旷明 XOS 客制化参考_QM102D

部门	
文档编号	
版 本 号	V0.0.1
作　　者	

更新记录

日期	更新人	版本	备注
2024/12/20	QUA	V0.0.1	初版

目录

1. ADC 按键适配

ADC 驱动介绍

Adc 按键驱动程序用于通过模拟输入设备（ADC）来检测按键输入。此驱动基于 ADC 通道，通过读取电 压值来判断哪个按键被按下，哪个按键被释放。此驱动程序支持自定义按键映射和按键反弹处理

关键代码介绍

```
st->chan_spec = fh_adckey_iio_channel_get();//获取 ADC 通
道 if (IS_ERR(st->chan_spec))
    return PTR_ERR(st->chan_spec);

error = adc_keys_load_keymap(dev, st);//加载按键映射
if (error)
    return error;
```

```
ret = fh_iio_read_channel_raw(st->chan_spec, &value);//读取 ADC 电压
值 if (unlikely(ret < 0)) {
    /* Forcibly release key if any was pressed */
    value                    =
st->keyup_voltage; } else {
    for (i = 0; i < st->num_keys; i++) {//查找最接近的按键
        diff = abs(st->map[i].voltage - value);//比较当前电压值与每个按键的预设电压阈
值
        if  (diff  <  closest)
            { closest = diff;
            keycode                    =
        st->map[i].keycode; }
    }
 }
...
if (abs(st->keyup_voltage - value) < closest)//判断按键是否释放
    keycode = 0;
```

用户配置部分介绍

因内核 linux-4.9 不支持 dts，故改用 board.c 文件进行配置

内核宏

打开内核源码目录

~/linux-4.9/arch/arm/configs/qua_10xv_kidcamera_defconfig

增加 **CONFIG_INPUT_KEYBOARD=y**，**CONFIG_KEYBOARD_ADC=y**。此功能选项用于启动键盘输入子系统和 ADC 按键驱动

平台数据及设备

打开内核源码目录

~/linux-4.9/arch/arm/mach-fh/qm102v/kidcamera/board.c

参考代码如下:

```
struct adc_keys_button adc_buttons[] ={ {
       .voltage = 570,//按键对应的预设电压阈值
       .keycode = KEY_SETUP,//按键按下时要发送的内核键
    码 },
    { .voltage = 900,
       .keycode = KEY_BACK,
    },
 };
static  struct  adc_keys_platform_data  adc_keys_data  =
    { .buttons = adc_buttons,
    .nbuttons = sizeof(adc_buttons) / sizeof(adc_buttons[0]),//按键数量
    .reference_voltage = 1800,//ADC 按键的参考电
压 };
static struct platform_device fh_adc_key_device = {
    .name           = "adc_keys",//平台设备名称，必须与驱动程序中的 driver.name 一致，才
能绑定
    .id         = 0,//平台设备的 id,用于区分同一类型的多个设备
    .dev = {
        .platform_data = &adc_keys_data,
    },
 };
static struct platform_device *qm102v_devices[] __initdata = {
    ...
    &fh_adc_key_device,//注册平台设备到总线
    ...
}
```

工作流程:

在内核启动时， qm102v_board_init 函数会被调用

platform_add_devices 函数会将 qm102v_devices 数组中的所有设备注册到平台总线。

平台总线会比较设备和驱动程序的名称（ name 成员）。如果名称匹配（都是 "adc_keys"），则内核会 调用驱动程序的 probe 函数。

在 probe 函数中，驱动程序可以通过 dev_get_platdata() 函数获取到 adc_keys_data 结构体，从而获 取按键的配置信息。

常见问题 check 及解决

按键电压不正确

问题描述：当用户操作按键时，检测到的初始电压值与硬件原理设计不一致，导致按键功能失效 排查步骤:

- 软件层面检查:

打开 linux 系统，查看目标路径

cat /sys/bus/iio/devices/iio:device0/in_voltage1_raw

该命令会读取 iio:device0 设备（通常是 ADC 设备）的 in_voltage1_raw 属性，该属性表示 ADC 通道 1 的 原始数值。将这个原始数值转换为实际电压值，将计算得到的实际电压值与硬件原理图上设计的按键电 压值进行比较

