T710充电 设备树：

charger-manager {

compatible = "charger-manager";

cm-name = "battery";

cm-poll-mode = <2>;//轮询模式

cm-poll-interval = <15000>;//轮询时间间隔

cm-battery-stat = <2>;//电池在位检测方法：2：电压法

cm-fullbatt-vchkdrop-ms = <30000>;//满电后的 检查复充条件的周期

cm-fullbatt-vchkdrop-volt = <60000>;//复充的电压条件

cm-fullbatt-voltage = <4200000>; //满电电压检测

cm-first-fullbatt-current = <150000>; //满电的电流第一次检测

cm-fullbatt-current = <120000>;//满电的电流检测

cm-fullbatt-capacity = <100>;//满电显示100%

cm-num-chargers = <1>;//充电ic数量

cm-num-fast-chargers = <1>;//快充ic数量

cm-chargers = "sc2703\_charger"; //充电器名称

cm-fast-chargers = "sc2730\_fast\_charger"; //快充充电ic名字

cm-fuel-gauge = "sc27xx-fgu";//电量计名字

/\* in deci centigrade \*/

cm-battery-cold = <200>;//电池低温阈值

cm-battery-cold-in-minus;//负温标志

cm-battery-hot = <800>;//电池高温阈值

cm-battery-temp-diff = <100>; //修正异常时cold和overhead阈值

/\* Allow charging for 10hr \*/

cm-charging-max = <36000000>; //允许连续充电的最长时长10小时

/\* recovery charging after stop charging 3hr \*/

cm-discharging-max = <10800000>;//停止充电后 插着充电器 允许复充的时间

/\* the interval to feed charger watchdog \*/

cm-wdt-interval = <60>; //喂看门狗的周期

/\* drop voltage in microVolts to allow shutdown \*/

cm-shutdown-voltage = <3150000>; //电压降到3150mv 后关机

/\* when 99% of the time is exceeded, it will be forced to 100% \*/

cm-tickle-time-out = <1500>;//电量维持在99% 15分钟后跳到100%

/\* how much time to allow capacity change \*/

cm-one-cap-time = <30>;//电量增加1%的 最快时间

/\* when the safe charging voltage is exceeded, stop charging \*/

cm-charge-voltage-max = <6500000>;//充电器的过压保护阈值

/\* drop voltage in microVolts to restart charging \*/

cm-charge-voltage-drop = <700000>;

/\* when the safe charging voltage is exceeded, stop charging \*/

cm-fast-charge-voltage-max = <10500000>;

/\* drop voltage in microVolts to restart charging \*/

cm-fast-charge-voltage-drop = <700000>;//电压drop 700mv重新充电

cm-dcp-jeita-temp-table = <1000 1030 0 4400000>, <1150 1180 700000 4400000>,

<1450 1420 2000000 4400000>, <1600 1570 700000 4100000>;

cm-sdp-jeita-temp-table = <1000 1030 0 4400000>, <1150 1180 500000 4400000>,

<1450 1420 500000 4400000>, <1600 1570 500000 4100000>;

cm-cdp-jeita-temp-table = <1000 1030 0 4400000>, <1150 1180 700000 4400000>,

<1450 1420 1500000 4400000>, <1600 1570 500000 4100000>;

cm-unknown-jeita-temp-table = <1000 1030 0 4400000>, <1150 1180 500000 4400000>,

<1450 1420 500000 4400000>, <1600 1570 500000 4100000>;

cm-fchg-jeita-temp-table = <1000 1030 0 4400000>, <1150 1180 700000 4400000>,

<1450 1420 3250000 4400000>, <1600 1570 700000 4100000>;

monitored-battery = <&bat>;//电池参数信息

cm-capacity-track;

regulator@0 {

cm-regulator-name = "vddgen0";

cable@0 {

cm-cable-name = "USB";

extcon = <&pmic\_typec>;

};

};

};

1：

1：读取电池温度

Sc27xx\_fuel\_gauge.c (drivers\power\supply) 48190 2022/2/28

sc27xx\_fgu\_get\_property

--->case POWER\_SUPPLY\_PROP\_TEMP:

--->ret = sc27xx\_fgu\_get\_temp(data, &value);

--->ret = iio\_read\_channel\_processed(data->channel, &vol);//有俩个adc通道iio\_channel \*channel 和iio\_channel \*charge\_cha

--->regmap\_read(data->regmap, data->base + SC27XX\_FGU\_CURRENT, &cur);//读取通道0的adc

--->sc27xx\_fgu\_get\_current(data, &bat\_current);//读取电流

--->regmap\_read(data->regmap, data->base + SC27XX\_FGU\_CURRENT, &cur);

/\*

\* It is ADC values reading from registers which need to convert to

\* corresponding current values.

