Linux TWI

开发指南

版本号: 1.9
发布日期: 2025.03.18

版本历史

版本号	日期	制/修订人	内容描述
1.0	2022.07.27	XAA0248	初始版本。
1.1	2022.11.28	AWA1979	1. 更新文档模板。2. 增加 Linux-5.15 内核版本支持。
1.2	2023.1.16	XAA0248	1. 新增模块使用章节，介绍每个规格的配置使用方法。2. 完善 i2c-tools 使用介绍。
1.3	2023.3.17	XAA0309	更新全文格式。
1.4	2023.5.30	XAA0309	更新 aliases。
1.5	2023.12.21	XAA0311	添加原理及工作流程描述。
1.6	2024.3.15	XAA0311	对文档结构及内容进行优化。
1.7	2024.10.22	AWA2215	增加 Linux-5.4 内核版本支持，调整文档结构。
1.8	2024.11.25	AWA2215	增加 Linux-6.6 内核版本支持, 增加 apb 总线配置
1.9	2025.03.18	AWA2155	增加 TWI DVFS 新功能特性支持

版权所有 © 珠海全志科技股份有限公司。保留一切权利

目　录

1 前言 1
1.1 文档简介 1
1.2 目标读者 1
1.3 适用范围 1
1.4 文档约定 1
1.4.1 标志说明 1
1.4.2 地址与数据描述方法约定 2
1.4.3 数值单位约定 2
1.5 相关术语介绍 2
1.5.1 硬件术语 2
1.5.2 软件术语 3

2 模块介绍 4
2.1 模块规格介绍 4
2.2 源码模块结构 4
2.3 模块配置介绍 4
2.3.1 device tree 配置 5
2.3.2 board.dts 板级配置 5
2.3.3 kernel menuconfig 配置 6
2.3.4 工作模式 8
2.3.4.1 master 模式 8
2.3.4.2 slave 模式 9
2.4 驱动框架介绍 12

3 软件功能介绍 14
3.1 函数初始化流程 15
3.2 数据发送流程 16
3.3 delay_init 处理流程 18
3.4 休眠唤醒处理流程 18
3.5 关机处理流程 19
3.6 DVFS 处理流程 19

4 模块接口说明 20
4.1 内部接口 20
4.1.1 sunxi_twi_xfer() 20
4.1.2 sunxi_twi_handler() 20
4.2 DVFS 接口 20
4.2.1 sunxi_twi_dvfs_enable_all() 20
4.2.2 sunxi_twi_dvfs_disable_all() 21

版权所有 © 珠海全志科技股份有限公司。保留一切权利

4.2.3 sunxi_twi_dvfs_enable() . . . 21
4.2.4 sunxi_twi_dvfs_disable() . . . 21
4.2.5 sunxi_twi_dvfs_set_slave_addr() . . . 21
4.2.6 sunxi_twi_dvfs_set_interval() . . . 21
4.2.7 sunxi_twi_dvfs_chan_init() . . . 22
4.2.8 sunxi_twi_dvfs_set_chan_reg() . . . 22
4.2.9 sunxi_twi_dvfs_set_chan_mask() . . . 22
4.2.10 sunxi_twi_dvfs_set_chan_rb() . . . 22
4.2.11 sunxi_twi_dvfs_set_chan_prio() . . . 22
4.3 i2c-core 接口 . . . 23
4.3.1 i2c_transfer . . . 23
4.3.2 i2c_master_recv . . . 23
4.3.3 i2c_master_send . . . 24
4.4 i2c 用户态调用接口 . . . 24
4.4.1 i2cdev_open . . . 24
4.4.2 i2cdev_read . . . 25
4.4.3 i2cdev_write . . . 25
4.4.4 i2cdev_ioctl . . . 26

5 用户态 TWI 功能开发 27
5.1 功能概述 . . . 27
5.2 开发流程 . . . 27
5.3 注意事项 . . . 28
5.4 编程示例 . . . 28

6 内核态 TWI 功能开发 29
6.1 功能概述 . . . 29
6.2 开发流程 . . . 29
6.3 注意事项 . . . 30
6.4 编程示例 . . . 30

7 调试方法 31
7.1 i2c-tools 调试工具 . . . 31
7.1.1 i2cdetect . . . 31
7.1.1.1 参数说明 . . . 31
7.1.1.2 使用示例 . . . 31
7.1.2 i2cdump . . . 32
7.1.2.1 参数说明 . . . 32
7.1.2.2 使用示例 . . . 32
7.1.3 i2cset . . . 33
7.1.3.1 参数说明 . . . 33
7.1.3.2 使用示例 . . . 34
7.1.4 i2cget . . . 34
7.1.4.1 参数说明 . . . 34

版权所有 © 珠海全志科技股份有限公司。保留一切权利

7.1.4.2 使用示例 34
7.1.5 i2ctransfer 35
7.1.5.1 参数说明 35
7.1.5.2 使用示例 35

8 FAQ 37
8.1 TWI 数据未完全发送 37
8.2 TWI 起始信号无法发送 37
8.3 TWI 终止信号无法发送 38
8.4 TWI 传送超时 38

插 图

图 2-1 Allwinner BSP 6
图 2-2 Device Drivers 7
图 2-3 TWI Drivers 7
图 2-4 I2C Support for Allwinner SoCs 8
图 2-5 slave 模式配置 10
图 2-6 驱动加载成功 11
图 2-7 TWI 模块结构框图 12
图 3-1 模块功能结构 14
图 3-2 模块初始化流程 15
图 3-3 数据发送流程 16
图 3-4 cpu 传输 17
图 7-1 i2cdetect 命令结果 32
图 7-2 i2cdump 结果分析 33
图 7-3 i2cset 运行结果 34
图 7-4 i2cget 运行结果 35
图 7-5 i2ctransfer 运行结果 35
图 7-6 i2ctransfer 读结果 36