- 硬件层面检查:

如果软件读取的电压值与预期不符，则需要检查硬件电路是否正常工作。

使用万用表测量芯片引脚的电压。 找到连接到 ADC 输入端的芯片引脚，使用万用表测量该

引脚在不同 按键按下和松开状态下的电压值。将万用表测量的电压值与硬件原理图上设计的电压值进行比较。

- 引脚复用检查:

如果硬件电路检查正常，且软件读取的电压值仍然不正确，则需要检查芯片的引脚复用配置是否正确。 找到引脚复用配置文件。 在 Linux 内核源码中，引脚复用配置通常位于 **~/linux-4.9/arch/arm/mach-fh/qm102v/kidcamera/board.h**

查找对应的 PINOUT 管脚复用寄存器表，验证和修改复用配置。 检查该引脚是否被配置为正确的 ADC 输入功能。如果配置不正确，则需要修改相应的寄存器配置。

2. PinMux 配置

通过配置 PinMux 相关的寄存器，可以修改引脚复用、上下拉、驱动能力。

2.1 QM102D PinMux 配置

QM102D 的 PinMux 支持通过两种方式来设置：第一种时修改 SDK 中提供的 EXCEL 表格来配置，第二种是内核下使用 pinctrl 子系统来管理配置

2.1.1 Excel 配置 PinMux

表格路径：\bsp\ramboot\board\molchip\ddr 目录下的 LT00_PIN.xlsx 表格

2.1.2 Kernel 下 PinMux 配置

Kernel 下 PinMux 的配置位于 dts 文件中，路径为 \bsp\kernel\linux-5.10.y\arch\arm\boot\dts\molchip-kernel-lt00.dtsi
在 client device 中添加 pin 的状态定义

```
device-node-name {
        pinctrl-names = "default";
        pinctrl-0 = <&xxx_pins>;
}
```

pinctrl-0 表示了该设备的一个状态，这里的数字 0 就是 pinctrl-names 中对应的字符串数组 index，pinctrl-0 在这里对应的是 default 状态，xxx_pins 就是驱动具体的 pin 配置属性，需要在 pinctrl 设备节点处定义。

```
pinctrl: pinctrl@10200000 {
          compatible = "molchip,lt00-pinctrl";
          reg = <0x10200000 0x1000>;
          status = "okay";
          …
          pins-are-numbered;
          xxx_pins: xxx_pins {
              pins1 {
                  pinmux = < MCLT00_PIN_3_PAD4_G1__FUNC_SPI0_CSN0>;
                  drive-strength = <MC_DRIVE_8mA>;
                  bias-pull-up = <MC_PUPD_SET_R1R0_10>;
              };
               pins2 {
                    pinmux = < MCLT00_PIN_4_PAD5_G1__FUNC_SPI0_DI>;
                 };
            };
      };
}
```

Pinmux 的选择可以对照 include\dt-bindings\pinctrl 下的 mclt00-pinfunc.h 修改，这个文件中包含所有 SOC pin 的选择功能，drive-strength 与 bias-pull-up 如果不需要做修改可以删除（如

pins1 子节点中的描述）。

Pin 配置的电气特性主要参数如下：

bias-disable: disable any pin bias

bias-pull-up: pull up the pin

bias-pull-down: pull down the pin

drive-strength: sink or source at most X mA

input-schmitt-enable: enable schmitt trigger

slew-rate: set the slew rate

3. Nor FLASH 驱动移植

基于 SPIC 控制器，提供 Flash 器件的移植方法及相关操作。

当选用的 SPI Nor Flash 器件的型号与 SPI Nor Flash 器件兼容性列表中型号不同时，需要在 SPI Nor Flash 驱动的 ID 列表中新增器件 ID 节点以及器件相关的功能函数，主要包括 id 值、时钟、实现并指定器件相关的功能函数接口（4byte 寻址、四线读使能、复位函数等）。

