StarFive
赛昉科技

JH-7110 DevKit Power Management Developing and Porting Guide

Version: 1.0

Date: 2023/07/13

Doc ID: JH7110-DGEN-009

Legal Statements

Important legal notice before reading this documentation.

PROPRIETARY NOTICE

Contact Us

Address: Room 502, Building 2, No. 61 Shengxia Rd., China (Shanghai) Pilot Free Trade Zone, Shanghai, 201203, China

Website: http://www.starfivetech.com

Email:

- Sales: sales@starfivetech.com
- Support: support@starfivetech.com

Contents

List of Tables

List of Figures

Preface

About this guide and technical support information.

About this document

This document mainly provides the SDK developers with the programing basics and debugging know-how for the power management of the StarFive next generation SoC platform - JH-7110.

Audience

This document mainly serves the power management relevant driver developers. If you are developing other modules, place a request to your sales or support consultant for our complete documentation set on JH-7110.

Revision History

Table 0-1 Revision History

Version	Released	Revision
1.0	2023/07/13	The First Official Release.

Notes and notices

The following notes and notices might appear in this guide:

- **Tip:**
  Suggests how to apply the information in a topic or step.
- **Note:**
  Explains a special case or expands on an important point.
- **Important:**
  Points out critical information concerning a topic or step.
- **CAUTION:**
  Indicates that an action or step can cause loss of data, security problems, or performance issues.
- **Warning:**
  Indicates that an action or step can result in physical harm or cause damage to hardware.

1. Introduction

The JH-7110 DevKit Power Management Subsystem provides users the ability to configure the device in various power modes and profiles, for the purposes of current consumption measurements. The user can choose to work in the Interactive (simple) mode or to use the advanced mode, where they can change the define values in the source code in order to control the behavior of the application. The chapter has recommended guidelines on how to optimize the power consumption for the different power profiles.

1.1. Overview

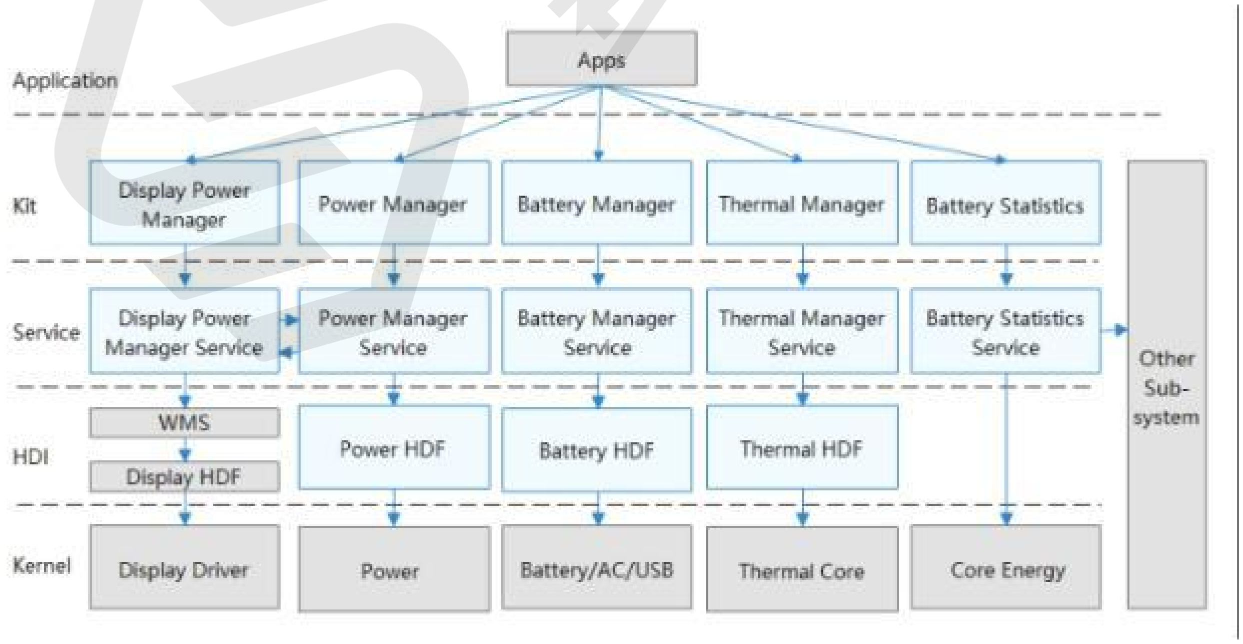
Each power supply in the system is generally divided into two types: DCDC (Direct Current) and LDO (Low Dropout Regulator). The overall features of the two power supplies are as follows:

- DCDC: It has the advantages of high efficiency, wide input voltage range, strong driving ability and low static current.
- LDO: It is a linear regulator and can only be used in buck applications, which means the output voltage must be less than the input voltage. It is with the advantages of good stability, fast load response, and small output ripple.

