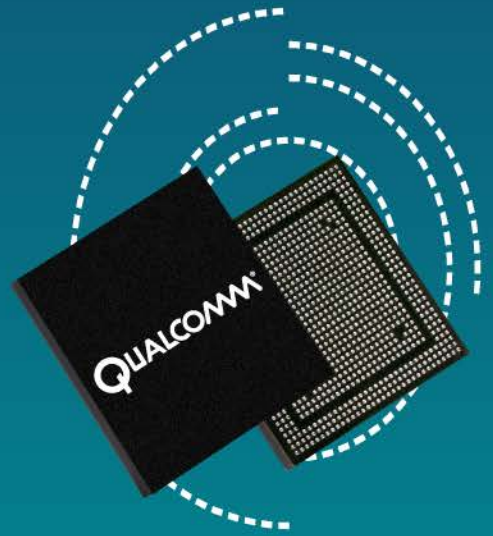
Qualcomm
TECHNOLOGIES, INC



PM8941 Linux Drivers Overview

80-NA157-36 B

Qualcomm Technologies, Inc.
5775 Morehouse Drive
San Diego, CA 92121
U.S.A.

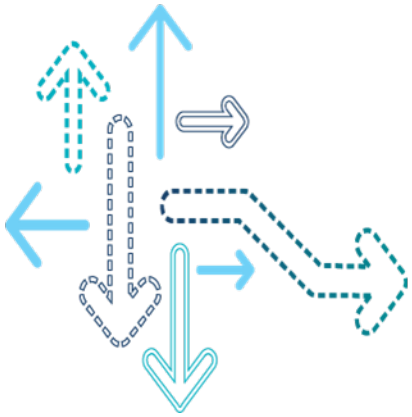
Revision History

Revision	Date	Description
A	Aug 2012	Initial release
B	Apr 2014	Updated setting for unused BAT_ID pin

Contents

- PM8941 PMIC Chip Overview
- PM8941 Linux PMIC APIs
- References
- Questions?

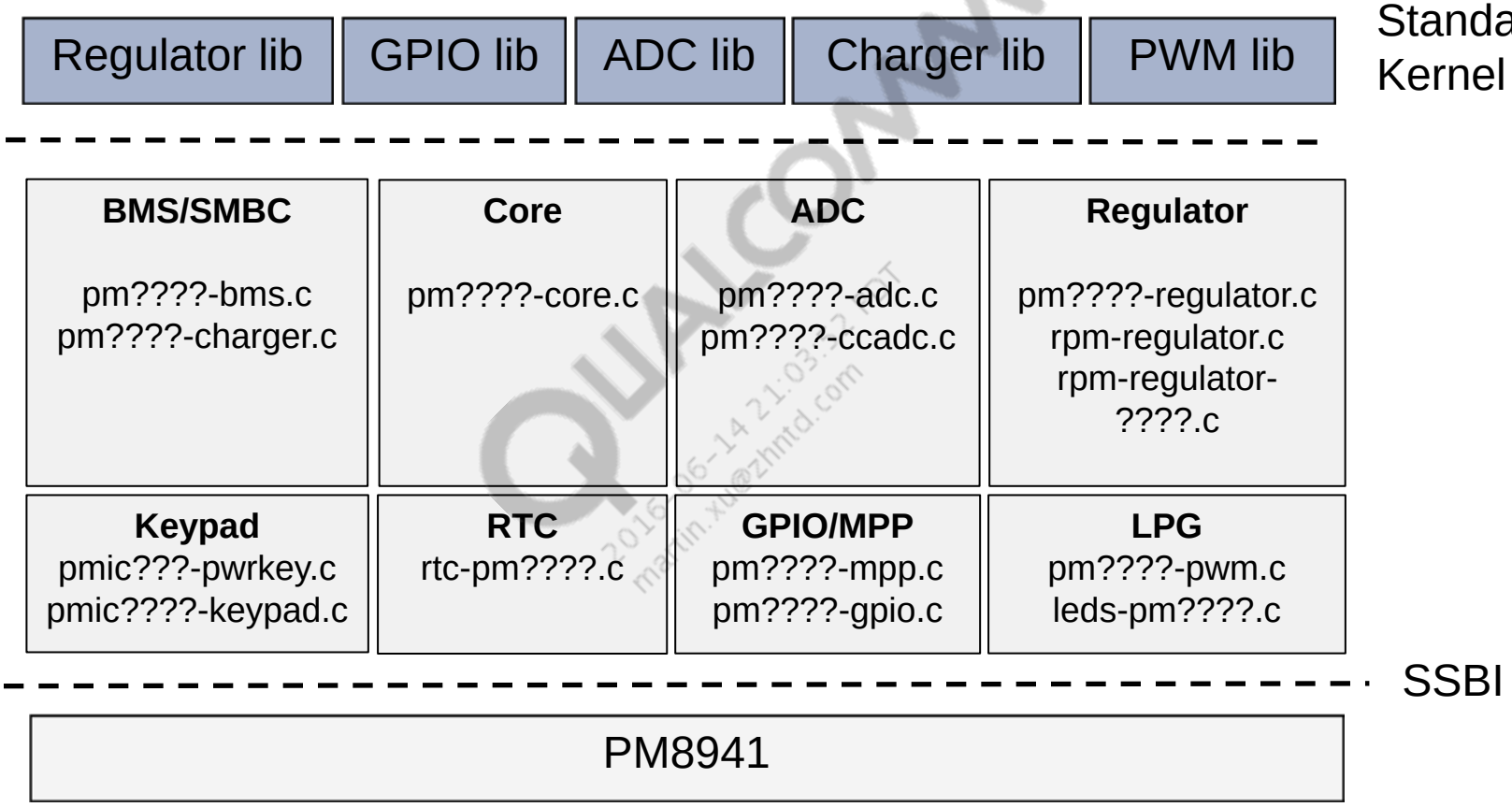
PM8941 PMIC Chip Overview



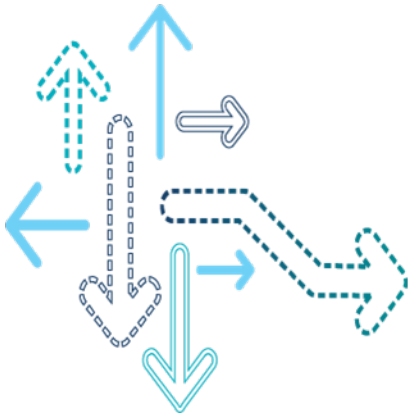
Overview

- This document briefly covers the various PMIC APIs available in the Linux kernel on the MSM8974 apps processors.
- PM8941 features include regulators, LPG, charger, fuel gauge, ADC, RTC, etc.