版权所有 © 珠海全志科技股份有限公司。保留一切权利

1 前言

1.1 文档简介

介绍 Sunxi 平台上 TWI 驱动接口与调试方法，为 TWI 模块开发提供参考。

1.2 目标读者

TWI 模块内核层以及应用层的开发、维护人员。

1.3 适用范围

表 1-1: 适用产品列表

内核版本	驱动文件
Linux-5.10	twi-sunxi.c
Linux-5.15	twi-sunxi.c
Linux-5.4	twi-sunxi.c
Linux-6.6	twi-sunxi.c

1.4 文档约定

1.4.1 标志说明

⚠ **注意**

- **提醒操作中应注意的事项。不当的操作可能会损坏器件，影响可靠性、降低性能等。**

说明

为准确理解文中指令、正确实施操作而提供的补充或强调信息。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

💡技巧

一些容易忽视的小功能、技巧。了解这些功能或技巧能帮助解决特定问题或者节省操作时间。

1.4.2 地址与数据描述方法约定

本文档在描述地址、数据时遵循如下约定：

表 1-2: 地址与数据描述方法约定

符号	例子	说明
0x	0x0200，0x79	地址或数据以 16 进制表示。
0b	0b010，0b00 000 111	数据采用二进制表示 (寄存器描述除外)。
X	00X，XX1	数据描述中，X 代表 0 或 1。 例如，00X 代表 000 或 001；XX1 代表 001，011，101 或 111。

1.4.3 数值单位约定

本文档在描述数据容量（如 NAND 容量）时，单位词头代表的是 1024 的倍数；描述频率、数据速率等时则代表的是 1000 的倍数。具体如下：

表 1-3: 数值单位约定

类型	符号	对应数值
数据容量（如 NAND 容量）	1 K	1024
	1 M	1 048 576
	1 G	1 073 741 824
频率，数据速率等	1 k	1000
	1 M	1 000 000
	1 G	1 000 000 000

1.5 相关术语介绍

1.5.1 硬件术语

表 1-4: 硬件术语

术语	解释说明
TWI	Two Wire Interface，全志平台兼容 I2C 标准协议的总线控制器
DVFS	Dynamic Voltage Frequency Scaling，动态电压频率调节

1.5.2 软件术语

表 1-5: 软件术语

术语	解释说明
Sunxi	全志科技使用的 linux 开发平台。
I2C_dapter	linux 内核中 I2C 总线适配器的抽象定义；又称为 IIC 总线的控制器。
I2C_algorithm	linux 内核中 I2C 总线通信的抽象定义。描述 I2C 总线适配器与 I2C 设备之间的通信方法。
I2C Client	linux 内核中 I2C 设备的抽象定义。
I2C Driver	linux 内核中 I2C 设备驱动的抽象定义。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

2 模块介绍

2.1 模块规格介绍

全志公司的 twi 总线兼容 i2c 总线协议，是一种简单、双向二线制同步串行总线。它只需要两根线即可在连接于总线上的器件之间传送信息。TWI 控制器支持的标准通信速率为 100kbps，最高通信速率可以达到 400kbps。全志的 twi 控制器支持以下功能：

- 支持主机模式和从机模式；
- 主机模式下支持 dma 传输；
- 主机模式下在多个主机的模式下支持总线仲裁；
- 主机模式下支持时钟同步，位和字节等待；
- 从机模式下支持地址检测中断；
- 支持 7bit 从机地址和 10bit 从机地址；
- 支持常规的 i2c 协议模式和自定义传输模式。
- 主机模式下支持硬件动态电压频率调节（DVFS，是否支持请查看 datasheet）

sunxi 平台支持多路 TWI，包含 TWI 与 S_TWI。

2.2 源码模块结构

TWI 总线驱动的源代码位于内核在 sdk/bsp/drivers/twi/目录下：

```
sdk/bsp/drivers/twi/
├── twi-sunxi.c        /* Sunxi平台的TWI控制器驱动代码 */
```

2.3 模块配置介绍

在不同的 Sunxi 硬件平台中，TWI 控制器的数目不同；但对于同一块板子上的每一个 TWI 控制器来说, 模块配置类似，本小节展示 Sunxi 平台上的 TWI0 控制器配置（其他 TWI 控制器配置类似）。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

2.3.1 device tree 配置

在 soc 级的 **dtsi** 文件中提炼了内存基地址、中断控制、时钟等共性信息，是该类芯片所有平台的模块配置，soc 级的 **dtsi** 文件的路径为：sdk/bsp/configs/{linux-ver}/{CHIP}.dtsi(CHIP 为研发代号，如 sun50iw10p1 等)，TWI 总线的设备树配置如下所示：

```
twi0: twi@5002000 {
    #address-cells = <1>;
    #size-cells = <0>;
    compatible = "allwinner,sun50i-twi";      /* 具体的设备，用于驱动和设备的绑定 */
    device_type = "twi0";            /* 设备节点名称，用于sys_config.fex匹配 */
    reg = <0x0 0x02502000 0x0 0x3ff>;      /* 设备使用的地址 */
    interrupts = <GIC_SPI 10 IRQ_TYPE_LEVEL_HIGH>; /* 设备使用的中断：中断号、中断类型 */
    clocks = <&ccu CLK_TWI0>, <&ccu CLK_APB1>;      /* 设备使用的时钟 */
    clock-names = "bus", "apb";          /* 使用的时钟名 */
    resets = <&ccu RST_BUS_TWI0>;        /* 设备使用的复位时钟 */
    dmas = <&dma 43>, <&dma 43>;         /* 设备使用的dma通道号 */
    dma-names = "tx", "rx";          /* 设备使用的dma通道名 */
    twi_dvfs_enable;              /* 使能DVFS功能，平台是否支持DVFS功能请查看datasheet */
    status = "disabled";            /* 设备是否使用，dtsi中设为disabled，在board.dts中设置 */
};
```

为了在 TWI 总线驱动代码中区分每一个 TWI 控制器，需要在 Device Tree 中的 aliases 节点中为每一个 TWI 节点指定别名：

```
    aliases {
        twi0 = &twi0;
        twi1 = &twi1;
        twi2 = &twi2;
        twi3 = &twi3;
        ...
    };
```