\*/

--->\*val = sc27xx\_fgu\_adc\_to\_current(data, cur - SC27XX\_FGU\_CUR\_BASIC\_ADC);//adc转换为电流

--->temp = (s64)adc \* 1000 + data->cur\_1000ma\_adc / 2;

--->div\_s64(temp, data->cur\_1000ma\_adc);

获取通道:电池温度adc：data->channel = devm\_iio\_channel\_get(&pdev->dev, "bat-temp");充电电压adc data->charge\_cha = devm\_iio\_channel\_get(&pdev->dev, "charge-vol");

Adc通道:Sc2730.dtsi (arch\arm64\boot\dts\sprd) 11155 2022/2/28

pmic\_fgu: fgu@c00 {

compatible = "sprd,sc27xx-fgu", "sprd,sc2730-fgu";

reg = <0xc00>;

bat-detect-gpio = <&pmic\_eic 3 GPIO\_ACTIVE\_HIGH>;

nvmem-cell-names = "fgu\_calib";

nvmem-cells = <&fgu\_calib>;

io-channels = <&pmic\_adc 0>, <&pmic\_adc 14>;

io-channel-names = "bat-temp", "charge-vol";

interrupt-parent = <&sc2730\_pmic>;

interrupts = <3 IRQ\_TYPE\_LEVEL\_HIGH>;

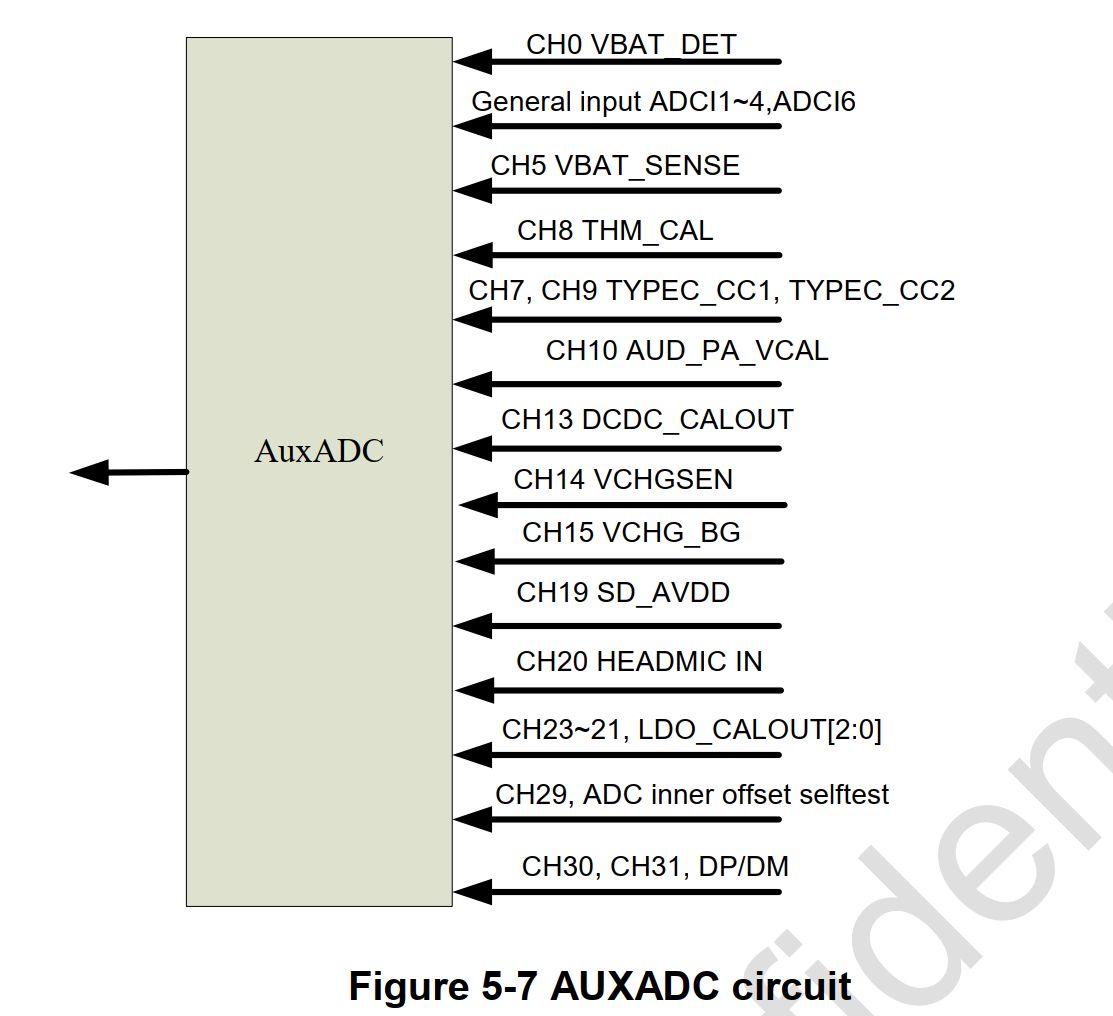
};