PM8941 Linux Software Architecture



PM8941 Linux PMIC APIs



PM8941 Core Driver

- Location – kernel/drivers/mfd/pm????-core.c
- The driver provides a communication layer between the SSBI driver (under I2C framework) and other PMIC function drivers, such as keypad, MPP, etc.
- It also provides interrupt multiplexing
- External APIs
  - pm????_readb(const struct device *dev, u16 addr, u8 *val)
  - pm????_writeb(const struct device *dev, u16 addr, u8 val)
  - pm????_read_buf(const struct device *dev, u16 addr, u8 *buf, int cnt)
  - pm????_write_buf(const struct device *dev, u16 addr, u8 *buf, int cnt)
  - pm????_read_irq_stat(const struct device *dev, int irq)

PM8941 Regulator Driver

- Location – kernel/drivers/regulator/pm????_regulator.c
- Header file – kernel/include/linux/regulator/pm????-regulator.h
- pm????_regulator driver is normally used only during bringup because it cannot safely disable or change the voltage of regulators that are shared with other subsystems
- RPM regulator drivers send regulator requests to RPM via shared memory
- RPM aggregates these requests with those from other subsystems before modifying PMIC regulator registers
- MSM8974 RPM regulator drivers are located in kernel/arch/arm/mach-msm/
  - rpm-regulator.c
  - rpm-regulator-????c

Important Regulator APIs

- regulator_get(dev, id) – Returns regulator handle for consumer to use
- regulator_set_voltage(regulator, min_uV, max_uV)
  - Sets voltage to value within range specified
  - Aggregated with other consumers' ranges as well as board file-defined constraint range
- regulator_set_optimum_mode(regulator, load_uA) – Sets mode of regulator so that it can output at least the current specified
  - Summed with other consumers' current requirements before making mode selection decision; note that this function returns values greater than 0 on success
- regulator_enable(regulator) – Enable regulator
  - Aggregated with other consumers' enable/disable requests
- regulator_disable(regulator) – Disable regulator
  - Segregated with other consumers' enable/disable requests
- regulator_put(dev, id) – Frees (with kfree) regulator handle

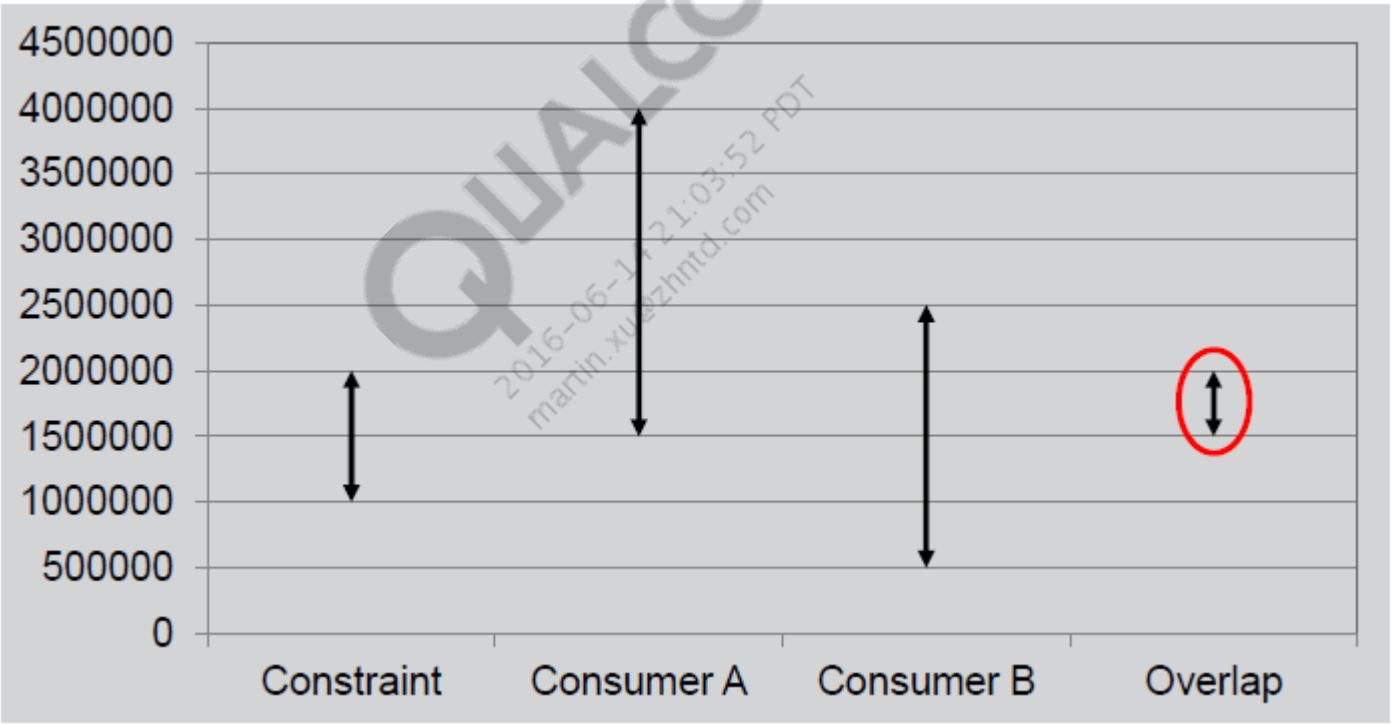
Regulator Declaration and Constraints – RPM Control

- Regulator framework values
  - a_on – Always_on (disable is nonoperational)
  - min_uV – Minimum allowed voltage
  - max_uV – Maximum allowed voltage
  - supply – Parent regulator
- Driver-specific values
  - ID – Driver-specific ID
  - pd – Pull-down enabled when off
  - ss – Sleep selectable
  - sys_uA – System load current
  - init_ip – Initial peak current set in base RPM request

```
static struct rpm_regulator_init_data
msm_rpm_regulator_init_data[] __devinitdata = {
                /* ID a_on  pd  ss   min_uV   max_uV   supply    sys_uA  freq */
RPM_SMPS(S1,    1,    1,   0,  1225000,  1225000,  NULL,     100000,  3p20),
RPM_SMPS(S4,    1,    1,   0,  1800000,  1800000,  NULL,     100000,  3p20),
               /* ID a_on  pd   ss  min_uV     max_uV  supply     sys_uA  init_ip */
RPM_LDO(  L1,    1,    1,   0,   1050000,  1050000, "8921_s4",   0,       10000),}
```


Voltage Aggregation Example

- Board file constraint for regulator foo – 1,000,000 to 2,000,000 µV
- Consumer A's request for foo – 1,500,000 to 4,000,000 µV
- Consumer B's request for foo – 500,000 to 2,500,000 µV



- Regulator driver sets to minimum physically possible setpoint in the overlap range – 1,500,000 to 2,000,000 µV

Load Current Aggregation Example

- Regulator S2 can supply up to 100,000 µA in LPM and 1,500,000 µA in HPM.
- Regulator L5 can supply up to 10,000 µA in LPM and 300,000 µA in HPM.
- Consumer A requires 9000 µA when active and 2000 µA when suspended.
- Consumer B requires 7000 µA all the time.
  - If A and B are supplied by S2 and both call regulator_set_optimum_mode with their current requirements, then S2 is put into LPM by the regulator driver.