别名形式为字符串 “twi” 加连续编号的数字，在 TWI 总线驱动程序中可以通过 of_alias_get_id() 函数获取对应 TWI 控制器的数字编号，从而区分每一个 TWI 控制器。

2.3.2 board.dts 板级配置

board.dts 用于保存每一个板级平台的设备信息（如 demo 板，perf1 板，ver1 板等等），里面的配置信息会覆盖上面的 device tree 默认配置信息。board.dts 的路径为 sdk/device/config/chips/$SoC/configs/${BOARD}/${KERNEL_VER}/board.dts。

对应 board.dts 里面 TWI0 的具体配置如下：

```
&twi0 {
    clock-frequency = <400000>;    /* TWI控制器的时钟频率 */
    pinctrl-names = "default", "sleep"; /* 设备使用的pin脚名称 */
    pinctrl-0 = <&twi0_pins_default>;  /* 设备使用的pin脚配置（default） */
    pinctrl-1 = <&twi0_pins_sleep>;   /* 设备使用的pin脚配置（sleep） */
    twi_drv_used = <1>;       /* 是否启用drv模式传输数据：0-engine模式，1-drv模式 */
    twi-supply = <&reg_dcdc1>;   /* regulator相关配置 */
    status = "okay";        /* 设备是否使用 */
```

```
  eeprom@50 {          /* eeprom从设备 */
    compatible = "atmel,24c16"; /* eeprom从设备对应的设备驱动compatible，用于驱动和设备的绑定 */
    reg = <0x50>;        /* eeprom从设备地址为0x50 */
    status = "disabled";      /* eeprom从设备status */
  };
};
```

其中，TWI 速率由 “clock-frequency” 属性配置，最大支持 400K。

对于 TWI 设备，可以把设备节点填充作为 Device Tree 中相应 TWI 控制器的子节点。TWI 控制器驱动的 probe 函数透过 of_i2c_register_devices()，自动展开作为其子节点的 TWI 设备。

对于 twi0 中引用的 pin 口，具体的配置如下：

```
  twi0_pins_a: twi0@0 {
    pins = "PB0", "PB1";      /* 使用的引脚 */
    function = "twi0";     /* 引脚功能 */
    drive-strength = <10>;    /* 引脚配置 */
    bias-pull-up;
  };

  twi0_pins_b: twi0@1 {
    pins = "PB0", "PB1";      /* 使用的引脚 */
    function = "gpio_in";     /* 引脚功能 */
  };
```

2.3.3 kernel menuconfig 配置

在根目录中执行./build.sh menuconfig，选择 Allwinner BSP 选项进入下一级配置，如下图所示：

```
        Allwinner BSP  --->
        General setup  --->
    [*] Support DMA zone
    [*] Support DMA32 zone
        Platform selection  --->
        Kernel Features  --->
        Boot options  --->
        Power management options  --->
        CPU Power Management  --->
        Firmware Drivers  --->
    [ ] Virtualization  ----
    [ ] ARM64 Accelerated Cryptographic Algorithms  ----
        General architecture-dependent options  --->
    [*] Enable loadable module support  --->
    [*] Enable the block layer  --->
        IO Schedulers  --->
        Executable file formats  --->
        Memory Management options  --->
    [*] Networking support  --->
        Device Drivers  --->
        File systems  --->
        Security options  --->
    -*- Cryptographic API  --->
        Library routines  --->
        Kernel hacking  --->
```

图 2-1: Allwinner BSP

ALLWINNER®

选择 Device Drivers 选项进入下一级配置，如下图所示：

```
Platform Selection  --->
Device Drivers  --->
NAND Drivers  --->
GPU Drivers  --->
```

图 2-2: Device Drivers

选择 TWI Drivers 选项，进入下一级配置，如下图所示：

```
CE Drivers  --->
SMC Drivers  --->
SID Drivers  --->
SECURE STORAGE Drivers  --->
DRM HEAP Drivers  --->
WUPIO Drivers  --->
PCIe Drivers  --->
PHY Drivers  --->
Emac Drivers  --->
Gmac Drivers  --->
Stmmac Drivers  --->
USB Drivers  --->
TWI Drivers  --->
SPI Drivers  --->
SPI NG Drivers  --->
LED Drivers  --->
PWM Drivers  --->
IR-TX Drivers  --->
IR-RX Drivers  --->
General Purpose ADC Drivers  --->
Low Rate ADC Drivers  --->
Resistive Touch Panel Drivers  --->
Input Device Drivers  --->
```

图 2-3: TWI Drivers

选择 I2C Support for Allwinner SoCs 选项，可选择直接编译进内核，也可编译成模块。如下图所示：

ALLWINNER®

```
<*> I2C Support for Allwinner SoCs
[ ]    SUNXI TWI hardware init delay
[ ]    SUNXI TWI DYNAMIC DEBUG
```

图 2-4: I2C Support for Allwinner SoCs

2.3.4 工作模式

2.3.4.1 master 模式

TWI 控制器默认工作在 master 模式，无需其他额外配置。

2.3.4.1.1 engine 模式配置

说明

engine 模式：传输过程中的每个步骤都会产生一次中断，由软件控制下一次的要进行的操作

技巧

engine 是中断模式驱动通信，占用 cpu 资源，可做 slave 模式，时钟延展，**一般作 slave 或者 slave 设备需要时钟延展特性时才使用 engine 模式**

工作在 engine 模式，主要是配置 twi_drv_used = <0> 属性，TWI 模块在 **dts 文件**中的具体配置如下：

```
&twi0 {
    clock-frequency = <400000>;    /* TWI控制器的时钟频率 */
    pinctrl-names = "default", "sleep"; /* 设备使用的pin脚名称 */
    pinctrl-0 = <&twi0_pins_default>;  /* 设备使用的pin脚配置（default） */
    pinctrl-1 = <&twi0_pins_sleep>;    /* 设备使用的pin脚配置（sleep） */
    twi_drv_used = <0>;       /* 是否启用drv模式传输数据：0-engine模式 */
    no_suspend = <1>;         /* 默认为0，无需配置；当pmu依赖twi时需要置1，标识twi在休眠唤醒阶段不休眠 */
    status = "okay";          /* 设备是否使用 */
};
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

2.3.4.1.2 drv 模式配置

说明

drv 模式：由硬件自动控制发送完整的传输，过程中无中断，仅在完成/异常时产生一次中断

技巧

drv 模式对 cpu 占用低，稳定性高，**一般 master 模式下，默认使用 drv 模式**

工作在 drv 模式，主要是配置 twi_drv_used = <1> 属性，TWI 模块在 **dts 文件**中的具体配置如下：

