashlights-none", -1, 0},
        {0, 1, 1, "flashlights-none", -1, 0},
        {1, 0, 1, "flashlights-none", -1, 0},
        {1, 1, 1, "flashlights-none", -1, 0},
};
```

配置main/HT

配置main/LT

配置 sub/HT

配置 sub/LT

Driver backup

flashlight_device_id 配置举例

1. 只有main flashlight，单闪

```
const struct flashlight_device_id flashlight_id[] = {
  /* {TYPE, CT, PART, "NAME", CHANNEL, DECOUPLE} */
  {0, 0, 0, "flashlights-lm3643", 0, 0},
  {0, 1, 0, "flashlights-none", -1, 0},
  {1, 0, 0, "flashlights-none", -1, 0},
  {1, 1, 0, "flashlights-none", -1, 0},
  {0, 0, 1, "flashlights-none", -1, 0},
  {0, 1, 1, "flashlights-none", -1, 0},
  {1, 0, 1, "flashlights-none", -1, 0},
  {1, 1, 1, "flashlights-none", -1, 0},
};
```

2. 只有main flashlight，dual flash

```
const struct flashlight_device_id flashlight_id[] = {
  /* {TYPE, CT, PART, "NAME", CHANNEL, DECOUPLE} */
  {0, 0, 0, "flashlights-lm3643", 0, 0},
  {0, 1, 0, "flashlights-lm3643", 1, 0},
  {1, 0, 0, "flashlights-none", -1, 0},
  {1, 1, 0, "flashlights-none", -1, 0},
  {0, 0, 1, "flashlights-none", -1, 0},
  {0, 1, 1, "flashlights-none", -1, 0},
  {1, 0, 1, "flashlights-none", -1, 0},
  {1, 1, 1, "flashlights-none", -1, 0},
};
```

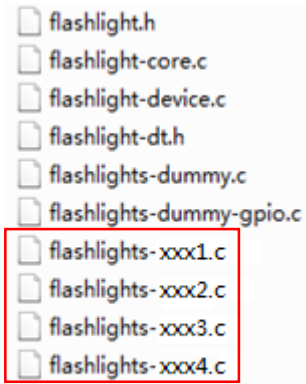
3. main & sub flashlight，都是单闪

```
const struct flashlight_device_id flashlight_id[] = {
  /* {TYPE, CT, PART, "NAME", CHANNEL, DECOUPLE} */
  {0, 0, 0, "flashlights-lm3643", 0, 0},
  {0, 1, 0, "flashlights-none", -1, 0},
  {1, 0, 0, "flashlights-lm3643", 1, 0},
  {1, 1, 0, "flashlights-none", -1, 0},
  {0, 0, 1, "flashlights-none", -1, 0},
  {0, 1, 1, "flashlights-none", -1, 0},
  {1, 0, 1, "flashlights-none", -1, 0},
  {1, 1, 1, "flashlights-none", -1, 0},
};
```

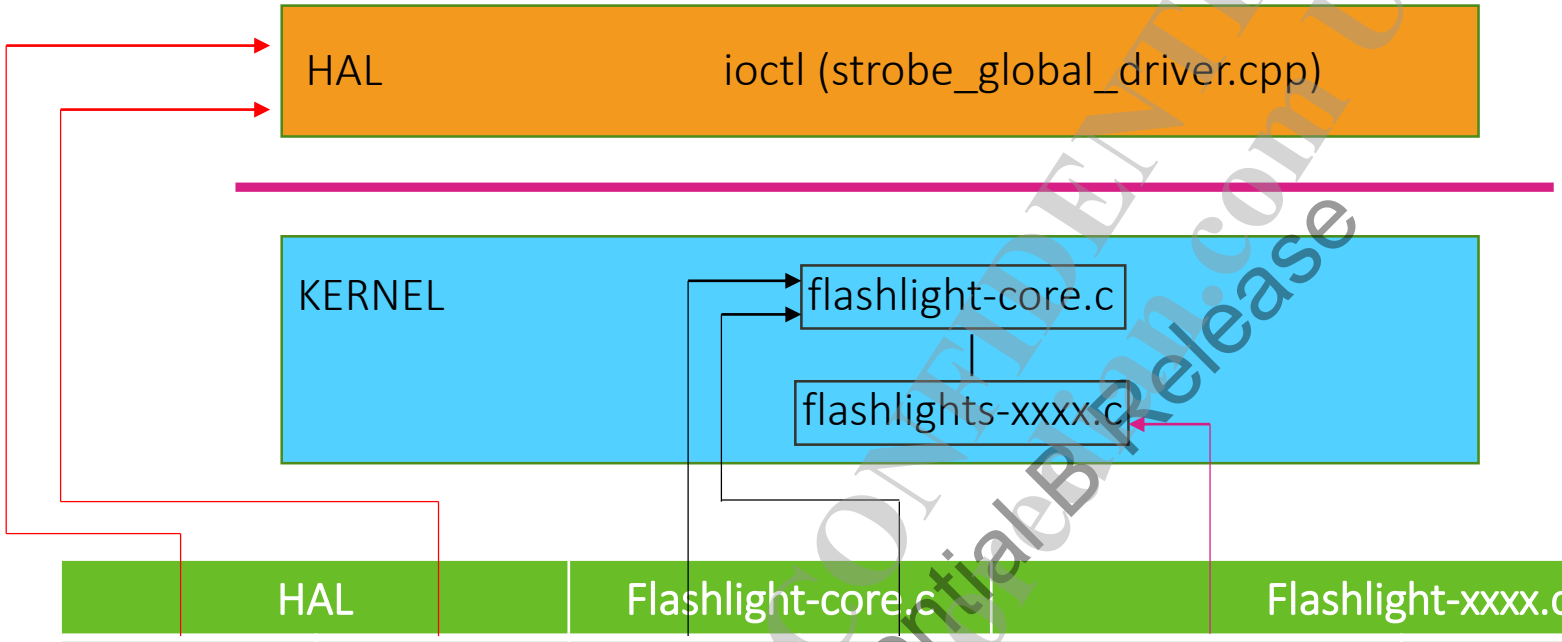
4. main & sub flashlight，都是dual