    9000 + 7000 = 16000 < 100,000 and 2000 + 7000 = 9000 < 100,000
  - If A and B are supplied by L5, then L5 will be put into HPM when A is active and LPM when A is suspended.

    9000 + 7000 = 16,000 ≥ 10000 and 2000 + 7000 = 9000 < 10,000

PM8941 GPIO Driver

- Located in kernel/drivers/gpio/pm????-gpio.c
- pm????-gpio driver is middle layer between PMIC hardware and Android™ GPIO framework (kernel/Documentation/gpio.txt)
- Use gpiolib APIs
  - gpio_request()
  - gpio_set_value_cansleep() – Output
  - gpio_get_value_cansleep() – Input
  - gpio_to_irq() – Map gpio number to irq number
  - gpio_free()
- Use interrupt APIs for gpio interrupt
  - request_threaded_irq()
  - enable_irq()
  - disable_irq()
  - free_irq()

PM8941 GPIO Driver (cont.)

- Macro PM8941_GPIO_PM_TO_SYS is used to convert local GPIO pin number to systemwide GPIO number
- Example – Set the GPIO_10 pin to high

```
int pm_gpio = 10; // gpio 10 in schematics
int sys_gpio, err;
int status = 1;   // high = 1, low = 0
sys_gpio = PM8941_GPIO_PM_TO_SYS(pm_gpio);
err = gpio_request(sys_gpio);
If (err){
   // handle the error
}
gpio_set_value_cansleep(sys_gpio, status);
```


PM8941 IRQ Driver

- Located in kernel/drivers/mfd/pm????-irq.c
- PM8941 provides three interrupt outputs
  - PM_INT_SEC_N – Secure interrupt to TrustZone of apps processor
  - PM_INT_USR_N – User interrupt to User mode of apps processor
  - PM_INT_MDM_ N – Standard interrupt to modem processor
- For interrupts that are of interest to more than one processor, interrupt should go to SEC (highest security), then modem (lowest power) or USR
- Secure processor defines which nonsecure processor (USR/modem) should get interrupt and sets permission accordingly
- Permission interrupts on Linux side
  - CHARGER, RTC/PON, OSC Halt, Cable, Temp, PWR key, HSED, ADC, Keypad, BATT_TEMP, LPG, BMS, MPP, and GPIO

PM8941 Interrupts

- Interrupt trigger type
  - FE – Falling Edge triggered
  - RE – Rising Edge triggered
  - BE – Both Edges; interrupt is trigged on both interrupt edges, i.e., on state change
  - H – High-level triggered; triggered when the expression is TRUE; cannot be cleared if the signal is still high
  - L – Low-level triggered; triggered when the expression is FALSE; cannot be cleared if the signal is still low

Note: Continuous interrupts can happen if the trigger type is set to level and the IRQ is not masked after the interrupt is triggered.

PM8941 MPP Driver

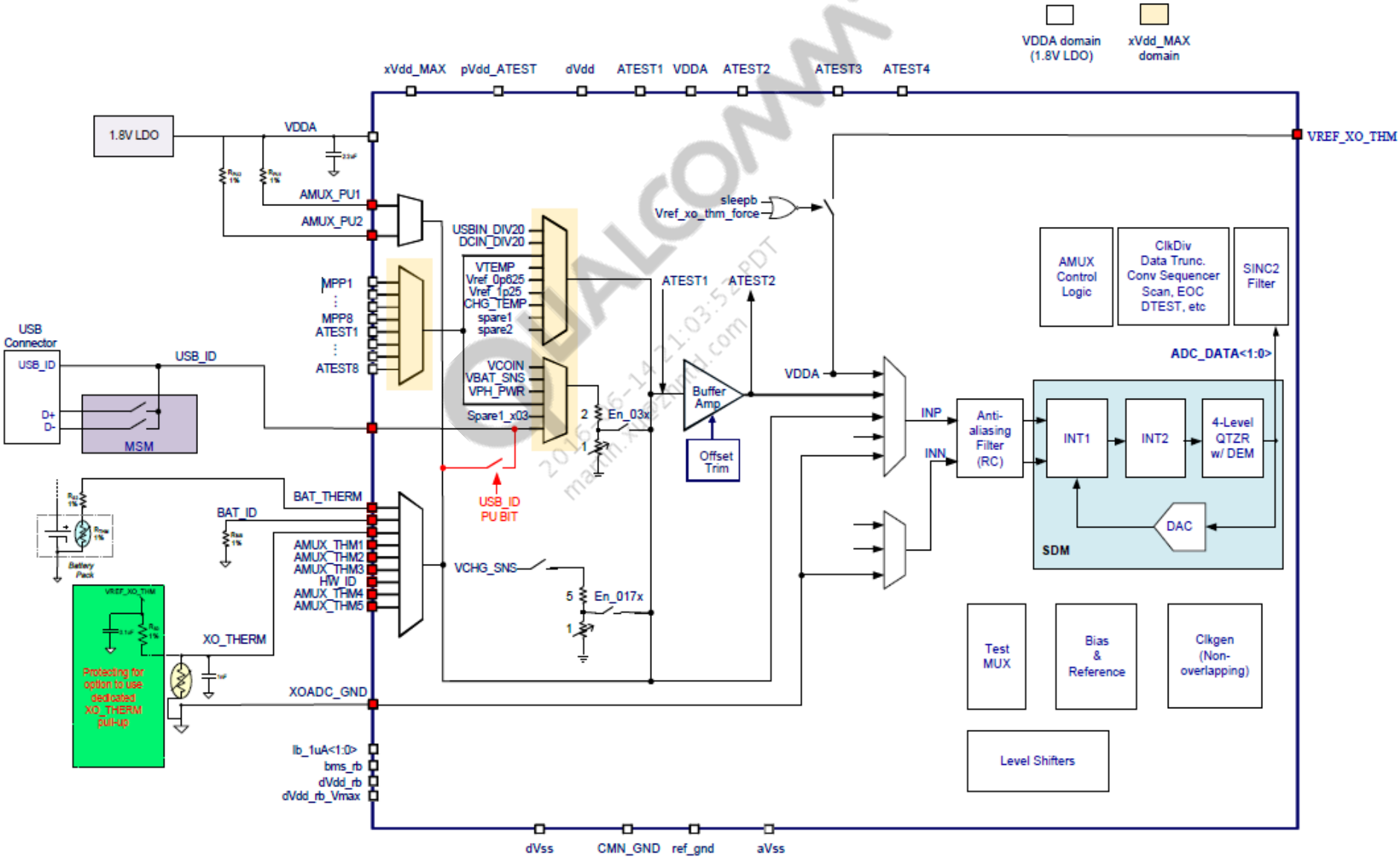
- Macro PM8941_MPP_PM_TO_SYS is used to convert the local MPP pin number to a systemwide GPIO number
- Example – Read the status of MPP_3 pin

```
int pm_mpp = 3; // mpp_3 in schematics
int sys_gpio, err;
int status;

sys_gpio = PM8941_MPP_PM_TO_SYS(pm_mpp);
err = gpio_request(sys_gpio);
If (err){
   // handle the error
}

// low: status=0, high: status=1
status = gpio_get_value_cansleep(sys_gpio);
```