```
&twi0 {
    clock-frequency = <400000>;    /* TWI控制器的时钟频率 */
    pinctrl-names = "default", "sleep"; /* 设备使用的pin脚名称 */
    pinctrl-0 = <&twi0_pins_default>;  /* 设备使用的pin脚配置（default） */
    pinctrl-1 = <&twi0_pins_sleep>;    /* 设备使用的pin脚配置（sleep） */
    twi_drv_used = <1>;       /* 是否启用drv模式传输数据：1-drv模式 */
    no_suspend = <1>;        /* 默认为0，无需配置；当pmu依赖twi时需要置1，标识twi在休眠唤醒阶段不休眠 */
    status = "okay";         /* 设备是否使用 */
};
```

2.3.4.1.3 DVFS 功能配置

DVFS 功能必须工作在 twi drv 模式下，需要打开宏配置项：CONFIG_AW_TWI_DVFS。

该功能设备树参考 2.3.1 章节配置 soc 级 dtsi，参考上一小节配置板级 dts。

2.3.4.1.4 传输速率配置

工作在标准模式，时钟频率为 100k，配置 clock-frequency = <100000>;

工作在快速模式，时钟频率为 400k，主要配置 clock-frequency = <400000>。

TWI 模块在 **dts 文件**中的具体配置如下：

```
&twi0 {
    clock-frequency = <400000>;/* TWI控制器的时钟频率，默认为400k */
    ...
};
```

2.3.4.2 slave 模式

TWI 控制器支持作为从机使用，但是默认工作在 master 模式，slave 模式需要特殊配置。

ALLWINNER®

2.3.4.2.1 menuconfig 配置

需要打开以下两个宏配置选项。

CONFIG_TWI_SLAVE, CONFIG_TWI_SLAVE_EEPROM should be configured as y or M。

```
--- I2C support
[*]   Enable compatibility bits for old user-space
-*-   I2C device interface
< >   I2C bus multiplexing support
[*]   Autoselect pertinent helper modules
      I2C Hardware Bus support  --->
< >   I2C/SMBus Test Stub
[*]   I2C slave support
<*>     I2C eeprom slave driver
< >     I2C eeprom testunit driver (NEW)
[ ]   I2C Core debugging messages
[ ]   I2C Algorithm debugging messages
[ ]   I2C Bus debugging messages
```

图 2-5: slave 模式配置

ALLWINNER®

2.3.4.2.2 dts 配置

slave 模式时 TWI 控制器需要配置为 engine 模式，并且在 dts 中注册为 slave，具体配置如下：

TWI 模块在 **dts 文件**中的具体配置如下：

```
&twi0 {
    ...
    twi_drv_used = <0>;  /* slave must work in engine-mode */
    status = "okay";
    slave {
        compatible = "slave-24c02";  /* Slave device drivers provided by the kernel */
        reg = <0x50>;  /* slave_addr，可以修改 */
        status = "okay";
    };
};
```

按上述配置完成后，该 TWI 控制器即注册为一个从设备，设备地址为上述定义的 slave_addr，该 TWI 控制器与其他 master 的 sck 和 sda 连接后可以进行正常通信。

驱动加载成功后，如下 log 显示：