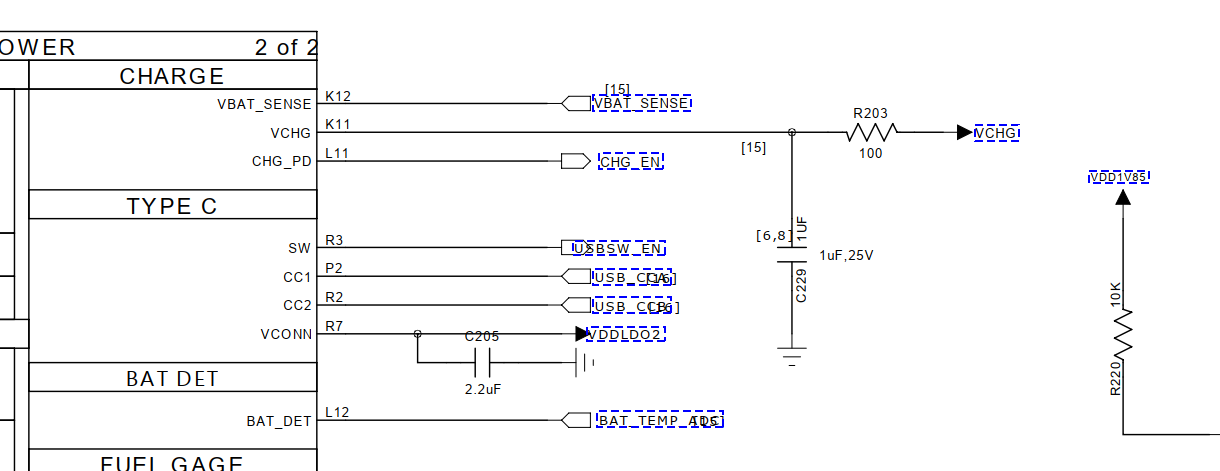
电池温度adc ---> bat-temp ---> pmic\_adc 0通道