1.2. Block Diagram

The following is the block diagram of the power management subsystem:

Figure 1-1 Block Diagram



1.3. Power-Up Sequence

JH-7110 DevKit has 3 power supplies, VDD, AVDL, and AVDH. VDD and AVDL are both core power supplies and while separate pins are used for noise isolation purposes.

The recommended power sequence is that from digital core voltage to analog I/O voltage and from low voltage level to high voltage level. This is not considered a constraint, but instead, a guideline, as it could result in the best-case operating scenario, where the leakage currents during power up are kept to a minimum.

The following diagram shows the recommended power-on sequence of different power groups.

Figure 2-9 Power Up Sequence

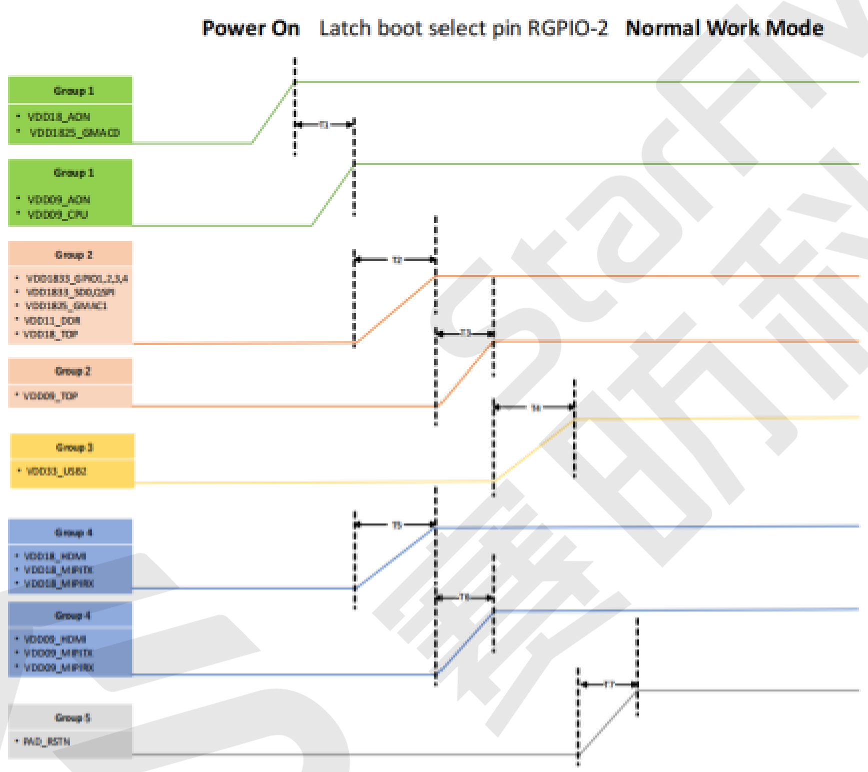


Table 1-1 Power Up Sequence Intervals

Sequence	Interval
T1	100 us
T2	300-500 us
T3	50-100 us
T4	300-500 us
T5	300-500 us
T6	50-100 us
T7	>10 ms

2. Configuration

The following is the configuration of the JH-7110 DevKit power management subsystem.

2.1. Drive and Menuconfig

- Drive file:

```
drivers/regulator/axp15060-regulator.c
```

- DTS file can refer to:

```
arch/riscv/boot/dts/starfive/jh7110-devkits.dts
```

- The corresponding macro configuration in the menuconfig:

```
CONFIG_REGULATOR_APX15060
```

