PMIC driver Modification Guide

Based on X3J3 Platform

X3J3

v1.0

2021-02

Important Notice and Disclaimer

Information in this document is provided solely to enable system and software implementers to use Horizon products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

All statements, information and recommendations in this document are provided "AS IS". Horizon makes no warranty, representation or guarantee of any kind, express or implied, regarding the merchantability, fitness or suitability of its products for any particular purpose, and non-infringement of any third party intellectual property rights, nor does Horizon assume any liability arising out of the application or use of any product, and specifically disclaims any and all liability, including without limitation consequential or incidental damages.

"Typical" parameters that may be provided in Horizon datasheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals", must be validated for each customer application by customer's technical experts. Buyers and others who are developing systems that incorporate Horizon products (collectively, "Users") understand and agree that Users shall remain responsible for using independent analysis, evaluation and judgment in designing their applications and that Users have full and exclusive responsibility to assure the safety of Users' applications and compliance of their applications (and of all Horizon products used in or for Users' applications) with all applicable regulations, laws and other applicable requirements.

User agrees to fully indemnify Horizon and its representatives against any claims, damages, costs, losses and/or liabilities arising out of User's unauthorized application of Horizon products and non-compliance with any terms of this notice.

Horizon Robotics, Inc.

https://www.horizon.ai

Revision History

The revision record lists the major changes that have occurred between document versions.

The following table lists the technical content of each document update。

Version	Revision Date	Revision Description
0.1	2020-4-26	Document Created
1.0	2021-02	1.0 Released

Contents

1 Overview

1.1 Document overview

This document is used to guide how to code PMIC driver based on the kernel regulator framework. The document takes the AXP15060 power management module produced by X-powers as an example for description.

AXP15060 supports 23 power outputs (including 6 DCDC). In order to ensure the safety and stability of the power system, AXP15060 integrates protection circuits such as over voltage protection (OVP), under voltage protection (UVP) and over temperature protection (OTP).

1.2 Multiple power output

Table 1-1 lists the multiple power output of AXP15060 PMIC.

	Output path	type	Default voltage	Startup sequence	Application suggestion	Load capacity(MAX)
1	DCDC1	BUCK	3.3v	2	IO/USB	2000mA
2	DCDC2	BUCK	0.9v	2	CPU	3500mA
3	DCDC3	BUCK	0.9v	1	GPU	3500mA
4	DCDC4	BUCK	0.9v	1	SYS	2000mA
5	DCDC5	BUCK	1.1v/DC5SET	1	DDR	2000mA
6	DCDC6	BUCK	OFF	OFF	LDO	2000mA
7	ALDO1	LDO	OFF	OFF	N/A	600mA
8	ALDO2	LDO	1.8V	2	N/A	300mA
9	ALDO3	LDO	1.8V	2	N/A	200mA
10	ALDO4	LDO	3.3V	2	N/A	300mA
11	ALDO5	LDO	2.5V	1	N/A	300mA
12	BLDO1	LDO	OFF	OFF	N/A	300mA
13	BLDO2	LDO	OFF	OFF	N/A	500mA

14	BLDO3	LDO	OFF	OFF	N/A	300mA
15	BLDO4	LDO	OFF	OFF	N/A	400mA
16	BLDO5	LDO	1.8V	2	N/A	600mA
17	CLDO1	LDO	OFF	OFF	N/A	200mA
18	CLDO2	LDO	3.3V	2	N/A	200mA
19	CLDO3	LDO	OFF	OFF	N/A	300mA
20	CLDO4	LDO	OFF	OFF	N/A	200mA
21	VCPUS	LDO	0.9V	1	CPUs/Reference of DDR	200mA
22	RTC-LDO	LDO	1.8V	Always on	RTC	100mA
23	DC1SW	Switch	OFF	OFF	N/A	1000mA

Table1-1 AXP15060 Multi-Power Outputs

Which output ports are required to be determined according to the hardware requirements.