```
const struct flashlight_device_id flashlight_id[] = {
  /* {TYPE, CT, PART, "NAME", CHANNEL, DECOUPLE} */
  {0, 0, 0, "flashlights-xxx1", 0, 0},
  {0, 1, 0, "flashlights-xxx2", 1, 0},
  {1, 0, 0, "flashlights-xxx3", 2, 0},
  {1, 1, 0, "flashlights-xxx4", 3, 0},
  {0, 0, 1, "flashlights-none", -1, 0},
  {0, 1, 1, "flashlights-none", -1, 0},
  {1, 0, 1, "flashlights-none", -1, 0},
  {1, 1, 1, "flashlights-none", -1, 0},
};
```

新增一颗IC，就多port一支文件
如例4，如果使用4颗不同的IC，那么文件结构如右图所示



command transform list



HAL		Flashlight-core.c		Flashlight-xxxx.c	
SensorDev	StrobeID	type	ct	Channel	decouple
1	1	0	0	flashlight_id[0].channel	flashlight_id[0]. decouple
1	2	0	1	flashlight_id[1].channel	flashlight_id[1]. decouple
2	1	1	0	flashlight_id[2].channel	flashlight_id[2]. decouple
2	2	1	1	flashlight_id[3].channel	flashlight_id[3]. decouple

Control Flow

[name:flashlight&]: [FLASHLIGHT] _flashlight_ioctl: FLASHLIGHTIOC_X_SET_DRIVER(0,0,0): 0

flashlight-core.c

```
static long _flashlight_ioctl(struct file *file, unsigned int cmd, unsigned long arg)

    type_index = flashlight_get_type_index(fl_arg.type_id);
    ct_index = flashlight_get_ct_index(fl_arg.ct_id);
    part_index = flashlight_get_part_index(1);

    fl_dev_arg.channel = fl_channel[type_index][ct_index][part_index];

    switch (cmd) {
            case FLASHLIGHTIOC_X_SET_DRIVER:
                    fl_dbg("FLASHLIGHTIOC_X_SET_DRIVER(%d,%d,%d): %d\n",
                                    type_index, ct_index, part_index, fl_arg.arg);
                    mutex_lock(&fl_mutex);
                    pf = fl_ops[type_index][ct_index][part_index];
                    mutex_unlock(&fl_mutex);
                    if (pf) {
                            pf->flashlight_set_driver();
                            mutex_lock(&fl_mutex);
                            fl_status[type_index][ct_index][part_index] = 1;
                            mutex_unlock(&fl_mutex);
                    } else {
                            fl_info("Failed with no flashlight ops\n");
                            return -EFAULT;
                    }
                    break;
```

[name:flashlight&]: [FLASHLIGHT] _flashlight_ioctl: FLASH_IOC_SET_DUTY(0,0,0)

[name:flashlights_s2mu005&]: [FLASHLIGHT] xxx_ioctl: FLASH_IOC_SET_DUTY(0): 0

```
case FLASH_IOC_SET_DUTY:
        fl_dbg("FLASH_IOC_SET_DUTY(%d,%d,%d)\n",
                        type_index, ct_index, part_index);
        mutex_lock(&fl_mutex);
        ret = fl_set_level(type_index, ct_index, part_index, fl_arg.arg);
        mutex_unlock(&fl_mutex);
        if (ret)
                return -EFAULT;
        break;
```

flashlight-core.c