```
Line 19948: [    2.993789][  T116] sunxi-twi 7081400.s_twi0: supply twi not found, using dummy regulator
Line 19948: [    2.993789][  T116] sunxi-twi 7081400.s_twi0: supply twi not found, using dummy regulator
Line 19986: [    3.323057][  T116] input: axp2202-pek as /devices/platform/soc@3000000/7081400.s_twi0/i2c-6/6-0034/axp2101-pek.0/input/input0
Line 20005: [    4.565604][  T116] sunxi:twi_sunxi@7081400.s_twi0[INFO]: v2.6.6 probe success
Line 20005: [    4.565604][  T116] sunxi:twi_sunxi@7081400.s_twi0[INFO]: v2.6.6 probe success
Line 20088: [    5.527929][  T116] sunxi:twi_sunxi@2502000.twi0[INFO]: v2.6.6 probe success
Line 20088: [    5.527929][  T116] sunxi:twi_sunxi@2502000.twi0[INFO]: v2.6.6 probe success
Line 20090: [    5.552407][  T116] sunxi:twi_sunxi@2502400.twi1[INFO]: v2.6.6 probe success
Line 20090: [    5.552407][  T116] sunxi:twi_sunxi@2502400.twi1[INFO]: v2.6.6 probe success
Line 20146: [    6.208621][  T116] sunxi:twi_sunxi@2502800.twi2[INFO]: v2.6.6 probe success
Line 20146: [    6.208621][  T116] sunxi:twi_sunxi@2502800.twi2[INFO]: v2.6.6 probe success
Line 20147: [    6.217453][  T116] sunxi:twi_sunxi@2502c00.twi3[INFO]: v2.6.6 probe success
Line 20147: [    6.217453][  T116] sunxi:twi_sunxi@2502c00.twi3[INFO]: v2.6.6 probe success
Line 20148: [    6.226297][  T116] sunxi:twi_sunxi@2503000.twi4[INFO]: v2.6.6 probe success
Line 20148: [    6.226297][  T116] sunxi:twi_sunxi@2503000.twi4[INFO]: v2.6.6 probe success
Line 20150: [    6.245680][  T116] sunxi:twi_sunxi@2503400.twi5[INFO]: v2.6.6 probe success
Line 20150: [    6.245680][  T116] sunxi:twi_sunxi@2503400.twi5[INFO]: v2.6.6 probe success
Line 21163: [  202.003537][    T1] sunxi:twi_sunxi@2503400.twi5[INFO]: xfer completed, shutdown twi directly
Line 21163: [  202.003537][    T1] sunxi:twi_sunxi@2503400.twi5[INFO]: xfer completed, shutdown twi directly
Line 21163: [  202.003537][    T1] sunxi:twi_sunxi@2503400.twi5[INFO]: xfer completed, shutdown twi directly
Line 21164: [  202.013161][    T1] sunxi:twi_sunxi@2503400.twi5[INFO]: shutdown finish
Line 21164: [  202.013161][    T1] sunxi:twi_sunxi@2503400.twi5[INFO]: shutdown finish
```

图 2-6: 驱动加载成功

技巧

对应的 twi 总线加载成功后，会打印出 vx.x.x probe success, 其中 vx.x.x 代表目前驱动的版本号，其目的在于后期出现问题时，双方快速对其环境。

2.4 驱动框架介绍

图 2-7: TWI 模块结构框图

Linux 中 I2C 体系结构上图所示，图中用分割线分成了三个层次：

- 用户空间，包括所有使用 I2C 设备的应用程序；
- 内核，也就是驱动部分；
- 硬件，指实际物理设备，包括了 I2C 控制器和 I2C 外设。

其中，Linux 内核中的 I2C 驱动程序从逻辑上又可以分为 6 个部分：

- I2C framework 提供一种 “访问 I2C slave devices” 的方法。由于这些 slave devices 由 I2C controller 控制，因而主要由 I2C controller 驱动实现这一目标。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

- 经过 I2C framework 的抽象，用户可以不用关心 I2C 总线的技术细节，只需要调用系统的接口，就可以与外部设备进行通信。正常情况下，外部设备是位于内核态的其它 driver（如触摸屏，摄像头等等）。I2C framework 也通过字符设备向用户空间提供类似的接口，用户空间程序可以通过该接口访问从设备信息。
- 在 I2C framework 内部，有 I2C core、I2C busses、I2C algos 和 I2C muxes 四个模块。
- I2C core 使用 I2C adapter 和 I2C algorithm 两个子模块抽象 I2C controller 的功能，使用 I2C client 和 I2C driver 抽象 I2C slave device 的功能（对应设备模型中的 device 和 device driver）。另外，基于 I2C 协议，通过 smbus 模块实现 SMBus（System Management Bus，系统管理总线）的功能。
- I2C busses 是各个 I2C controller drivers 的集合，位于 drivers/i2c/busses/目录下，twi-sunxi-test.c、twi-sunxi.c、twi-sunxi.h。
- I2C algos 包含了一些通用的 I2C algorithm，所谓的 algorithm，是指 I2C 协议的通信方法，用于实现 I2C 的 read/write 指令，一般情况下，都是由硬件实现，不需要特别关注该目录。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

3 软件功能介绍

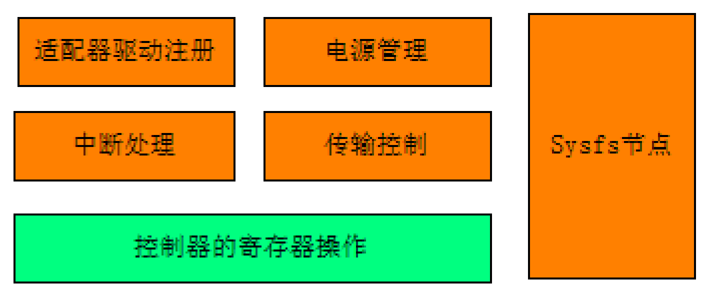


图 3-1: 模块功能结构

上面各个部分的作用如下所示：

- 适配器驱动注册，向 I2C Core 注册 I2C 总线驱动，驱动注册后会跟 Device Tree 中描述的 TWI 控制器进行匹配，匹配成功后会调用 probe 函数，初始化 TWI 控制器；
- 电源管理，负责 TWI 控制器的休眠、唤醒功能；
- 传输控制，I2C Core 将通过消息方式触发调用此子模块，在 "传输控制" 中发送 Start 就会启动一次 I2C 的读写操作；
- 中断处理，和 "传输控制" 一起构成 I2C 控制器驱动的软件处理核心部分，I2C 外设如果有数据过来就会触发中断，进入此中断处理函数，根据当前 I2C 控制器的不同状态做相应的处理，完成 I2C 的读写时序；
- 控制器的寄存器操作，主要是对 I2C 控制寄存器的操作进行了封装；
- sysfs 节点，提供一些在线的调试方法。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

3.1 函数初始化流程

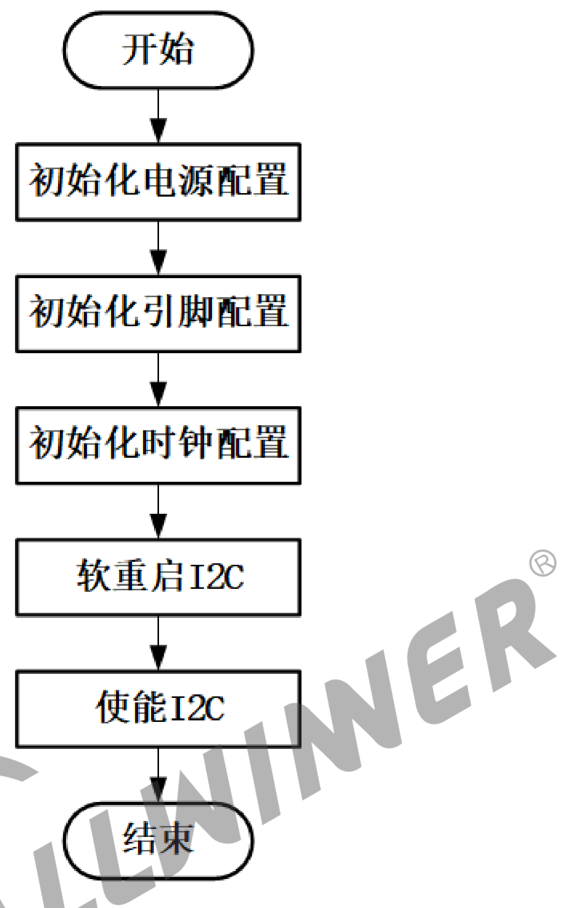


图 3-2: 模块初始化流程

I2C 设备初始化流程如下所示:

- 获取 I2C 设备的时钟，获取电源配置，获取引脚配置
- 配置电源适配器，配置引脚，配置时钟
- 如果需要使用 DMA 传输，软重启设备

版权所有 © 珠海全志科技股份有限公司。保留一切权利

3.2 数据发送流程

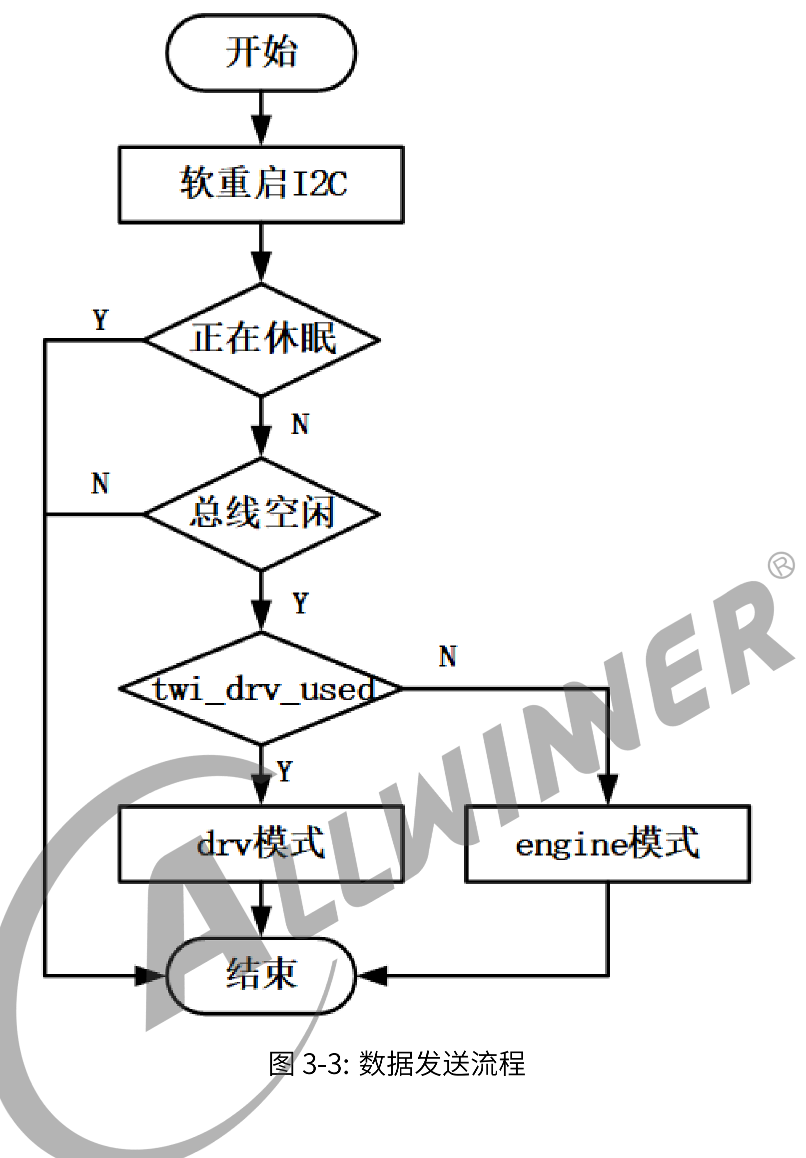


图 3-3: 数据发送流程

当上层应用使用 I2C 在传输数据的时候，会调用会 twi_algorithm 结构体的 master_xfer 函数，具体如下：