2 DTS Configuration

Configure the specific DTS according to the hardware schematic diagram. The Horizon Robotics X3J3 platform uses the DCDC1 ~ DCDC6, BLDO1, CLDO2 and other voltage output ports of the AXP15060 PMIC。

2.1 X3J3 Platform PMIC Configuration

The PMIC DTS configuration of Horizon Robotics X3J3 platform is as follows:

```
&i2c0 {
   axp15060@37 {
        compatible = "X-Powers,axp15060";
        reg = <0x37>;
        regulators {
             sys_pd1_3v3_reg: DCDC1 {
                  regulator-name = "VDD_3V3";
                  regulator-min-microvolt = <3300000>;
                  regulator-max-microvolt = <3300000>;
                  regulator-always-on;
             };

             cnn0_pd_reg: DCDC2 {
                  regulator-name = "VCC_CNN0";
                  regulator-min-microvolt = <800000>;
                  regulator-max-microvolt = <1000000>;
                  regulator-enable-ramp-delay = <3000>;
             };

             cnn1_pd_reg: DCDC3 {
                  regulator-name = "VCC_CNN1";
                  regulator-min-microvolt = <800000>;
                  regulator-max-microvolt = <1000000>;
                  regulator-enable-ramp-delay = <3000>;
             };

             cpu_pd_reg: DCDC4 {
                  regulator-name = "VCC_CPU";
                  regulator-min-microvolt = <800000>;
                  regulator-max-microvolt = <1000000>;
                  regulator-always-on;
             };
```

```
            ddr_ao_1v1: DCDC5 {
                regulator-name = "DDR_AO_1V1";
                regulator-min-microvolt = <1100000>;
                regulator-max-microvolt = <1100000>;
                regulator-always-on;
            };

            ddr_pd_reg: DCDC6 {
                regulator-name = "VDD_DDR_PD";
                regulator-min-microvolt = <800000>;
                regulator-max-microvolt = <800000>;
                regulator-always-on;
            };

            core_ao_reg: BLDO1 {
                regulator-name = "VDD_CORE_AO";
                regulator-min-microvolt = <800000>;
                regulator-max-microvolt = <800000>;
                regulator-always-on;
            };

            sys_pd_1v8_reg: CLDO2 {
                regulator-name = "VDD_1V8";
                regulator-min-microvolt = <1800000>;
                regulator-max-microvolt = <1800000>;
                regulator-always-on;
            };

        };
    };

};

&cpu0 {
    cpu-supply = <&cpu_pd_reg>;
};

&cpu1 {
    cpu-supply = <&cpu_pd_reg>;
};

&cpu2 {
    cpu-supply = <&cpu_pd_reg>;
```

```
};

&cpu3 {
    cpu-supply = <&cpu_pd_reg>;
};

&cnn0 {
    cnn-supply = <&cnn0_pd_reg>;
};

&cnn1 {
    cnn-supply = <&cnn1_pd_reg>;
};
```

When coding the PMIC driver, it should be noted that the regulator-name option needs to be consistent with the Horizon Robotics platform. If it is inconsistent, some modules may not work properly.

When configuring DTS, it is generally to configure parameters such as "regulator-min-microvolt", "regulator-max-microvolt", and "regulator-always-on". For a detailed introduction to the other attributes of the regulator, please refer to the kernel documentation, which is located in the following directory：kernel/Documentation/devicetree/bindings/regulator/regulator.txt.

3 PMIC Driver Coding

The kernel regulator subsystem has completed most of the work of PMIC for us. The main task of writing a PMIC driver is to define related macros or structures according to the PMIC data sheet. The defined macro is mainly PMIC register, some operation bits and the number of steps of PMIC output voltage. This structure mainly includes the linear output voltage range of the PMIC. The macro or structure is mainly developed around the following macros.

```
#define AXP15060_REG(_name, _id, _linear, _step, _vset_mask, _enable)       \
    [ID_##_id] = {                    \
        .name           = _name,            \
        .id         = ID_##_id,     \
        .type           = REGULATOR_VOLTAGE,        \
        .ops            = &axp15060_ops,            \
        .n_voltages     = AXP15060_VOLTAGE_NUM##_step,      \
        .linear_ranges      = axp15060_voltage_ranges##_linear, \
        .n_linear_ranges    = ARRAY_SIZE(axp15060_voltage_ranges##_linear), \
        .vsel_reg       = AXP15060##_##_id##_VSET, \
        .vsel_mask      = AXP15060_VSET_MASK##_vset_mask,       \
        .enable_reg     = (AXP15060_ON_OFF_CTRL##_enable),  \
        .enable_mask        = AXP15060##_##_id##_ENA,           \
        .disable_val        = AXP15060_POWER_OFF,           \
        .owner          = THIS_MODULE,          \
    }

static const struct regulator_desc axp15060_regulators[] = {
    AXP15060_REG("DCDC1", DCDC1, 1, 20, 5, 1),
    AXP15060_REG("DCDC2", DCDC2, 2, 88, 7, 1),
    AXP15060_REG("DCDC3", DCDC3, 2, 88, 7, 1),
    AXP15060_REG("DCDC4", DCDC4, 2, 88, 7, 1),
    AXP15060_REG("DCDC5", DCDC5, 3, 69, 7, 1),
    AXP15060_REG("DCDC6", DCDC6, 4, 30, 5, 1),
    AXP15060_REG("BLDO1", BLDO1, 5, 27, 5, 2),
    AXP15060_REG("CLDO2", CLDO2, 5, 27, 5, 3),
};
```

This part of the content is the core part of the PMIC driver. The members in the axp15060 regulators structure are all the regulators just configured in DTS.