PM8941 ADC Driver

- Located in kernel/drivers/mfd/pm????-adc.c
- ADC conversions can be requested by configuring up to five arbiter register banks
  - ADC_ARB_SECP
  - ADC_ARB_USRP
  - ADC_ARB_MP
  - ADC_ARB_BMS
  - ADC_ARB_BTM
- When a bank requests a conversion, its mirrored amux/ADC register contents are written to the corresponding amux/ADC register
- Converted ADC value is stored in the bank's data registers when conversion is completed
- With arbitration taken care of automatically in the hardware, each client can control the amux and ADC as if it has its own amux and ADC

PM8941 Amux Channels



PM8941 ADC Driver APIs

- API to read ADC value of dedicated pin
  - pm????_adc_read(enum pm????_adc_channels channel, struct pm????_adc_chan_result *result)
  - Example – Indicate end of charging time based on the charging current
    - pm????_adc_read(CHANNEL_ICHG, &result)
- API to read ADC value of MPP pin
  - pm????_adc_mpp_config_read(uint32_t mpp_num, enum pm????_adc_channels channel, struct pm????_adc_chan_result *result);
  - Example – Query voltage of main_therm pin via mpp_7 on AP side
    - pm????_adc_mpp_config_read(pm????_AMUX_MPP_7, ADC_MPP_1_AMUX6, &result)

PM8941 Charger Driver

- Located in kernel/drivers/power/pm????-charger.c
- Before charging starts, battery has to be present
- PM8941 provides two ways to identify the battery presence
  - BAT_ID
  - BAT_THEM
- Battery presence is detected by sensing presence of battery thermistor or ID resistor
- Two dedicated BPD comparators monitor BAT_THM and BAT_ID voltage level; battery is considered as gone if either one is above the 95% threshold
- If BAT_ID is not used, the unused pin has to be set as follows:
  - Internal charger + external BMS – Connect to GND
  - External charger + internal or external BMS – NC
  - External charger + bharger boost + internal or external BMS – NC
- Some customers are unable to start charging since they use BAT_THM to detect battery presence and forget to ground BAT_ID

Battery Temperature Monitoring

- To prevent permanent damage of battery, battery charging is stopped if the battery temperature is out of range
- PM8941 BTM is used to monitor battery temperature
- No BTM during the first hardware-controlled ATC
- SMBC BTM is disabled before the software configures it
  - Enable BTM – pm_chg_masked_write(chip, CHG_CNTRL_2, CHG_BAT_TEMP_DIS_BIT, 0) in m8921_chg_hw_init()
- For customers using the external fuel gauge
  - Disable BTM – pm_chg_masked_write(chip, CHG_CNTRL_2, CHG_BAT_TEMP_DIS_BIT, 1) in m8921_chg_hw_init()
- BTM will be in later ATCs if VCOIN > 2 V

Battery Temperature Monitoring (cont.)

- Configuration of cool and warm thresholds – JEITA compliance only

```
static struct
pm????_charger_platform_data pm????_chg_pdata __devinitdata =
{
  .cool_temp      = 10,                 // 10 degree celsius
  .warm_temp      = 40,                 // 40 degree celsius
  .cool_bat_chg_current   = 350,   // 350 mA  (max value = 2A)
  .warm_bat_chg_current   = 350, // 350 mA
  .temp_check_period =1// 1 second (max value = 16 seconds)
};
```

Selection of Thermistor Pullup Resistors (Rs1 and Rs2)

1. Find battery thermistor parameters – Room temperature resistance (R0) and temperature coefficient (B)
2. Determine allowable battery charging temperature range, e.g., 0°C (TCOLD) to 40°C (THOT)
3. Calculate thermistor resistance at cold and hot
   - RCOLD = R0·exp(B·(1/TCOLD-1/To))
   - RHOT = R0·exp(B·(1/THOT-1/To))
4. Select BTM comparator thresholds
   - For traditional battery charging temperature window, such as 0°C to 40/45°C, the cold and hot thresholds should be set to 70% and 35%, respectively.
   - For the JEITA-compliant extended battery charging temperature window, such as -10°C to 60°C, select 80% and 25% as the cold and hot threshold, respectively.

Selection of Thermistor Pullup Resistors (Rs1 and Rs2) (cont.)

