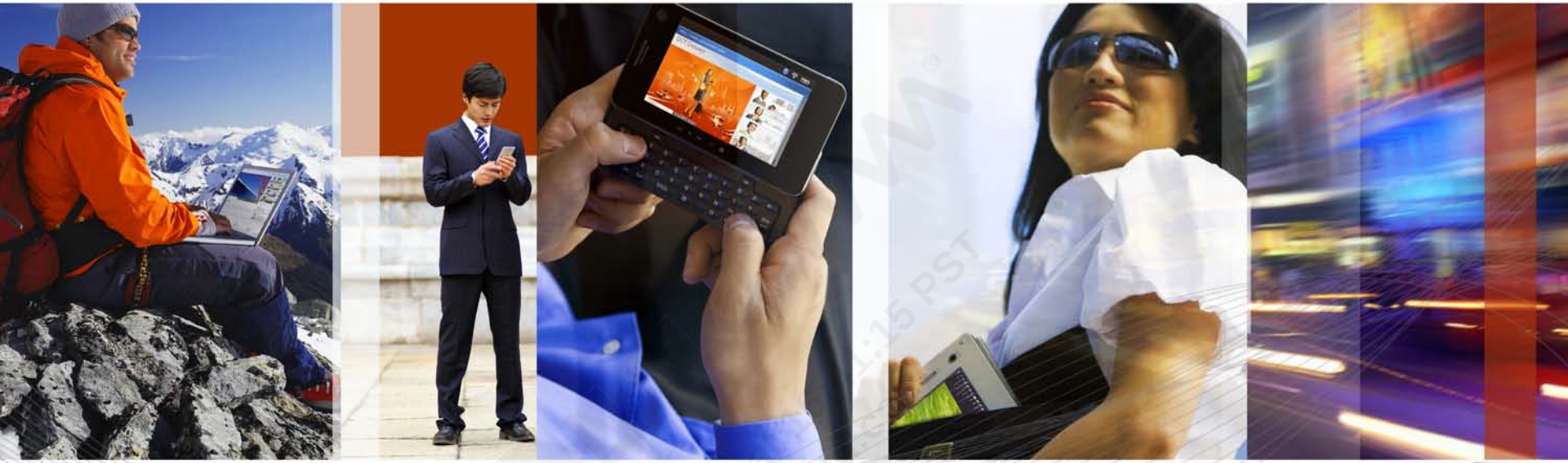

QUALCOMM CDMA Technologies

REDEFINING MOBILITY

MSM8960™ Linux PMIC Drivers Overview

80-N6836-1 A

QUALCOMM Incorporated
5775 Morehouse Drive
San Diego, CA 92121-1714
U.S.A.

Revision History

Version	Date	Description
A	Jul 2011	Initial release

Contents

PM8921™ Overview

Overview

- This document will briefly cover the various PMIC APIs available in the Linux kernel on the MSM8960™ apps processors.
- The apps processors will communicate with the PMICs through the SSBI buses.
- PM8921 provides switching and linear regulator support for a number of system rails and interfaces.
- PM8921 also features clock management, RTC, LPG, Switch mode battery charger, battery monitoring system, ADC, etc.

MSM8960 Power-On

- PM8921 PON module needs regulators L1_L2_L12_L18 for its operation
- Five events could trigger a power-on sequence
  - Keypad power-on button is pressed and KPD_PWR_N signal is pulled low
  - External supply source is detected (the voltage on the VCHG pin exceeds its valid threshold)
  - Real-time clock alarm is triggered
  - Serial cable is inserted and one or more cable power-on pins are pulled low
  - SMPL condition was detected and an SMPL recovery is initiated
- MSM™ or QSC is On when PS_HOLD signal is driven high

PM8921 Linux APIs

PM8921 Core Driver

- Location – kernel/drivers/mfd/pm8921-core.c
- Driver provides a communication layer between SSBI driver (under I2C framework) and other PMIC function drivers like keypad, MPP, etc.
- Also provides interrupt multiplexing
- External APIs
  - pm8921_readb(const struct device *dev, u16 addr, u8 *val)
  - pm8921_writeb(const struct device *dev, u16 addr, u8 val)
  - pm8921_read_buf(const struct device *dev, u16 addr, u8 *buf, int cnt)
  - pm8921_write_buf(const struct device *dev, u16 addr, u8 *buf, int cnt)
  - pm8921_read_irq_stat(const struct device *dev, int irq)

PM8921 Kernel Permission IRQs

- CHARGER
- RTC/PON
- OSC halt
- Cable
- Temp
- PWR key
- HSED
- ADC
- Keypad
- BATT_TEMP
- LPG
- BMS
- MPP
- GPIO