3.1 name field

For example, the corresponding relationship between the name field of AXP15060_REG and DTS configuration is:

AXP15060_REG("DCDC1", DCDC1, 1, 20, 5, 1) corresponds to DTS "sys_pd1_3v3_reg: DCDC1".

3.2 id field

The declaration of the id field is in the kernel/include/linux/regulator directory, as follows:

```
enum {
    ID_DCDC1,
    ID_DCDC2,
    ID_DCDC3,
    ID_DCDC4,
    ID_DCDC5,
    ID_DCDC6,
    ID_BLDO1,
    ID_CLDO2,
};
```

3.3 type field

The type field is fixed, and the type is REGULATOR_VOLTAGE, which is used for output voltage control.

3.4 ops field

The ops field is a collection of PMIC operations, which is also fixed and has been implemented by the regulator subsystem. These operating functions are mainly implemented in the kernel/drivers/regulator directory, mainly for mapping voltage, setting and reading output voltage.

```
static struct regulator_ops axp15060_ops = {
    .list_voltage       = regulator_list_voltage_linear_range,
    .map_voltage        = regulator_map_voltage_linear_range,
    .get_voltage_sel    = regulator_get_voltage_sel_regmap,
    .set_voltage_sel    = regulator_set_voltage_sel_regmap,
    .enable         = regulator_enable_regmap,
    .disable        = regulator_disable_regmap,
    .is_enabled     = regulator_is_enabled_regmap,
};
```

3.5 n_voltages field

The n_voltages field refers to the number of steps of the regulator's output, this needs to be obtained from the PMIC Datasheet.

```
/*
 * PF5024 voltage number
 */
```

```
#define AXP15060_VOLTAGE_NUM20      20
#define AXP15060_VOLTAGE_NUM27      27
#define AXP15060_VOLTAGE_NUM30      30
#define AXP15060_VOLTAGE_NUM36      36
#define AXP15060_VOLTAGE_NUM69      69
#define AXP15060_VOLTAGE_NUM88      88
```

REG 13： DC/DC 1 voltage control

Default： 12H

Reset： system reset

Bit	Description	R/W	Default
7-5	Reserved	RW	0
4-0	DCDC-1 voltage setting bit4-0, default is 3.3V: 1.5~3.4V， 100mV/step， 20steps	RW	10010

Figure3-1 Step number of DCDC1

3.6 linear_ranges field

The linear_ranges field is the output range of the regulator voltage, which mainly describes the relationship between the output voltage of the regulator and the number of steps, including the minimum voltage value, step value, etc. These parameters also need to be confirmed according to the datasheet.

The voltage value described in struct regulator_linear_range is in uv. 500000 means that the lowest voltage is 0.5v. 0, 70, 71, 87 means step, 10000, 20000 means step value.

```
static const struct regulator_linear_range axp15060_voltage_ranges2[] = {
    REGULATOR_LINEAR_RANGE(500000, 0, 70, 10000),
    REGULATOR_LINEAR_RANGE(1220000, 71, 87, 20000),
};
```

For example, DCDC2 corresponds to the implementation of the above structure, and its Datasheet description is shown in Figure 3-2:

REG 14： DC/DC 2 voltage control

Default： 3CH

Reset： system reset

Bit	Description	R/W	Default
7	Reserved	RW	0
6-0	DCDC-2 voltage setting bit6-0, default is 1.1V: 0.5~1.2V， 10mV/step， 71steps 1.22~1.54V， 20mV/step， 17steps	RW	0111100

Figure3-2 Step value of DCDC2

3.7 n_linear_ranges field

The n_linear_ranges field describes the number of elements in the struct regulator_linear_range structure.

3.8 vsel_reg field

The vsel_reg field is the register for setting the voltage and needs to be confirmed according to the datasheet. For example, the register address of DCDC6 setting voltage is 0x18.