3.1.1 SPI-Nor U-boot2016 下的移植

相关驱动路径 SPL 下的路径

base/soc/qm10xd/linux/bsp/uboot/uboot/drivers/mtd/spi

路径下的相关移植文件

mc_flash.c ：flash 型号的添加入口

如何新增一颗 SPI Nor Flash

以 gd25q128 为例，讲述如何新增一颗 flash

步骤 1：查阅器件手册，新增 id 编号

通过 9FH 命令找到 Flash id 信息

Operation Code	MID7-MID0	ID15-ID8	ID7-ID0
9FH	C8	40	18

获取器件的 chip size、block size 信息

在相应的文件 mc_flash.c 中 spi_nor_ids[]位置添加此款 flash 信息，如下所示：

```
{ "gd25q128", DUMMY_BYTE_0, DUMMY_BYTE_4, SFC_SAMPLE_DELAY , INFO(0xc84018, 0, 32 * 1024, 512,
                        SECT_4K | SPI_NOR_DUAL_READ | SPI_NOR_QUAD_READ |
            SPI_NOR_HAS_LOCK | SPI_NOR_HAS_TB)
            },
```

其中 id、pages_per_sector、sectors_per_block、nr_block 按 datasheet 所述填写即可。

gd25q128 （flash name）,0xc84018 （id）,0 (ext id),32*1024 （sector_size），512 （nr_block）按 datasheet 所述填写即可。

步骤 2：确认 QE 使能设置

使能 QE 需要设置寄存器的特定比特位。这只能通过实际的器件手册来分析。 我们以 GD25Q128 为例讲解一下接口实现。

```
179:        case SNOR_MFR_MX:
180:            params->quad_enable = spi_mx_quad_enable;
181:            break;
182:             case SNOR_MFR_SPANSION:
183:                 params->quad_enable = spi_spansion_quad_enable
184:                 break;
185:             case SNOR_MFR_GIGADEVICE:
186:                 params->quad_enable = spi_giga_quad_enable;
187:                 break;
188:             case SNOR_MFR_EON:
189:                 params->quad_enable = NULL;
190:                 break;
191:        default:
192:            /* Kept only for backward compatibility purpose. */
193:            params->quad_enable = spi_quad_enable;
```

按照读开写使能 -> 写 qe 寄存器 -> 等待完成 -> 确认 qe 是否写 成功的步骤进行即可(因为大部分厂商 QE 使能相同，而有些厂商有些不同，所以可以根据不同厂商的 spec 确定是不是要添加自己的 quad_enable 函数)

3.1.2 SPI-Nor 内核 5.10 下的移植

SPI-Nor 内核下的移植

相关驱动路径

目前内核下驱动路径:

base/soc/qm10xd/linux/bsp/kernel/linux-5.10.y/drivers/mtd/spi-nor

core.c：flash 探测及注册的入口，区分不同厂家的 flash 并进行注册与初始化

如何新增一颗 SPI Nor Flash

在这个目录下找到对应的厂商的文件进行添加，如果没有就建立一个对应的厂商文件。

比如我们用的是 gigadevice 的就在 gigadevice.c 里添加 以 GD25Q128E 为例，讲解一下 id 添加与参数配置，找到 id 表格中 GD 所划分的地方

```
{ "gd25q128", INFO(0xc84018, 0, 64 * 1024, 256,
          SECT_4K | SPI_NOR_DUAL_READ | SPI_NOR_QUAD_READ |
          SPI_NOR_HAS_LOCK | SPI_NOR_HAS_TB) },
```

```
-rw-rw-r-- 1 majiwei majiwei  2761 Mar 22 12:56 atmel.c
-rw-rw-r-- 1 majiwei majiwei   855 Mar 22 12:56 catalyst.c
drwxrwxr-x 2 majiwei majiwei  4096 Mar 22 12:56 controllers
-rwxrwxr-x 1 majiwei majiwei 93941 Mar 22 12:56 core.c
-rwxrwxr-x 1 majiwei majiwei 15496 Mar 22 12:56 core.h
-rwxrwxr-x 1 majiwei majiwei  1954 Mar 22 12:56 dosilicon.c
-rwxrwxr-x 1 majiwei majiwei  1297 Mar 22 12:56 eon.c
-rw-rw-r-- 1 majiwei majiwei   641 Mar 22 12:56 esmt.c
-rw-rw-r-- 1 majiwei majiwei   771 Mar 22 12:56 everspin.c
-rwxrwxr-x 1 majiwei majiwei   548 Mar 22 12:56 fudanmicro.c
-rw-rw-r-- 1 majiwei majiwei   473 Mar 22 12:56 fujitsu.c
-rwxrwxr-x 1 majiwei majiwei  2463 Mar 22 12:56 gigadevice.c
-rw-rw-r-- 1 majiwei majiwei   783 Mar 22 12:56 intel.c
-rw-rw-r-- 1 majiwei majiwei  2786 Mar 22 12:56 issi.c
-rwxrwxr-x 1 majiwei majiwei  1177 Mar 22 12:56 Kconfig
-rw-rw-r-- 1 majiwei majiwei  4068 Mar 22 12:56 macronix.c
-rwxrwxr-x 1 majiwei majiwei   812 Mar 22 12:56 Makefile
-rwxrwxr-x 1 majiwei majiwei 30721 Mar 22 12:56 mc-sfc.c
-rwxrwxr-x 1 majiwei majiwei 11959 Mar 22 12:56 mc-sfc.h
-rw-rw-r-- 1 majiwei majiwei  6152 Mar 22 12:56 micron-st.c
-rw-rw-r-- 1 majiwei majiwei 35770 Mar 22 12:56 sfdp.c
-rw-rw-r-- 1 majiwei majiwei  3764 Mar 22 12:56 sfdp.h
-rwxrwxr-x 1 majiwei majiwei  5060 Mar 22 12:56 spansion.c
-rw-rw-r-- 1 majiwei majiwei  3899 Mar 22 12:56 sst.c
-rwxrwxr-x 1 majiwei majiwei  5731 Mar 22 12:56 winbond.c
-rw-rw-r-- 1 majiwei majiwei  2764 Mar 22 12:56 xilinx.c
-rwxrwxr-x 1 majiwei majiwei   868 Mar 22 12:56 xmc.c
```

其中 INFO 的参数讲解为：

```
#define INFO(_jedec_id, _ext_id, _sector_size, _n_sectors, _flags)    \
        .id = {                                                       \
            ((_jedec_id) >> 16) & 0xff,                               \
            ((_jedec_id) >> 8) & 0xff,                                \
            (_jedec_id) & 0xff,                                       \
            ((_ext_id) >> 8) & 0xff,                                  \
            (_ext_id) & 0xff,                                         \
            },                                                        \
        .id_len = (!(_jedec_id) ? 0 : (3 + ((_ext_id) ? 2 : 0))),     \
        .sector_size = (_sector_size),                                \
        .n_sectors = (_n_sectors),                                    \
        .page_size = 256,                                             \
        .flags = (_flags),
```

按照格式与 Flash datasheet 描述如实填写即可，需要注意的点为： 从 0 开始，
第 5 项的 flag 用来添加是否开始多线读模式，如支持两线或四线读，即添加 SPI_NOR_DUAL_READ|SPI_NOR_QUAD_READ 标志位，否则置 0 即可；
步骤 1：通过 9F 命令查询 ID 通过 flash 器件的 datasheet 找到对应 ID 与相关参数，按照格式填入表格。
步骤 2：确认函数接口 由于 gigadevice.c 下的读写擦函数接口均遵循标准协议，如无特殊器件，无需再核对读写擦 接口，要确认的函数接口主要是 QE 使能接口。
首先确认 QE 使能接口，可以找到

```
static int spi_nor_quad_enable(struct spi_nor *nor)
{
    if (!nor->params->quad_enable)
        return 0;

    if (!(spi_nor_get_protocol_width(nor->read_proto) == 4 ||
        spi_nor_get_protocol_width(nor->write_proto) == 4))
        return 0;

    return nor->params->quad_enable(nor);
}
```

```
static void gd25q256_default_init(struct spi_nor *nor)
{
    /*
     * Some manufacturer like GigaDevice may use different
     * bit to set QE on different memories, so the MFR can't
     * indicate the quad_enable method for this case, we need
     * to set it in the default_init fixup hook.
     */
    nor->params->quad_enable = spi_nor_sr1_bit6_quad_enable;
}
```