PM8921 Regulator Driver

- Location – kernel/drivers/regulator/pmic8921_regulator.c
- Header file – kernel/include/linux/regulator/pmic8921-regulator.h
- Qualcomm has taken care of hooking pmic8921_regulator APIs to standard Linux regulator API framework found in consumer.h
- Some APIs licensees should use
  - regulator_enable(struct regulator *regulator)
  - regulator_disable(struct regulator *regulator)
  - regulator_set_voltage(struct regulator *regulator, int min_uV, int max_uV)
- A REGULATOR_SUPPLY line must be added for each regulator that a driver uses under the appropriate VREG_CONSUMER in the regulator board file (arch/arm/mach-msm/board-msm8960-regulator.c for MSM8960)

PM8921 Regulator Driver (cont.)

- Example 1

```
 VREG_CONSUMERS(L12) = {
    REGULATOR_SUPPLY("8921_l12",     NULL), /* the existing entries with NULL
  as the dev name will go away */
     REGULATOR_SUPPLY("8921_l12",     "reg-debug-consumer"),
  REGULATOR_SUPPLY(“foobar_1.2V”,  "foobar_dev"),};


#include <linux/regulator/consumer.h>
static struct regulator *foo_vreg;
static int foo_vreg_on;

/* This regulator name must match the one specified in the regulator board
  file. */

static const char *foo_vreg_name = "VDD_foo_3v";
#define FOO_LOAD_UA 34000
#define FOO_MIN_UV 3000000
#define FOO_MAX_UV 3300000

int foo_vreg_init(struct device *foo_dev) {
  int rc;
  foo_vreg = regulator_get(foo_dev, foo_vreg_name);
  if (!foo_vreg || IS_ERR(foo_vreg)) {
      rc = PTR_ERR(foo_vreg);
      vreg = NULL;     }
  foo_vreg_on = 0;
  return rc;
}
```

PM8921 Regulator Driver (cont.)

```
int foo_vreg_power(int on, int min_microvolts) {
    int rc = 0;

    if (!foo_vreg)
            return -EINVAL;

    if (on && !foo_vreg_on) {

            rc = regulator_set_voltage(foo_vreg, min_microvolts, FOO_MAX_UV);
            if (rc) {
                        pr_err("%s: failed to set voltage for %s, rc=%d\n", __func__, foo_vreg_name, rc);
                        goto done;
            }
            rc = regulator_set_optimum_mode(foo_vreg, FOO_LOAD_UA);
            if (rc < 0) {
                        pr_err("%s: failed to set optimum mode for %s, rc=%d\n", __func__, foo_vreg_name, rc);
                        goto done;
            }
            rc = regulator_enable(foo_vreg);
            if (rc) {
                        pr_err("%s: failed to enable %s, rc=%d\n", __func__, foo_vreg_name, rc);
                        goto done;
            }
            foo_vreg_on = 1;
    }
    else if (!on && foo_vreg_on) {
            /* disable the regulator after setting voltage and current */
            rc = regulator_set_voltage(foo_vreg, 0, FOO_MAX_UV);
            if (rc) {
                        pr_err("%s: failed to set voltage for %s, rc=%d\n", __func__, foo_vreg_name, rc);
                        goto done;
            }
            rc = regulator_set_optimum_mode(foo_vreg, 0);
            if (rc < 0) {
                        pr_err("%s: failed to set optimum mode for %s, rc=%d\n", __func__, foo_vreg_name, rc);
                        goto done;
            }
            rc = regulator_disable(foo_vreg);
            if (rc) {
                        pr_err("%s: failed to disable %s, rc=%d\n", __func__, foo_vreg_name, rc);
                        goto done;
            }
            foo_vreg_on = 0;
    }
done:
    return rc;
}
```

PM8921 Regulator Driver (cont.)

- Example 2

```
/* Called when the regulator is no longer required at all by the driver,
   typically in module_exit, etc */

int foo_vreg_free(struct device *foo_dev) {
   int rc;
   rc = foo_vreg_power(0, 0);
   regulator_put(foo_vreg);
   foo_vreg = NULL;
   return rc;
}
```

PM8921 ADC Driver

- Location – kernel/drivers/mfd/pm8921-adc.c
- Header file – kernel/include/linux/mfd/pm8921-adc.h
- List of supported ADC APIs
  - pm8921_adc_read(enum pm8921_adc_channels channel,
    struct pm8921_adc_chan_result *result)
  - pm8921_adc_mpp_read(enum pm8921_adc_mpp_channels channel,
    struct pm8921_adc_chan_result *result,
    enum pm8921_adc_premux_mpp_scale_type mpp_scale)
  - pm8921_adc_btm_configure(struct pm8921_adc_arb_btm_param *btm_param)
  - pm8921_adc_btm_start(void)
  - pm8921_adc_btm_end(void)

PM8921 ADC Driver (cont.)