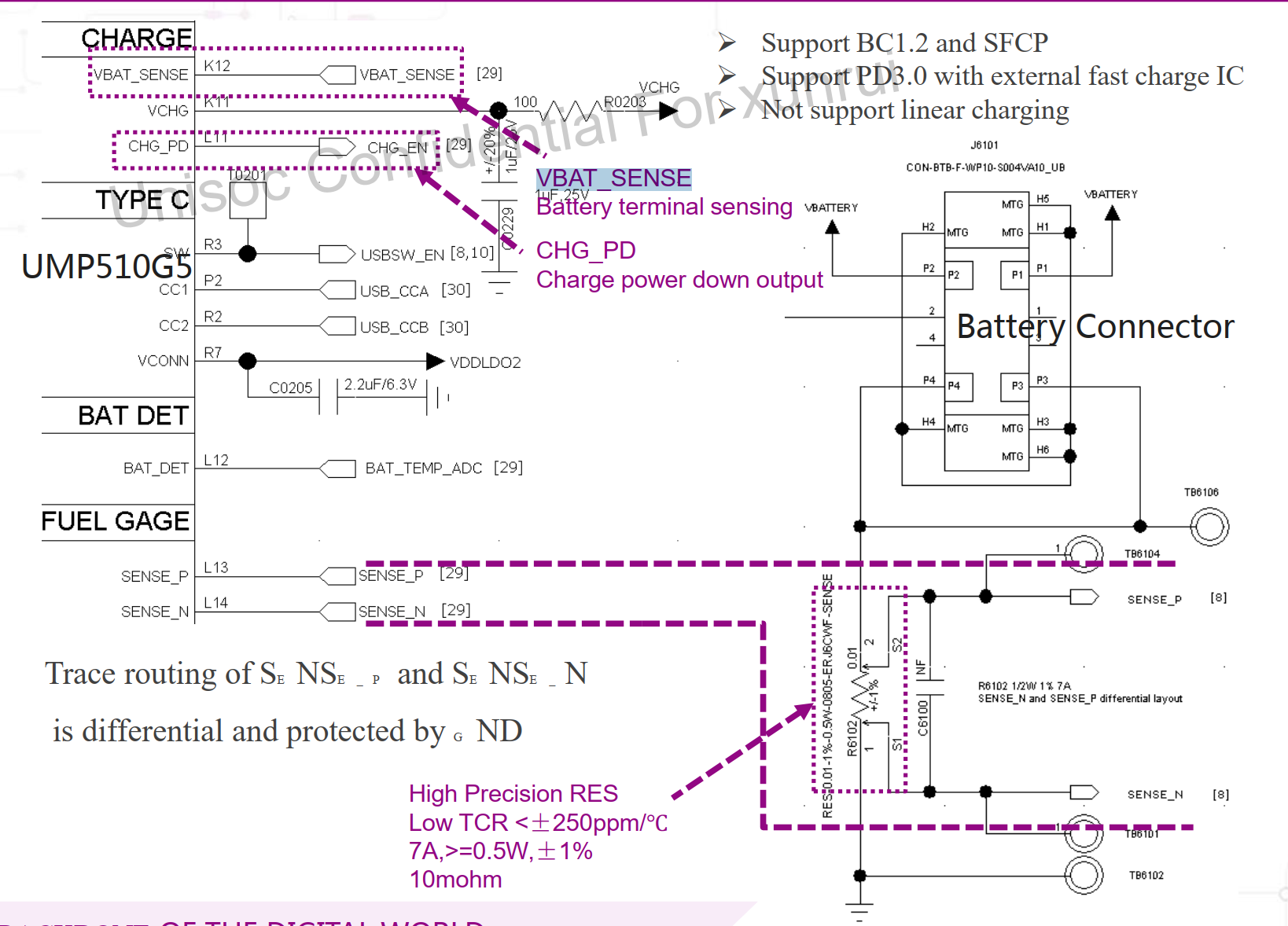
充电电压adc ---> charge-vol ---> pmic\_adc 14通道

adc通道32个，26个用于内部的状态监控，如电池电压，充电电压等，另外6个用于外部使用

UMP510G芯片







--->resistance\_vol = bat\_current \* data->comp\_resistance / 1000;

--->vol = vol - resistance\_vol + (resistance\_vol \*(vol - resistance\_vol)) / (1850 - resistance\_vol);

电压的计算：因为ntc电阻是连接到库仑计的内阻上，且板地阻抗是1850mv,所以需要补偿库伦电阻对地阻抗的电压Loss

/\*

\* Due to the ntc resistor is connected to the coulomb counter

\* internal resistance and the board ground impedance at 1850mv.

\* so need to compensate for coulomb resistance and voltage loss

\* to ground impedance.