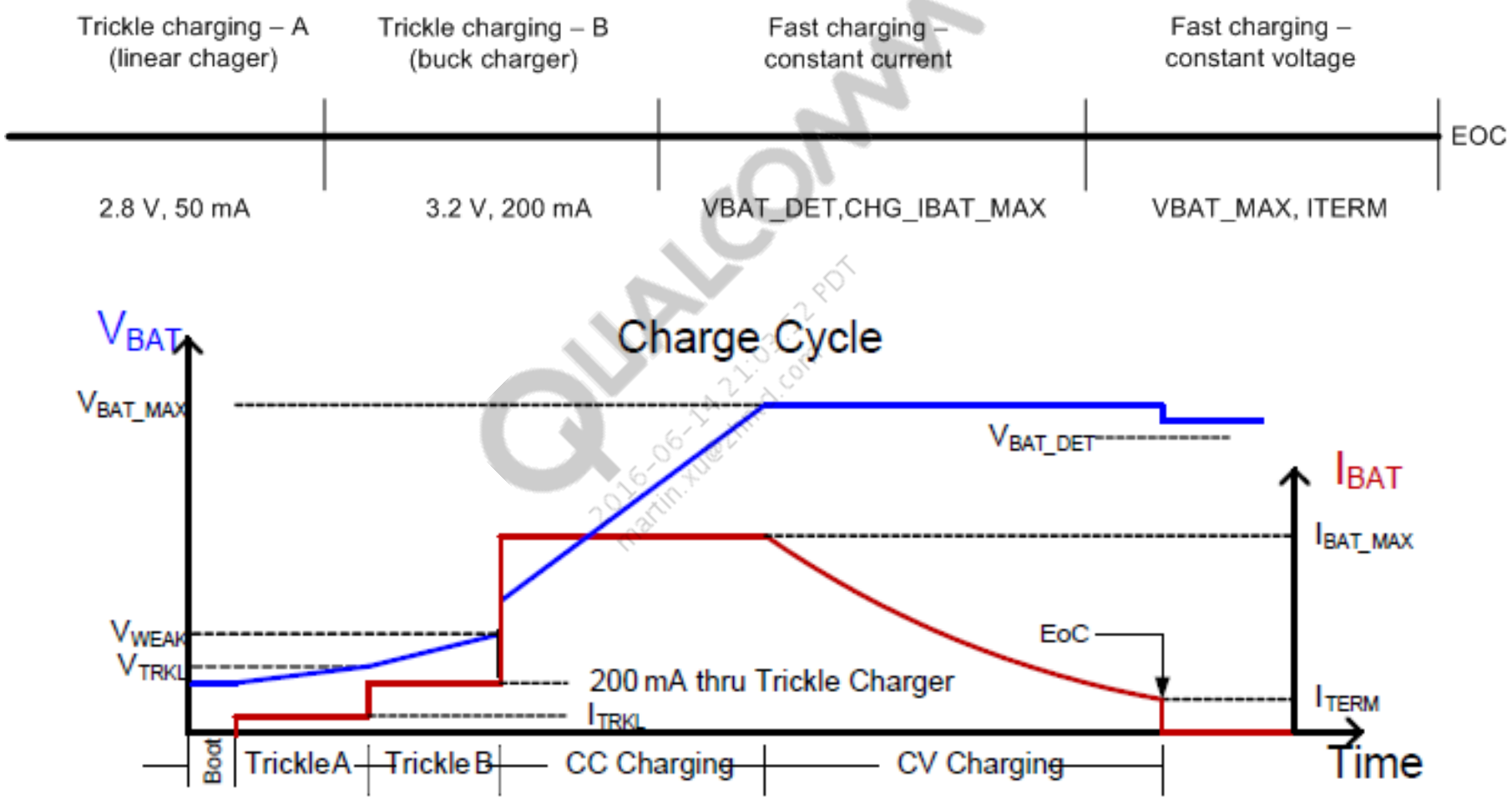
- Another root cause that charging cannot be started – If customer selects a wrong resistor, the charger may not start because the PMIC chip will incorrectly regard the temperature as too cold or too hot.

Batter charging temperature window	BTM comparator thresholds	R_{S1} and R_{S2} calculation
0°C to 40/45°C	70%/35%	$R_{S1} = \frac{39 \bullet (R_{COLD} - R_{HOT})}{70}$ $R_{S2} = \frac{3R_{COLD} - 13R_{HOT}}{70}$
-10°C to 60°C	80%/25%	$R_{S1} = \frac{4 \bullet (R_{COLD} - R_{HOT})}{15}$ $R_{S2} = \frac{R_{COLD} - 16R_{HOT}}{15}$