2.2. DTS Configuration

The configuration of DTS includes: I2C mount, body, regulator, rtc, and other parts. The following code block shows an example of the device tree source code of the "&i2c5" and "pmic_axp15060: axp15060_reg@36 ".

```
&i2c5 {
    clock-frequency = <100000>;
    i2c-sda-hold-time-ns = <300>;
    i2c-sda-falling-time-ns = <510>;
    i2c-scl-falling-time-ns = <510>;
    auto_calc_scl_lhcnt;
    pinctrl-names = "default";
    pinctrl-0 = <&i2c5_pins>;
status = "okay";

    pmic_axp15060: axp15060_reg@36 {
        compatible = "starfive,axp15060-regulator";
        reg = <0x36>;

        regulators {
            cpu_vdd: DCDC2 {
                regulator-boot-on;
                regulator-always-on;
                regulator-compatible = "cpu_vdd";
                regulator-name = "cpu_vdd";
                regulator-min-microvolt = <500000>;
                regulator-max-microvolt = <1540000>;
            };
            emmc_vdd: BLDO3 {
                regulator-boot-on;
                regulator-always-on;
                regulator-compatible = "emmc_vdd";
                regulator-name = "emmc_vdd";
                regulator-min-microvolt = <1800000>;
                regulator-max-microvolt = <1800000>;
            };
            vcc_3v3: BLDO4 {
                regulator-boot-on;
                regulator-always-on;
                regulator-compatible = "vcc_3v3";
                regulator-name = "vcc_3v3";
                regulator-min-microvolt = <3300000>;
                regulator-max-microvolt = <3300000>;
            };
        };
    };
};
```

The following list provides explanations for the parameters included in the above code block.

- **clock-frequency:** The frequency of the above clock.
- **pinctrl-names:** The name of the pinctrl.
- **status:** The work status of the i2c.
- **compatible:** Compatibility information, used to associate the driver and its target device.
- **reg:** Register base address of i2c.
- **regulators:** Regulators that needed to be controlled.
- **regulator-boot-on:** The regulator that is enabled by the bootloader or firmware.
- **regulator-always-on:** Never perform the power-off operation on the regulator.
- **regulator-min-microvolt:** The minimum voltage that the user can set.
- **regulator-max-microvolt:** The maximum voltage that the user can set.

2.2.1. DVFS

This section includes the following two parts:

CPU DVFS Configuration

Dynamic Voltage and Frequency Scaling is a framework to change the frequency and/or operating voltage of a processor(s) based on system performance requirements at the given point of time. Frequency scaling is achieved using **CPUFreq** framework.

OPP List

Operating Performance Point (OPP) is a tuple consisting a frequency value and voltage required to run at the frequency. OPP table contains OPPs with a cpu/device name they are applicable to and an availability flag. OPP information of each device is added to OPP list.

Linux4.4 kernel puts the frequency and voltage-related configurations in `device tree`, and we call the node composed of these configuration information as OPP Table. OPP Table nodes include OPP nodes describing frequency and voltage, leakage related configuration attributes, PVTM related configuration attributes, etc.

The following code block is the JH-7110 DevKit opp-table:

```
cluster0_opp: opp-table-0 {
            compatible = "operating-points-v2";
            opp-shared;
            opp-375000000 {
                    opp-hz = /bits/ 64 <375000000>;   # Unit: Hz
                    opp-microvolt = <800000>;    # Unit: uV
            };
            opp-500000000 {
                    opp-hz = /bits/ 64 <500000000>;   # Unit: Hz
                    opp-microvolt = <800000>;   # Unit: uV
            };
            opp-750000000 {
                    opp-hz = /bits/ 64 <750000000>;   # Unit: Hz
                    opp-microvolt = <800000>;    # Unit: uV
                    opp-suspend;
            };
            opp-1500000000 {
                    opp-hz = /bits/ 64 <1500000000>;   # Unit: Hz
                    opp-microvolt = <1040000>;    # Unit: uV
            };

            /* CPU opp table for 1.25GHz */
            opp-312500000 {
                    opp-hz = /bits/ 64 <312500000>;    # Unit: Hz
                    opp-microvolt = <800000>;    # Unit: uV
            };
            opp-417000000 {
                    opp-hz = /bits/ 64 <417000000>;    # Unit: Hz
                    opp-microvolt = <800000>;     # Unit: uV
            };
            opp-625000000 {
                    opp-hz = /bits/ 64 <625000000>;    # Unit: Hz
                    opp-microvolt = <800000>;    # Unit: uV
                    opp-suspend;
            };
            opp-1250000000 {
                    opp-hz = /bits/ 64 <1250000000>;    # Unit: Hz
                    opp-microvolt = <1000000>;    # Unit: uV
            };
    };
```

2.2.2. Charge Part

JH-7110 DevKit supports the monitoring of charging, discharging, battery and charging status, including the update and reporting of status, and also includes shutdown charging.