\* Follow the formula below:

\* formula:

\* Vadc = Vresistance + (1850 - Vresistance) \* R / 47k + R

\* ->

\* UR = Vadc -Vresistance +

\* Vresistance \* (Vadc - Vresistance) / (1850 - Vresistance)

\*/

--->\*temp = sc27xx\_fgu\_vol\_to\_temp(data->temp\_table,

data->temp\_table\_len,

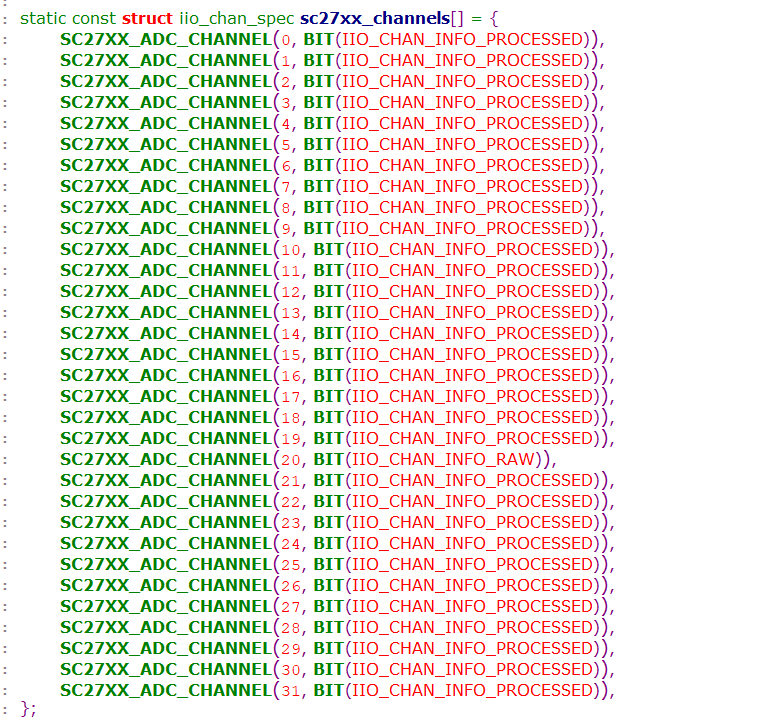
vol \* 1000);

--->\*temp = sc27xx\_fgu\_get\_average\_temp(data, \*temp);

--->data->bat\_temp = \*temp;

Sc27xx\_adc.c (drivers\iio\adc) 20793 2022/2/28

--->一共32个adc通道



通过iio方式注册

indio\_dev = devm\_iio\_device\_alloc(&pdev->dev, sizeof(\*sc27xx\_data));

sc2703-charger sc2703-charger: battery present = 1, charger type = 0 //充电电流为2A

enum usb\_charger\_type {

    UNKNOWN\_TYPE,

    SDP\_TYPE,

    DCP\_TYPE,

    CDP\_TYPE,

    ACA\_TYPE,

};

各种充电类型的电流设置

1:设备树

./ud710-2h10.dts +172

--->capacity-temp-table = <45 100>, <25 100>, <10 97>, <0 95>, <(-10) 82>, <(-20) 62>;

resistance-temp-table = <45 100>, <25 100>, <10 483>, <0 680>, <(-10) 789>, <(-20) 816>;

charge-sdp-current-microamp = <500000 500000>;//sdp充电电流500MA 对应的是pc电脑,D+ D-具有15欧下拉电阻，可以数据通信

charge-dcp-current-microamp = <2000000 3000000>;//dcp 2A d+ d-短路

charge-cdp-current-microamp = <1500000 1500000>;//cdp 1.5A D+ D-具有15欧下拉电阻，可以数据通信 快充模式

charge-unknown-current-microamp = <500000 500000>;//unknown 500MA

charge-fchg-current-microamp = <3000000 2000000>;