S15	S14	S13	S12	S11	S10	S9	S8
SUS1	CMP	LB3	LB2	LB1	SUS2	QE	SRP1

S7	S6	S5	S4	S3	S2	S1	S0
SRP0	BP4	BP3	BP2	BP1	BP0	WEL	WIP

针对不同的型号，QE 使能接口并不一致，如需添加厂 Flash 器件的 quad_enabel，可以在对应文件文件内添加。

4. QM10XD 下 SPI nand Flash 的配置

4.1SPI-Nand U-boot2016 下的移植

相关驱动路径

SPL 下的路径：

base/soc/qm10xd/linux/bsp/uboot/uboot/drivers/mtd/nand 路径下的相关移植文件

nand_ids.c ：flash 型号的添加入口

如何新增一颗 SPI NAND Flash

首先进入 nand_ids.c，然后查看 nand_flash_ids 表各成员，定义在 nand.h

```
struct nand_flash_dev {
    char *name;
    union {
        struct {
            uint8_t mfr_id;
            uint8_t dev_id;
        };
        uint8_t id[NAND_MAX_ID_LEN];
    };
    unsigned int pagesize;
    unsigned int chipsize;
    unsigned int erasesize;
    unsigned int options;
    uint16_t id_len;
    uint16_t oobsize;
    struct {
        uint16_t strength_ds;
        uint16_t step_ds;
    } ecc;
    int onfi_timing_mode_default;
} ? end nand_flash_dev ? ;
```

可以看到表中实际填写时的参数，

我们以 XT26G01C 为例添加一款 NandFlash

```
{"XT26G01CWSIG 3.3V SPI",
    { .id = {0x0B, 0x11 } },
      SZ_2K, SZ_128, SZ_128K, 0, 2, 64, NAND_ECC_INFO(8, SZ_1K),
    0 },
```

其中 id 通过 datasheet 中查询 9FH 命令找到为 0x0B11

Address	Value	Description
Byte 0	0BH	Manufacture ID (XTX)
Byte 1	11H	Device ID (SPI NAND 3.3V 1Gbit)

pagesize,oobsize,chipsize,erasesize,: 容量大小通过查询 array organization 找到

Each device has	Each block has	Each page has	Unit
128M + 8M	128K + 8K	2K + 128	bytes
1024 x 64	64	-	Pages
1024	-	-	Blocks

因为 ECCINFO 用的是 SOC 的 ecc，因此 ECCINFO 统一设置为 NAND_ECC_INFO(8, SZ_1K),

4.2 SPI-Nand 内核 5.10 下的移植

SPI Nand Flash 内核下的移植相关驱动路径

SPI Nand 驱动的路径：

base/soc/qm10xd/linux/bsp/kernel/linux-5.10.y/drivers/mtd/nand/raw/nand_ids.c

路径下与移植相关的文件

nand_ids.c:SPI Nand 器件 ID 表，移植 SPI Nand 器件主要修改的文件

如何新增一颗 SPI Nand Flash

首先进入 nand_ids.c 文件中，然后查看 id 表各成员，定义在

base/soc/qm10xd/linux/bsp/kernel/linux-5.10.y/include/linux/mtd/rawnand.h 中

```
+       {"XT26G01CWSIG 3.3V SPI",
+                { .id = {0x0B, 0x11 } },
+                  SZ_2K, SZ_128, SZ_128K, 0, 2, 64, NAND_ECC_INFO(8, SZ_1K)},
```

```
struct nand_flash_dev {
        char *name;
        union {
                struct {
                        uint8_t mfr_id;
                        uint8_t dev_id;
                };
                uint8_t id[NAND_MAX_ID_LEN];
        };
        unsigned int pagesize;
        unsigned int chipsize;
        unsigned int erasesize;
        unsigned int options;
        uint16_t id_len;
        uint16_t oobsize;
        struct {
                uint16_t strength_ds;
                uint16_t step_ds;
        } ecc;
};
```

Name XT26G01CWSIG,mfr_id 0x0b,dev_id 0x11,page size SZ_2K,chipsize SZ_128,erasesize SZ_128K,ECCINFO 为 NAND_ECC_INFO(8, SZ_1K)，因为用的是 soc 的 ecc，固定为 NAND_ECC_INFO(8, SZ_1K)

5.LCD 驱动接口使用

参考旷明《XOS LCD 驱动接口使用说明.doc》

6.WiFI 设备更换指导

参考《旷明 XOS wifi 设备更换指南.doc》