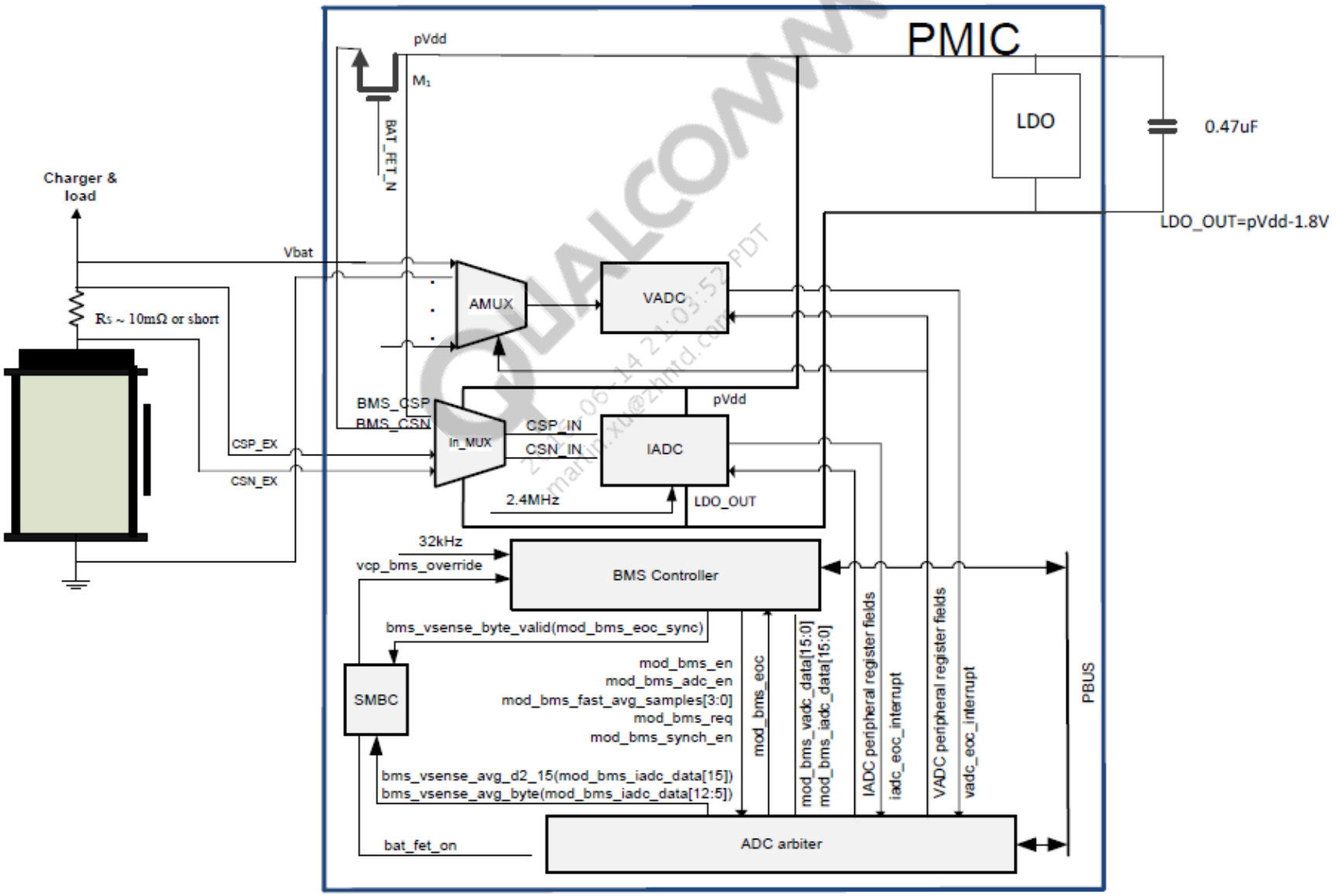
PM8941 Linux Charger APIs

- Enable/disable drawing current from source
  - pm????_disable_source_current – Be cautious while using this API; it will force the device to run from the battery
- USB supply max current – pm????_charger_vbus_draw
- Restrict charging current flowing in battery – pm????_set_max_battery_charge_current
- Set max charging timer – pm_chg_tchg_max_set
- Set charger termination current – pm_chg_iterm_set
- Set max charging current – pm????_set_max_battery_charge_current
- Set max charging voltage – pm_chg_vddmax_set
- Set safe (max and one time settable) charging current – pm_chg_ibatsafe_set
- Set safe (max and one time settable) charging voltage – pm_chg_vddsafe_set
- Set charging resume voltage – pm_chg_vbatdet_set

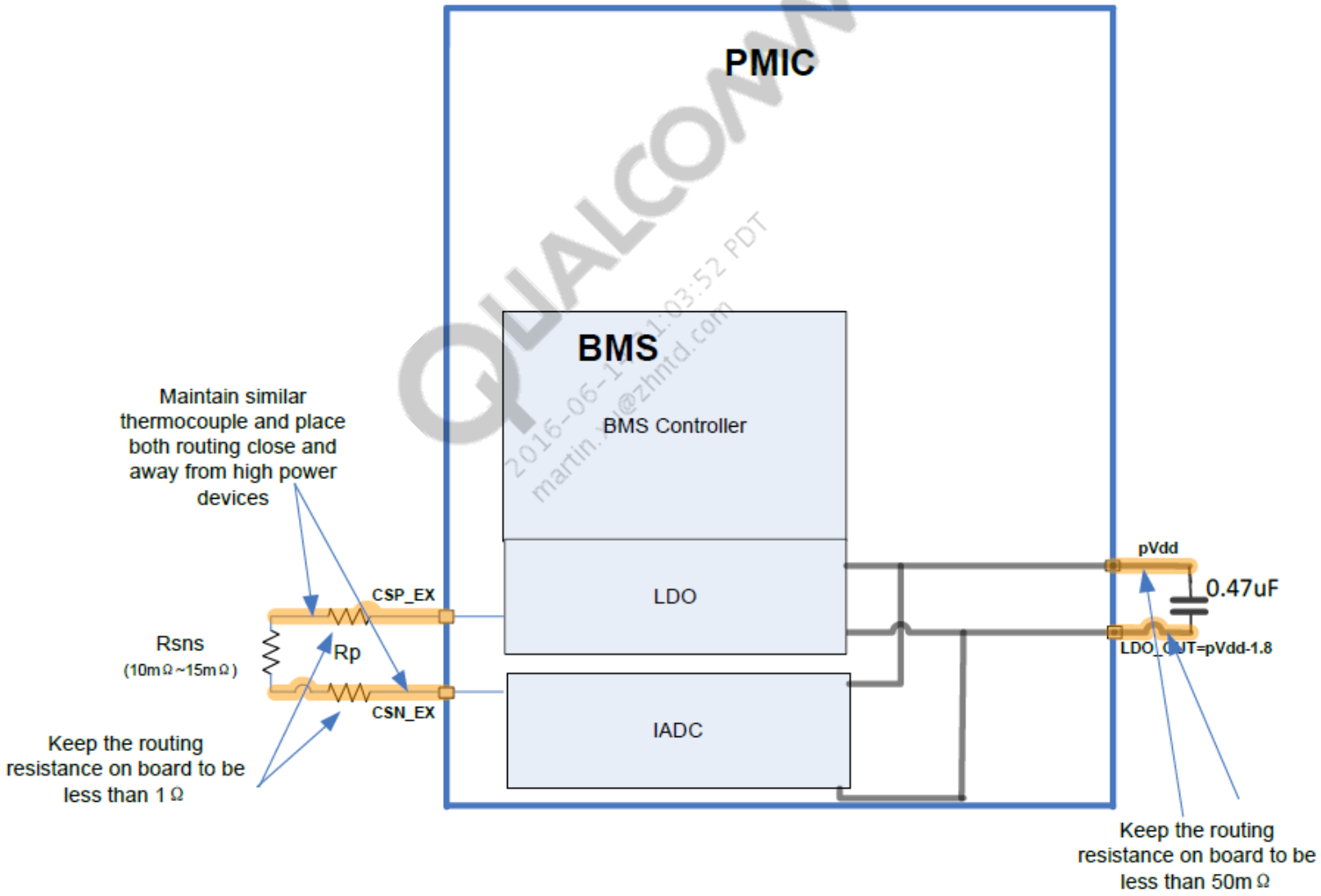
Trickle Charging and Fast Charging



PM8941 Battery Monitoring System (BMS)