● 标准下行端口（SDP）

这种端口的D+和D-线上具有15kΩ下拉电阻。限流值为：挂起时2.5mA，

连接时为100mA，连接并配置为较高功率时为500mA。

它其实就是一种普通的USB模式，当USB处于这种模式时

既可以为外部设备（手机充电、充电宝）充电，

也可以起到数据连接的作用（U盘、手机上传/下载）。

● 专用充电端口（DCP）

这种端口不支持任何数据传输，但能够提供1.5A以上的电流。

端口的D+和D-线之间短路。这种类型的端口支持较高充电能

力的墙上充电器和车载充电器，无需枚举。它其实就是简单

的充电器，当USB处于这种模式时只能进行充电而不能进行数据连接。

● 充电下行端口（CDP）

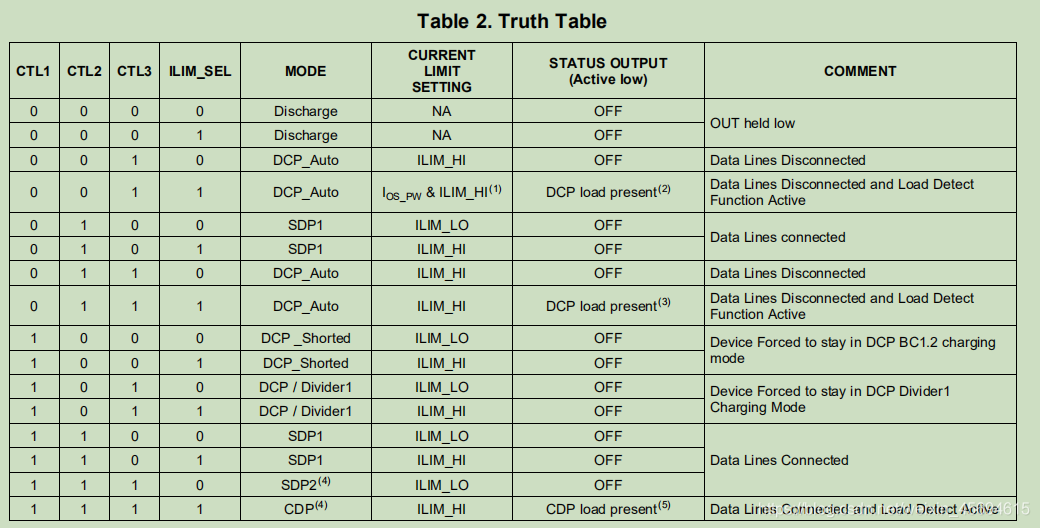
这种端口既支持大电流充电，也支持完全兼容USB 2.0的数据传输。

端口具有D+和D-通信所必需的15kΩ下拉电阻，也具有充电器检测阶

段切换的内部电路。内部电路允许便携设备将CDP与其它类型端口

区分开来。它其实就是带有快充功能（1.5A）的USB接口，当USB

处于这种模式时既可以进行快充，也可以起到数据连接的作用。



获取设备树

Power\_supply\_core.c (drivers\power\supply) 37232 2022/2/28

power\_supply\_get\_battery\_info

--->of\_property\_read\_u32\_index(battery\_np, "charge-sdp-current-microamp", 0,

&info->cur.sdp\_cur);

of\_property\_read\_u32\_index(battery\_np, "charge-sdp-current-microamp", 1,

&info->cur.sdp\_limit);

of\_property\_read\_u32\_index(battery\_np, "charge-dcp-current-microamp", 0,

&info->cur.dcp\_cur);

of\_property\_read\_u32\_index(battery\_np, "charge-dcp-current-microamp", 1,

&info->cur.dcp\_limit);

of\_property\_read\_u32\_index(battery\_np, "charge-cdp-current-microamp", 0,

&info->cur.cdp\_cur);

of\_property\_read\_u32\_index(battery\_np, "charge-cdp-current-microamp", 1,

&info->cur.cdp\_limit);

of\_property\_read\_u32\_index(battery\_np, "charge-aca-current-microamp", 0,

&info->cur.aca\_cur);

of\_property\_read\_u32\_index(battery\_np, "charge-aca-current-microamp", 1,

&info->cur.aca\_limit);

of\_property\_read\_u32\_index(battery\_np, "charge-unknown-current-microamp", 0,

&info->cur.unknown\_cur);

of\_property\_read\_u32\_index(battery\_np, "charge-unknown-current-microamp", 1,

&info->cur.unknown\_limit);

of\_property\_read\_u32\_index(battery\_np, "charge-fchg-current-microamp", 0,

&info->cur.fchg\_cur);

of\_property\_read\_u32\_index(battery\_np, "charge-fchg-current-microamp", 1,

&info->cur.fchg\_limit);

查看充电器的状态

查看输入电流

Cat /sys/class/power\_supply/sc2703\_charger/input\_current\_now

查看输入充电类型

Cat /sys/class/power\_supply/sc2703\_charger/charge\_type

充电流程：

Sc2703-charger.c (drivers\power\supply) 47867 2022/2/28

sc2703\_charger\_work

1:判断电池是否在位

--->present = sc2703\_charger\_is\_bat\_present(info);