```
static const struct i2c_algorithm sunxi_twi_algorithm = {
    .master_xfer      = sunxi_twi_xfer,
    .functionality    = sunxi_twi_functionality,
#if IS_ENABLED(CONFIG_I2C_SLAVE)
    .reg_slave        = sunxi_twi_reg_slave,
    .unreg_slave      = sunxi_twi_unreg_slave,
#endif
};
```

其中 sunxi_twi_xfer 的定义如下所示：

```
static int
sunxi_twi_xfer(struct i2c_adapter *adap, struct i2c_msg *msgs, int num)
{
    struct sunxi_twi *twi = (struct sunxi_twi *)adap->algo_data;
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

```
    ...
    /* then the sunxi_twi_runtime_reseme() call back */
    ret = pm_runtime_get_sync(twi->dev);
    sunxi_twi_soft_reset(twi);                    //软重启I2C
    ret = sunxi_twi_bus_barrier(&twi->adap);
    /* set the twi status to idle */
    twi->result = RESULT_IDLE;
    if (twi->twi_drv_used)
        ret = sunxi_twi_drv_xfer(twi, msgs, num);      //drv模式
    else
        ret = sunxi_twi_engine_xfer(twi, msgs, num);   //engine模式

    return ret;
}
```

可见该函数是判断使用 engine 模式传输数据还是 drv 模式传输数据，其决定于设备树中的 twi_drv_used 配置。下面将会对 engine 模式的传输流程进行分析：