#define AXP15060_DCDC6_VSET 0x18 //DCDC6 voltage set

REG 18：DC/DC 6 voltage control

Default：06H

Reset：system reset

Bit	Description	R/W	Default
7-5	Reserved	RW	0
4-0	DCDC-6 voltage setting bit4-0, default is 1.1V: 0.5~3.4V，100mV/step，30steps	RW	00110

Figure3-3 set voltage value on DCDC6

3.9 vsel_mask field

The vsel_mask field refers to the mask when setting the output voltage. In other words, which bits of the register are valid when setting and reading the output voltage, need to be confirmed according to the datasheet. For example, there are two types of vsel_mask for AXP15060, the lower 5 bits are valid or the lower 7 bits are valid.

```
#define AXP15060_VSET_MASK5      0x1F /*VSET - [4:0]*/
#define AXP15060_VSET_MASK7      0x7F /*VSET - [6:0]*/
```

REG 17: DC/DC 5 voltage control
Default: 2CH
Reset: system reset

Bit	Description	R/W	Default
7	Reserved	RW	0
6-0	DCDC-5 voltage setting bit6-0, default is 1.36V: 0.8~1.12V, 10mV/step, 33steps 1.14~1.84V, 20mV/step, 36steps	RW	0101100

REG 18: DC/DC 6 voltage control
Default: 06H
Reset: system reset

Bit	Description	R/W	Default
7-5	Reserved	RW	0
4-0	DCDC-6 voltage setting bit4-0, default is 1.1V: 0.5~3.4V, 100mV/step, 30steps	RW	00110

Figure3-4 mask bits of AXP15060 output voltage

3.10 enable_reg field

The enable_reg field refers to the register that controls the regulator output enable. Need to confirm according to the datasheet. For example, the register address that controls DCDC output enable is 0x10.

```
#define AXP15060_ON_OFF_CTRL1       0x10  //output power on-off control
#define AXP15060_ON_OFF_CTRL2       0x11  //output power on-off control
#define AXP15060_ON_OFF_CTRL3       0x12  //output power on-off control
```

REG 10: Output power on-off control 1
Default: 37H
Reset: system reset

Bit	Description		R/W	Default
7-6	Reserved		RW	0
5	DCDC-6 on-off control	0-off; 1-on	RW	1
4	DCDC-5 on-off control	0-off; 1-on	RW	1
3	DCDC-4 on-off control	0-off; 1-on	RW	0
2	DCDC-3 on-off control	0-off; 1-on	RW	1
1	DCDC-2 on-off control	0-off; 1-on	RW	1
0	DCDC-1 on-off control	0-off; 1-on	RW	1

Figure3-5 enable output voltage on DCDC

3.11 enable_mask field

The enable_mask field refers to the mask of the output voltage. It needs to be confirmed according to the datasheet. It means which bit of the enable register determines the enable of a certain output. This can be seen from Figure 3-5, for example, the enable bit of DCDC1 is bit0 of the 0x10 register。

```
#define AXP15060_DCDC6_ENA          0x20
#define AXP15060_DCDC5_ENA          0x10
#define AXP15060_DCDC4_ENA          0x08
#define AXP15060_DCDC3_ENA          0x04
#define AXP15060_DCDC2_ENA          0x02
#define AXP15060_DCDC1_ENA          0x01
```

3.12 disable_val field

The disable_val field indicates to disable the output of a certain voltage. If it is not strict, the field can be specified as 0.

3.13 PMIC DTS parse

After completing the above macro or structure, most of the pmic driver coding work has been completed. All that remains is to parse the data from DTS and probe.

To parse DTS is to parse the data under the regulators node in DTS in Section 2：

```
np = of_get_child_by_name(dev->of_node, "regulators");
if (!np) {
    dev_err(dev, "missing 'regulators' subnode in DT, %d\n", -EINVAL);
    return -EINVAL;
}
```

The specific function implementation can refer to the axp15060_pdata_from_dt function in the axp15060-regulator.c file。

3.14 PMIC probe

The main task completed by the probe function driven by pmic is to complete the registration of the regulator. The process is basically fixed, you can refer to the axp15060_pmic_probe function of the axp15060-regulator.c file.

```
/* Finally register devices */
```

```
for (i = 0; i < num_regulators; i++) {
    const struct regulator_desc *desc = ®ulators[i];
    struct regulator_config config = { };
    struct axp15060_regulator_data *rdata;
    struct regulator_dev *rdev;

    config.dev = dev;
    config.driver_data = axp15060;
    config.regmap = axp15060->regmap;

    rdata = axp15060_get_regulator_data(desc->id, pdata);
    if (rdata) {
        config.init_data = rdata->init_data;
        config.of_node = rdata->of_node;
    }

    rdev = devm_regulator_register(dev, desc, &config);
    if (IS_ERR(rdev)) {
        dev_err(dev, "failed to register %s\n", desc->name);
        return PTR_ERR(rdev);
    }
}
```