2:根据充电器类型设置充电电流和和限制电流

--->switch (info->usb\_phy->chg\_type) {

case SDP\_TYPE:

limit\_cur = info->cur.sdp\_limit;

cur = info->cur.sdp\_cur;

break;

case DCP\_TYPE:

limit\_cur = info->cur.dcp\_limit;

cur = info->cur.dcp\_cur;

break;

case CDP\_TYPE:

limit\_cur = info->cur.cdp\_limit;

cur = info->cur.cdp\_cur;

break;

default:

limit\_cur = info->cur.unknown\_limit;

cur = info->cur.unknown\_cur;

}

3:设置限制电流

--->sc2703\_charger\_set\_limit\_current(info, limit\_

4:设置充电电流

--->sc2703\_charger\_set\_current(info, cur);

5:启动充电

--->sc2703\_charger\_start\_charge(info);

Vbus充电通知

obj-$(CONFIG\_SPRD\_ROC1\_USB2\_PHY) += phy-sprd-roc1.o

obj-$(CONFIG\_SPRD\_USB3\_PHY) += phy-sprd-usb3.o

Phy-sprd-roc1.c (drivers\usb\phy) 16369 2022/2/28

--->

hsphy的充电通知和检测

static int sprd\_hsphy\_vbus\_notify(struct notifier\_block \*nb,

unsigned long event, void \*data)

{

struct usb\_phy \*usb\_phy = container\_of(nb, struct usb\_phy, vbus\_nb);

struct sprd\_hsphy \*phy = container\_of(usb\_phy, struct sprd\_hsphy, phy);

dev\_info(usb\_phy->dev, "[%s]enter phy->is\_host(%d) event(%ld)\n", \_\_func\_\_, (int)phy->is\_host, event);

if (phy->is\_host) {

dev\_info(phy->dev, "USB PHY is host mode\n");

return 0;

}

if (event)

usb\_phy\_set\_charger\_state(usb\_phy, USB\_CHARGER\_PRESENT);

else

usb\_phy\_set\_charger\_state(usb\_phy, USB\_CHARGER\_ABSENT);

return 0;

}

ssphy充电通知和检测

Phy-sprd-usb3.c (drivers\usb\phy) 21350 2022/2/28

static int sprd\_ssphy\_vbus\_notify(struct notifier\_block \*nb,

unsigned long event, void \*data)

{

struct usb\_phy \*usb\_phy = container\_of(nb, struct usb\_phy, vbus\_nb);

struct sprd\_ssphy \*phy = container\_of(usb\_phy, struct sprd\_ssphy, phy);

dev\_info(usb\_phy->dev, "[%s]enter phy->is\_host(%d) event(%ld)\n", \_\_func\_\_, (int)phy->is\_host, event);

if (sprd\_usbm\_event\_is\_active()) {

dev\_info(usb\_phy->dev, "is\_active\n");

return 0;

}

if (phy->is\_host) {

dev\_info(usb\_phy->dev, "USB PHY is host mode\n");

return 0;

}

if (event) {

usb\_phy\_set\_charger\_state(usb\_phy, USB\_CHARGER\_PRESENT);

} else {

u32 msk = MASK\_IPA\_AHB\_UTMISRP\_BVALID\_REG |

MASK\_IPA\_AHB\_OTG\_VBUS\_VALID\_PHYREG;

/\* dwc3 vbus invalid \*/

if (atomic\_read(&phy->inited))

regmap\_update\_bits(phy->ipa\_ahb, REG\_IPA\_AHB\_USB\_CTL0,

msk, 0);

usb\_phy\_set\_charger\_state(usb\_phy, USB\_CHARGER\_ABSENT);

}

return 0;

}

充电器检测

Phy-sprd-roc1.c (drivers\usb\phy) 16369 2022/2/28

static enum usb\_charger\_type sprd\_hsphy\_charger\_detect(struct usb\_phy \*x)

{

struct sprd\_hsphy \*phy = container\_of(x, struct sprd\_hsphy, phy);

if (!phy->pmic)

return UNKNOWN\_TYPE;

return sc27xx\_charger\_detect(phy->pmic);//返回充电类型

--->

}

Ssphy充电器检测

Phy-sprd-usb3.c (drivers\usb\phy) 21350 2022/2/28

static enum usb\_charger\_type sprd\_ssphy\_charger\_detect(struct usb\_phy \*x)