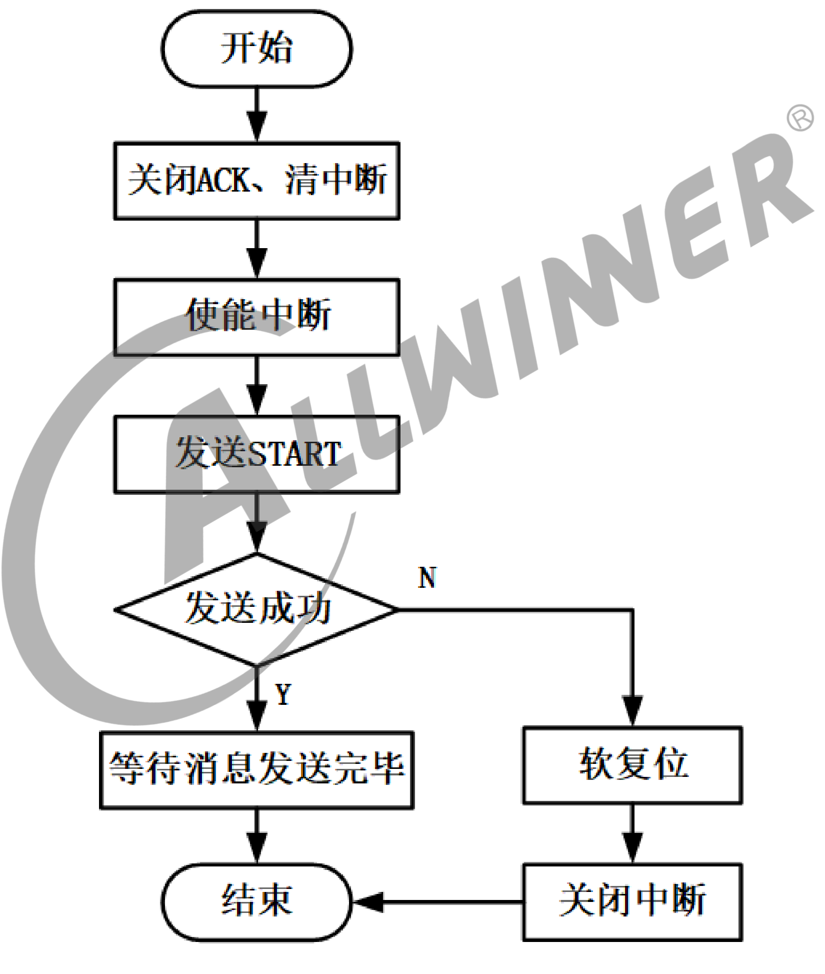


图 3-4: cpu 传输

在该传输中，先重启 TWI 设备，在配置需要传输的数据，使能 TWI 中断，然后启动 TWI 进行数据的传输，等待数据传输完毕。

3.3 delay_init 处理流程

delay_init 是为了满足在异构系统中先由异构小核使用 TWI 资源，在使用结束后通知 Linux 进行该路 twi 初始化，完成平滑切换的需求。因此在 Linux 中，该路 TWI 在执行 probe 时，跳过硬件资源的获取部分（也就是先不执行 sunxi_twi_hw_init() 函数），等到远端使用 twi 结束后，通过 rpmsg 信号通知 Linux，再执行资源获取的函数。

3.4 休眠唤醒处理流程

- 在 twi 中使用 runtime_suspend/runtime_resume 函数进行运行时的休眠唤醒，即当 5 秒内该路 twi 没有被调用，则会自动执行 runtime_suspend 函数，关闭总线、时钟和 pin，当该路 twi 再次被使用时，会执行 runtime_resume 函数，使能总线、时钟和 pin，恢复正常使用;
- 在执行系统休眠唤醒时，会调用 suspend_late/resume_early 函数 (不使用 noirq 函数的原因是，在休眠时 twi 需要将使用的电关掉，这个操作需要通过 s_twi0 操作 pmu 中的寄存器完成，而在 noirq 阶段，系统已经关闭中断，twi 无法使用，以上关电操作就会失败，因此改用 suspend_late/resume_early 函数);
- 当执行系统休眠时：
- 对于 pmu 使用的这一路 twi(一般为 s_twi0) **不做任何处理，直接退出 (为了保证在 cpus 以及唤醒时能够正常使用)**;
- 对于其他几路 twi，先关闭 I2C 子系统的调用 (**此时使用方调用 i2c_transfer 函数无法进行传输**)，判断此时是否存在正在进行的数据传输，若存在，则保证该笔传输完成；
- 最后调用 pm_runtime_force_suspend(真正会调用 runtime_suspend) 关闭电、总线、时钟和 pin，将此路 twi 状态设置为 suspended；若此时非 pmu 使用的 twi 并没有被使用且已经执行了 runtime_suspend 函数，则只执行断电操作 (关闭总线、时钟和 pin 已在 runtime_suspend 函数函数中完成，此路 twi 状态为 suspended)
- 在执行系统唤醒时：
- 对于 pmu 使用的这一路 twi(一般为 s_twi0) **不做任何处理，直接退出**;
- 而对于其他几路 twi，此时只完成了上电操作，使用 pm_runtime_force_resume 函数不会真正调用到 runtime_resume 函数，当该路 twi 被真正使用时，会调用到 runtime_resume 函数，使能总线、时钟和 pin;

注意: 当在休眠唤醒过程中出现问题，请先确认以下内容：

- 请确认自身模块的休眠唤醒函数是否在休眠是晚于或者唤醒时早于 TWI，即保证在使用时 TWI 进行传输时 TWI 资源已经准备好，能够正常使用;
- 请确保 TWI 的父时钟在休眠期间未被修改，若时钟被修改，则 twi 会在唤醒后因为分到一个异常的时钟频率而无法工作;

版权所有 © 珠海全志科技股份有限公司。保留一切权利

3.5 关机处理流程

- **在关机流程中，对于 pmu 使用的 twi，不做任何处理，直接退出**，保证在 cpus 中仍能继续正常使用完成关机；
- 对于正在使用中的 twi，保证传输完成最后一笔数据，并将状态修改为 SUNXI_TWI_XFER_STATUS_SHUTDOWN;
- 当 twi 处在 SUNXI_TWI_XFER_STATUS_SHUTDOWN 状态时，使用方使用 i2c_transfer 函数调用 twi 会显示失败 (此时 twi 拒接被使用);

3.6 DVFS 处理流程

TWI 支持多个内部模块 DVFS 调压请求通道，每个内部模块（如 CPU）有独立的通信接口连接 TWI 模块，在内部模块需要调压操作时，可直接向 TWI 模块发起调压请求，TWI 模块根据配置自动发起 I2C 读写操作控制外部 PMU 并修改对应的值，从而达到自动调压的目的，**DVFS 功能必须工作 twi drv 模式下。**TWI DVFS 支持以下功能：