- Debugging ADC based on debugfs
  - mount -t debugfs none /sys/kernel/debug
  - cd /sys/kernel/debug
  - cd pm8921_adc
- Example
  - Get battery voltage – cat vbat
  - Get die temperature – cat die_temp
  - Get battery temperature – cat batt_therm
  - Get battery ID – cat batt_id
  - Get battery current – cat ibat

PM8921 Charger Driver

- Location – kernel/drivers/power/pm8921-charger.c
- Header file – kernel/include/linux/mfd/pm8xxx/pm8921-charger.h
- List of supported charger APIs
  - pm8921_charger_register_vbus_sn(void (*callback)(int))
  - pm8921_charger_unregister_vbus_sn(void (*callback)(int))
  - USB calls the following API to show how much max USB current can be drawn
    - pm8921_charger_vbus_draw(unsigned int mA)

PM8921 Charger Driver (cont.)

- When the phone has a good battery (>3.2 V), the phone is not drawing charging current from USB
- When USB is suspended, the phone is not allowed to draw more than 2.5 mA from a PC
- Charger uses L14 for its operation
- Charger IRQs
  - USBIN_VALID_IRQ, USBIN_OV_IRQ, BATT_INSERTED_IRQ, VBATDET_LOW_IRQ, USBIN_UV_IRQ, VBAT_OV_IRQ, CHGWDOG_IRQ, VCP_IRQ, ATCDONE_IRQ, ATCFAIL_IRQ, CHGDONE_IRQ, CHGFAIL_IRQ, CHGSTATE_IRQ, LOOP_CHANGE_IRQ, FASTCHG_IRQ, TRKLCHG_IRQ, BATT_REMOVED_IRQ, BATTTEMP_HOT_IRQ, CHGHOT_IRQ, BATTTEMP_COLD_IRQ, CHG_GONE_IRQ, BAT_TEMP_OK_IRQ, COARSE_DET_LOW_IRQ, VDD_LOOP_IRQ, VREG_OV_IRQ, VBATDET_IRQ, BATFET_IRQ, PSI_IRQ, DCIN_VALID_IRQ, DCIN_OV_IRQ, DCIN_UV_IRQ

PM8921 BMS Driver

- Location – kernel/drivers/power/pm8921-bms.c
- Header file – kernel/include/linux/mfd/pm8xxx/pm8921-bms.h
- List of supported charger APIs
  - pm8921_bms_get_vsense_avg(int *result)
  - pm8921_bms_get_percent_charge(void)
  - pm8921_bms_charging_began(void)
  - pm8921_bms_charging_end(void)
- Standalone BMS profiling tool is needed for BMS calibration

PM8921 BMS Driver (cont.)

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

PM8921 BMS Driver Q&A

Question Where are look-up tables for Resistance/Temperature/Voltage stored?

Answer They are stored with the code itself.

Question Are different battery types supported (LiPo, LiMn)?

Answer The BMS and Battery Profiling Tool can handle any battery that the PM8921 can tolerate.

Question How is battery aging handled in a fuel gauge?

Answer There are three parameters that will vary with age, battery resistance, battery capacity (FCC), and the battery voltage vs SoC profile. The first two will be updated through normal operation learning that takes place. The last can be characterized by the Battery Profiling Tool.

Question Are there any resistance tables or other mechanism for battery learning?

Answer Battery resistance is not a table value, but updated over time and stored. FCC is updated over time and stored, but can also be an aging table value. These are the two values that are learned.

Question Is it possible to configure compensation for I*R drop around sense resistor?

Answer Voltage readings in the ADC remove the sense IR drop from battery voltage readings.

Question What is an accuracy for SoC reported by a fuel gauge?

Answer It is 1% average and 2% worst case errors at room temperature using a 10 mΩ sense resistor.

Question What is the criteria to take OCV measurements (current thresholds, battery relax time, etc)?

Answer Current must be less than a threshold and successive battery readings must show battery has relaxed below a voltage delta threshold prior to a good OCV reading being logged.

References

Ref.	Document	
Qualcomm		
Q1	*Application Note: Software Glossary for Customers*	CL93-V3077-1

Questions?

https://support.cdmatech.com