PM8941 BMS Layout Recommendation

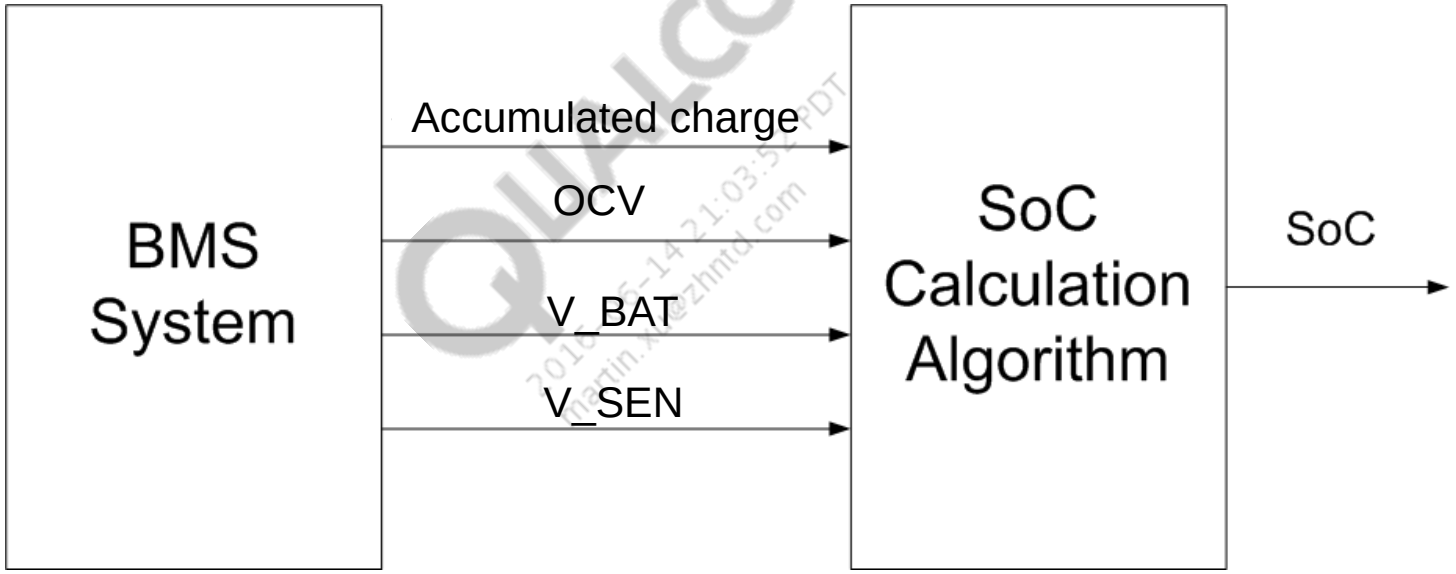


BMS Components

- CCADC – Coulomb Counter (counting accumulated charge) Analog-to-Digital Converter
  - Produces digitized Vsense
- XOADC – Crystal Oscillator Analog-to-Digital Converter
  - Produces digitized Vbatt
- BMS controller
  - Controls turn-on and turn-off of analog frontend
  - Determines what data (Vsense, Vbatt) is necessary at what time for accurate State-of-Charge (SoC)
  - SoC approximation software is located on the apps processor

PM8941 BMS

- Purpose of the BMS is to obtain SoC
  - SoC is the percentage of remaining usable capacity on a scale from 0% to 100%



Coulomb Counter

- Coulomb Counter (CC) is 2s compliment counter, which is centered at 0x0000_0000
- CC updates from battery Open Circuit Voltage (OCV) to reduce integrated error of SoC
- CC counts up when charge is removed from battery
  - BMS_CSP is negative relative to BMS_CSN pin
- CC counts down when battery is being charged
  - BMS_CSP is positive relative to BMS_CSN pin
- BMS_CSP must always be connected to the negative side of the battery
- BMS_CSN must be grounded

PM8941 SoC Algorithm

- BMS computes battery impedance Rbatt by three parameters
  - OCV_for_R
  - Vbat_for_R
  - Vsense_for_R

Rbatt = (OCV_for_R - Vbat_for_R) * Rsense/Vsense_for_R

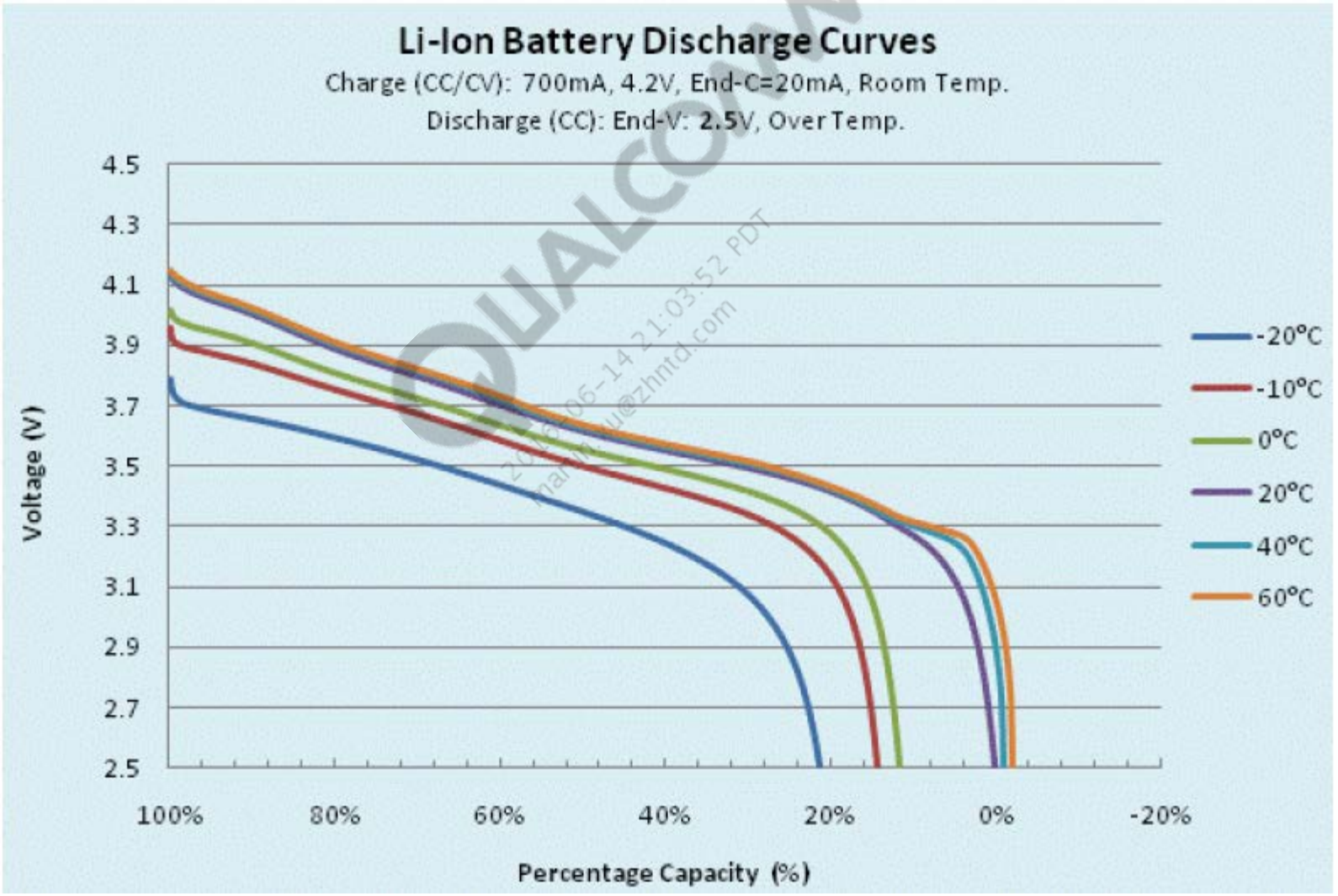
UUC = FCC*Lookup(Rbatt * Ipeak + Vfailure)