3. Interface Description

JH-7110 DevKit provides the following application interfaces for power management engine.

3.1. of_regulator_match

The interface has the following parameters.

- **Synopsis:**

```
int of_regulator_match(struct device *dev, struct device_node *node,
         struct of_regulator_match *matches,
         unsigned int num_matches);
```

- **Description:** During the procedure of probe for the provider, every component of the match table will be compared with all the regulators that are actually used by name. And all matches will be initialized according to device node.
- **Parameters:**
    - **dev:** The target i2c device
    - **node:** Regulator DTS node
    - **matches:** The defined regulator information
    - **num_matches:** The defined regulator number
- **Return:** Number of matches.

3.2. devm_regmap_init_i2c

The interface has the following parameters.

- **Synopsis:**

```
# define devm_regmap_init_i2c(i2c, config)    \
 __regmap_lockdep_wrapper(__devm_regmap_init_i2c, #config, \
    i2c, config)
```

- **Description:** This interface is to map the PMIC address.
- **Parameters:**
    - **i2c:** The target i2c device
    - **config:** The PMIC regmap information.

3.3. devm_regulator_register

The interface has the following parameters.

- **Synopsis:**

```
struct regulator_dev *devm_regulator_register(struct device *dev,
        const struct regulator_desc *regulator_desc,
        const struct regulator_config *config);
```

- **Description:** Register regulator.
- **Parameter:**
  - **dev:** The target i2c device
  - **regulator_desc:** The regulator information provided by PMIC
  - **config:** The configuration information of regulator.
- **Return:**
  - **Success:** 0.
  - **Fail:** Error code.

4. Standby and Wakeup

This chapter displays the standby mode and wakeup method of JH-7110 DevKit.

4.1. Standby Mode

Follow the steps below to enter the standby mode:

1. Configure PMU to turn off all internal digital power except CPU.
2. The system clock source is switched to the OSC (`clk_cpu_root`, `clk_bus_root`), because the PLL power will be turned off in the next step.
3. Configure the PMU wakeup source and the modules that need to be powered up, RTCWOL/ RGPIO.
4. Configure the PMU power-off CPU. The PMU will automatically pull down RGPIO2 and turn off all output power of the PMIC: VDD18, VDDO9VDD33, VDDQ11.
5. Enter standby mode.

4.2. Wakeup from WiFi and Key

This section shows two ways to wakeup JH-7110 DevKit.

4.2.1. Wakeup via WiFi

- Wakeup via WiFi:
- Awake function call process:

4.2.2. Wakeup via Key

- Wakeup via GPIO:
- Awake function call process:

```
echo mem > /sys/power/state
|--late_initcall_sync(software_resume)
|  |--swsusp_check  #  Check is the image is valid of not from
 swsusp_resume_device device. If it is invalid, the following process
 will not be excuted, you can exit and proceed with the normal startup
 process.
|  |--pm_prepare_console
|  |--__pm_notifier_call_chain
|   |--freeze_processes  # Freeze the processes of the current user
 space
|   |-load_image_and_restore
|   |   |--lock_device_hotplug
|   |   |--create_basic_memory_bitmaps
|   |   |--swsusp_read
|   |   |--swsusp_close
|   |   |--hibernation_restore  # Restore from the hibernate image
|   |   |   |--resume_target_kernel
|   |   |--swsusp_free
|   |   |--free_basic_memory_bitmaps
|   |   |--unlock_device_hotplug
|   |--thaw_processes  # Thaw the processes of the user sapce. And at
 this point, the process being thawed and restored is a process in
 Hibernate.
|   |--__pm_notifier_call_chain
|   |--pm_restore_console
|   |--swsusp_close
```

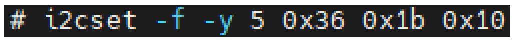
5. Debug

The drivers involved in PMIC are not complicated in use, the focus is reflected in the final register setting. Therefore, the commonly used debug method is to directly check the register of JH-7110 DevKit through the following `i2c`:

- Read register:

```
I2cget -f -y 5 0x36 addr
```

Figure 5-1 Example Output

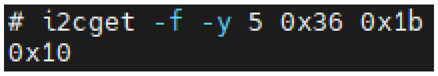


```
# i2cset -f -y 5 0x36 0x1b 0x10
```

- Write register:

```
I2cset  -f -y 5 0x36 addr data
```

Figure 5-2 Example Output:



```
# i2cget -f -y 5 0x36 0x1b
0x10
```