```
static int fl_set_level(int type_index, int ct_index, int part_index, int level)
{
  struct flashlight_operations *pf;
  struct flashlight_dev_arg fl_dev_arg;

  pf = fl_ops[type_index][ct_index][part_index];
  if (!pf) {
          fl_info("Failed with no flashlight ops\n");
          return -1;
  }

  /* ioctl */
  fl_dev_arg.channel = fl_channel[type_index][ct_index][part_index];
  fl_dev_arg.arg = level;
  if (pf->flashlight_ioctl(FLASH_IOC_SET_DUTY, (unsigned long)&fl_dev_arg)) {
          fl_err("Failed to set level.\n");
          return -1;
  }

  return 0;
}
```

flashlight-lm3643.c

Issue share

Issue Share_1

前闪打闪不亮

4034 01-01 00:10:39.944708 550 550 D [64.502629] (6)[550:cameraserver][name:flashlight&]: [FLASHLIGHT
_flashlight_ioctl: FLASHLIGHTIOC_X_SET_DRIVER(1,0,0): 0
4035 01-01 00:10:39.944718 550 550 I [64.502639] (6)[550:cameraserver][name:flashlight&]: [FLASHLIGHT]
_flashlight_ioctl: Failed with no flashlight ops
5252 01-01 00:10:44.475337 3266 3266 D [69.033258] (2)[3266:3ATHREAD][name:flashlight&]: [FLASHLIGHT
_flashlight_ioctl: FLASH_IOC_SET_ONOFF(1,0,0)
5253 01-01 00:10:44.475352 3266 3266 I [69.033273] (2)[3266:3ATHREAD][name:flashlight&]: [FLASHLIGHT]
_flashlight_ioctl: Failed with no flashlight ops

1) Flashlight-core.c 里面会call到 _flashlight_ioctl

```
static long _flashlight_ioctl(struct file *file, unsigned int cmd, unsigned long arg)

  case FLASHLIGHTIOC_X_SET_DRIVER:
        fl_dbg("FLASHLIGHTIOC_X_SET_DRIVER(%d,%d,%d): %d\n",
                        type_index, ct_index, part_index, fl_arg.arg);
        mutex_lock(&fl_mutex);
        pf = fl_ops[type_index][ct_index][part_index];
        mutex_unlock(&fl_mutex);
        if (pf) {
                pf->flashlight_set_driver();
                mutex_lock(&fl_mutex);
                fl_status[type_index][ct_index][part_index] = 1;
                mutex_unlock(&fl_mutex);
        } else {
                fl_info("Failed with no flashlight ops\n");
                return -EFAULT;
        }
        break;
```

此处 fl_ops[i][j][k] 为空

2) Flashlight-core.c 会找到 fl_ops[i][j][k] 初始化的函数

```
int flashlight_dev_register(const char *name, struct flashlight_operations *dev_ops)
{
    ......
   fl_ops[type_index][ct_index][part_index] = dev_ops;
    ......
}
```

3) Flashlight-xxxx.c, 在这里实际call到flashlight_dev_register去初始化fl_ops[][][]

以LM3643为例

```
/* register flashlight operations */
if (flashlight_dev_register(LM3643_NAME, &lm3643_ops)) {
        fl_err("Failed to register flashlight device.\n");
        err = -EFAULT;
        goto err_free;
}
```

flashlight_sub_pins_default

4) 结合flashlight_dev_register 和 flashlight-device.c的数组，就可以找到初始化失败的原因。一般有如下几种情况：

- name 匹配错误；检查宏定义和flashlight-device.c数组中的name元素。
- flashlight-device.c 配置出错；请参考：The meaning of flashlight-device.c
- i2c 没有match，导致flashlight-xxxx.c中的i2c probe没有执行；请参考：i2c driver register

Issue Share_2

现象：

flashlight是gpio方式控制，

1.flashlights-main 和flashlights-sub DTS配置好后，无法读取节点，查看设备树节点都正常.

2.若只添加任何一个，不同时添加二者，不会出现问题。

Log：

6507 [2.775358] <2>.(2)[1:swapper/0]mediatek-pinctrl 1000b000.pinctrl: missing pins property in node pins_cmd_dat .

Dts 配置如下：

```
&pio{
        flashlight_main_pins_default: default{
        };

&pio{
        flashlight_sub_pins_default: default{
        };
```

pinctrl-0冒号后面的名称不能跟别的名字重复1，之所以出现该问题，是有两个以上的地方都使用default而引起。

Resolve：

将flashlight_sub_pins_default: default 修改为flashlight_sub_pins_default: sub_default

将flashlight_main_pins_default: default 修改为flashlight_main_pins_default: main_default

Q&A

Thanks for Your Time.