RC = FCC * Lookup(OCV)

RUC = RC - (CC * sample_time/Rsense) - UUC

SoC = RUC/(FCC - UUC)

Battery Voltage vs Percentage Charge Under Different Temperatures



QCT BMS Profiling Tool

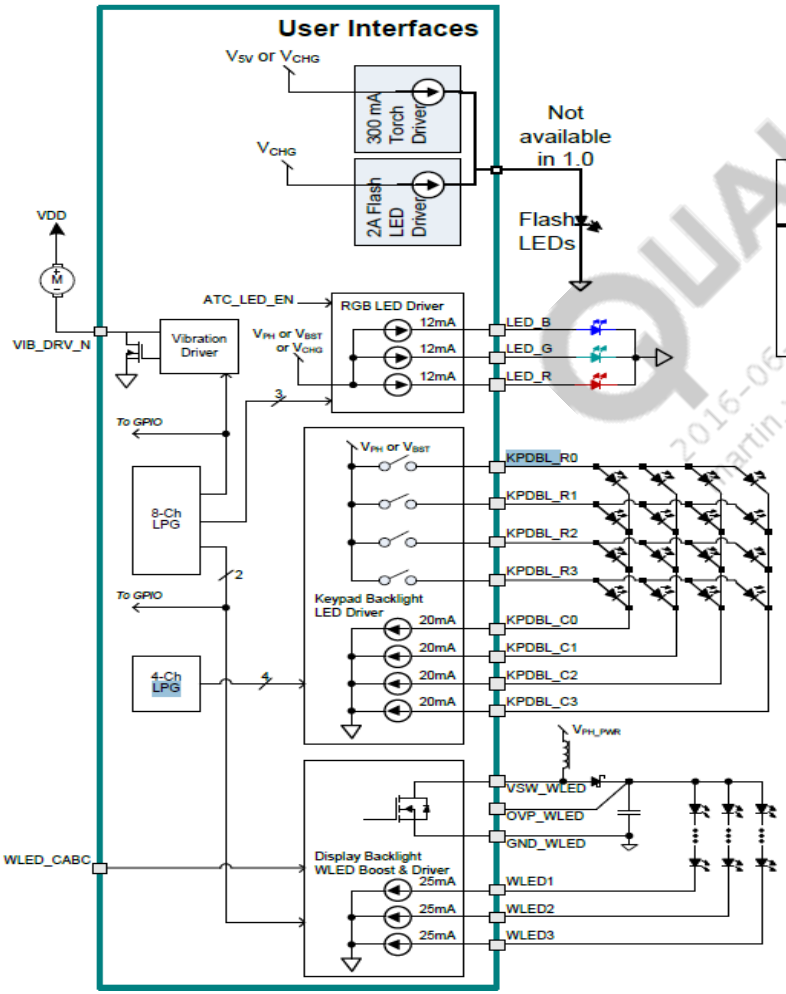
- To compute SoC, a battery profile is needed.
- The battery profile is stored in kernel/arch/arm/mach-msm/bms-batterydata.c.
- The battery profile consists of five lookup tables.
  - fcc_temp
  - fcc_sf (full charge capacity _scale factor) – Charge cycle related
  - pc_sf (percentage charge _scale factor)
  - pc_temp_ocv
  - rbatt
- Battery profile can be generated by the BMS calibration tool, which is supported by QDART.
- See [Q4].

Acronyms of PM8941 BMS Driver

Acronym	Description
DC	Design Capacity – Amount of energy that is stored within a new battery
FCC	Full Charge Capacity – Amount of charge passed from fully charged state to terminated voltage at discharged current less than 1/20; FCC changes with age and cycle life of battery
RC	Remaining Capacity – Amount of charge stored from present state to terminated voltage, assuming the energy left in battery is almost 0 mAh when discharging at low current
UUC	Unusable Capacity – Battery capacity that cannot be used when terminated voltage is reached; function of discharging current
UC	Usable Capacity – Charge held by the battery after FCC minus the charge that cannot be used at given load due to impedance, UC = FCC - UUC
RUC	Remaining Usable Capacity – RC - UUC
SoC	State-of-Charge – Defined as the ratio for RC to FCC; SoC = RC/FCC; most commercial products define SoC as RUC/UC, which is more useful to report to the end user
C-rate	Unit by which charge and discharge times are scaled; battery rated at 1 Ah provides 1 A for 1 h if discharged at 1°C; same battery discharged at 0.5°C would provide 500 mA for 2 h
OCV	Open Circuit Voltage – Battery voltage at 0 V (or near 0 V, which is less than C/20) current; must wait 5 to 30 min for battery to stabilize at this voltage; time constant of battery varies with types of battery temperature and also varies with aging and cycle life of battery

PM8941 LPG Driver

- Located in kernel/drivers/mfd/pm????-pwm.c
- PM8941 has eight LPG channels



LPG channel	8	7	6	5	4	3	2	1
Connects to	WLED1 GPIO36 DTEST8	RGB_LED 3 (R) GPIO35 DTEST7	RGB_LED 2 (G) GPIO34 DTEST6	RGB_LED 1 (B) GPIO33 DTEST5	WLED2 GPIO26 DTEST4	VIB WLED3 GPIO25 DTEST3	Flash_Drv 2 GPIO24 DTEST2	Flash_Drv 1 GPIO23 DTEST1

PM8941 LPG Driver (cont.)

- pm????-pwm driver supports the Android PWM framework
- PWM APIs
  - pwm_request()
  - pwm_config()
    - Configure ones PWM waveform – One period with one duty cycle
    - Note that the unit is µs, not ns
  - pwm_enable()
  - pwm_disable()
  - pwm_free()
- Use pm8941PWM LUT
  - pm????_pwm_lut_config()
  - pm????_pwm_lut_enable()

References

Ref.	Document	
Qualcomm Technologies		
Q1	*Application Note: Software Glossary for Customers*	CL93-V3077-1
Q2	*Presentation: MSM8960/PM8921 ADC Drivers Overview*	80-N8212-1
Q3	*Presentation: PM8921 Linux PMIC Interfaces – Charger and BMS*	80-N8278-1
Q4	*Application Note: Battery Characterization Test Procedure*	80-VA360-12

Questions?

https://support.cdmatech.com