{

struct sprd\_ssphy \*phy = container\_of(x, struct sprd\_ssphy, phy);

if (!phy->pmic)

return UNKNOWN\_TYPE;

return sc27xx\_charger\_detect(phy->pmic);

sc2730-usb-charger.h (include\linux\power) 950 2022/2/28

--->static enum usb\_charger\_type sc27xx\_charger\_detect(struct regmap \*regmap)

{

enum usb\_charger\_type type;

u32 status = 0, val;

int ret, cnt = 10;

do {

ret = regmap\_read(regmap, SC2730\_CHARGE\_STATUS, &val);

if (ret)

return UNKNOWN\_TYPE;

if (val & BIT\_CHG\_DET\_DONE) {

status = val & (BIT\_CDP\_INT | BIT\_DCP\_INT | BIT\_SDP\_INT);

break;

}

msleep(200);

} while (--cnt > 0);

switch (status) {

case BIT\_CDP\_INT:

type = CDP\_TYPE;

break;

case BIT\_DCP\_INT:

type = DCP\_TYPE;

break;

case BIT\_SDP\_INT:

type = SDP\_TYPE;

break;

default:

type = UNKNOWN\_TYPE;

}

return type;

}

}

sc2730-usb-charger.h (include\linux\power) 950 2022/2/28

充电模式检测

1:先设置充电检测函数

Phy-sprd-orca.c (drivers\usb\phy) 17254 2022/2/28

sprd\_ssphy\_probe

--->phy->phy.charger\_detect = sprd\_ssphy\_charger\_detect;

Phy-sprd-usb3.c (drivers\usb\phy) 21350 2022/2/28

phy->phy.charger\_detect = sprd\_ssphy\_charger\_detect;

2：初始化充电的工作队列

Phy.c (drivers\usb\phy) 22147 2022/2/28

static void usb\_charger\_init(struct usb\_phy \*usb\_phy)

{

usb\_phy->chg\_type = UNKNOWN\_TYPE;

usb\_phy->chg\_state = USB\_CHARGER\_DEFAULT;

usb\_phy\_set\_default\_current(usb\_phy);

INIT\_WORK(&usb\_phy->chg\_work, usb\_phy\_notify\_charger\_work);

}

3：在usb\_phy\_notify\_charger\_work中获取充电，类型并notify消息 notify the USB charger state

static void usb\_phy\_notify\_charger\_work(struct work\_struct \*work)

{

struct usb\_phy \*usb\_phy = container\_of(work, struct usb\_phy, chg\_work);

char uchger\_state[50] = { 0 };

char \*envp[] = { uchger\_state, NULL };

unsigned int min, max;

switch (usb\_phy->chg\_state) {

case USB\_CHARGER\_PRESENT:

if (usb\_phy->chg\_type == UNKNOWN\_TYPE)

usb\_phy->chg\_type = usb\_phy->charger\_detect(usb\_phy);//这个chg\_type更新后，pmic可以获取

usb\_phy\_get\_charger\_current(usb\_phy, &min, &max);

atomic\_notifier\_call\_chain(&usb\_phy->notifier, max, usb\_phy);

snprintf(uchger\_state, ARRAY\_SIZE(uchger\_state),

"USB\_CHARGER\_STATE=%s", "USB\_CHARGER\_PRESENT");

break;

case USB\_CHARGER\_ABSENT:

usb\_phy\_set\_default\_current(usb\_phy);

atomic\_notifier\_call\_chain(&usb\_phy->notifier, 0, usb\_phy);

snprintf(uchger\_state, ARRAY\_SIZE(uchger\_state),

"USB\_CHARGER\_STATE=%s", "USB\_CHARGER\_ABSENT");

break;

default:

dev\_warn(usb\_phy->dev, "Unknown USB charger state: %d\n",

usb\_phy->chg\_state);

return;

}

kobject\_uevent\_env(&usb\_phy->dev->kobj, KOBJ\_CHANGE, envp);

}

4：

设置电流会shedule 充电work

usb\_phy\_set\_charger\_current

--->schedule\_work(&usb\_phy->chg\_work);