- 一级仲裁：用于处理多 DVFS 请求同时发起的情况，可配置轮询模式或优先级模式
- 二级仲裁：用于 DVFS 请求与 CPU 请求同时发起的情况，采用轮询方式
- 回读功能：支持发送成功后，是否立刻回读进行数据比对
- Mask 功能：支持发送数据前，先回读数据并根据 mask 值将对应数据位 mask 掉后再将新的数据发出去

版权所有 © 珠海全志科技股份有限公司。保留一切权利

4 模块接口说明

4.1 内部接口

总线驱动除了标准的 probe、remove、resume、suspend 接口外，和 TWI 控制器密切相关的内部接口有以下几个。

4.1.1 sunxi_twi_xfer()

- 函数原型：static int sunxi_twi_xfer(struct i2c_adapter adap, struct i2c_msg msgs, int num)
- 功能描述：由 I2C Core 发来的消息触发调用此接口，通过发送 Start 然后启动一次 I2C 的读写操作。
- 参数说明：adap，指向当前的 I2C 适配器;msgs，指向待处理的 I2C 消息;num，待处理的 I2C 消息个数
- 返回值：已经成功处理的 I2C 消息个数

4.1.2 sunxi_twi_handler()

- 函数原型：static irqreturn_t sunxi_twi_handler(int this_irq, void * dev_id)
- 功能描述：处理 TWI 控制器产生的中断信号
- 参数说明：irq，中断号;dev_id，自定义的回调参数，在此接口中该参数类型是 struct sunxi_twi *
- 返回值：IRQ_HANDLED，中断已经出来；IRQ_NONE，不是此设备的中断。

4.2 DVFS 接口

DVFS 接口函数的定义在文件 inlude/linux/twi/sunxi-twi.h 中，主要有以下几个：

4.2.1 sunxi_twi_dvfs_enable_all()

- 函数原型：int sunxi_twi_dvfs_enable_all(struct i2c_adapter *adap)
- 功能描述：使能所有的 dvfs 通道

- 参数说明：adap，指向所属的 I2C 总线控制器
- 返回值：成功返回 0，失败返回负数

4.2.2 sunxi_twi_dvfs_disable_all()

- 函数原型：int sunxi_twi_dvfs_disable_all(struct i2c_adapter *adap)
- 功能描述：失能所有的 dvfs 通道
- 参数说明：adap，指向所属的 I2C 总线控制器
- 返回值：成功返回 0，失败返回负数

4.2.3 sunxi_twi_dvfs_enable()

- 函数原型：sunxi_twi_dvfs_enable(struct i2c_adapter *adap, int ch)
- 功能描述：使能 dvfs 指定通道
- 参数说明：adap，指向所属的 I2C 总线控制器；ch, 目标通道编号
- 返回值：成功返回 0，失败返回负数

4.2.4 sunxi_twi_dvfs_disable()

- 函数原型：sunxi_twi_dvfs_disable(struct i2c_adapter *adap, int ch)
- 功能描述：失能 dvfs 指定通道
- 参数说明：adap，指向所属的 I2C 总线控制器；ch, 目标通道编号
- 返回值：成功返回 0，失败返回负数

4.2.5 sunxi_twi_dvfs_set_slave_addr()

- 函数原型：sunxi_twi_dvfs_set_slave_addr(struct i2c_adapter *adap, u8 addr)
- 功能描述：设置 dvfs 设备配置
- 参数说明：adap，指向所属的 I2C 总线控制器；addr, dvfs 设备地址
- 返回值：成功返回 0，失败返回负数

4.2.6 sunxi_twi_dvfs_set_interval()

- 函数原型：sunxi_twi_dvfs_set_interval(struct i2c_adapter *adap, u16 interval)
- 功能描述：设置 packet 传输间隔时间
- 参数说明：adap，指向所属的 I2C 总线控制器；interval, 包间隔时间
- 返回值：成功返回 0，失败返回负数

版权所有 © 珠海全志科技股份有限公司。保留一切权利

4.2.7 sunxi_twi_dvfs_chan_init()

- 函数原型：sunxi_twi_dvfs_chan_init(struct i2c_adapter *adap, int ch, u8 priority, u8 reg, u8 mask, bool rb)
- 功能描述：dvfs 通道初始化
- 参数说明：adap，指向所属的 I2C 总线控制器；ch, 目标通道；priority，通道优先级；reg，寄存器地址；mask，通道遮罩；rb，通道回读功能
- 返回值：成功返回 0，失败返回负数

4.2.8 sunxi_twi_dvfs_set_chan_reg()

- 函数原型：sunxi_twi_dvfs_set_chan_reg(struct i2c_adapter *adap, int ch, u8 reg)
- 功能描述：设置 dvfs 通道寄存器地址
- 参数说明：adap，指向所属的 I2C 总线控制器；ch, 目标通道；reg，寄存器地址
- 返回值：成功返回 0，失败返回负数

4.2.9 sunxi_twi_dvfs_set_chan_mask()

- 函数原型：sunxi_twi_dvfs_set_chan_mask(struct i2c_adapter *adap, int ch, u8 mask)
- 功能描述：设置 dvfs 通道遮罩
- 参数说明：adap，指向所属的 I2C 总线控制器；ch, 目标通道；mask，通道遮罩
- 返回值：成功返回 0，失败返回负数

4.2.10 sunxi_twi_dvfs_set_chan_rb()

- 函数原型：sunxi_twi_dvfs_set_chan_rb(struct i2c_adapter *adap, int ch, bool rb)
- 功能描述：设置 dvfs 通道回读
- 参数说明：adap，指向所属的 I2C 总线控制器；ch, 目标通道；rb，开启通道回读
- 返回值：成功返回 0，失败返回负数

4.2.11 sunxi_twi_dvfs_set_chan_prio()

- 函数原型：sunxi_twi_dvfs_set_chan_prio(struct i2c_adapter *adap, int ch, u8 priority)
- 功能描述：设置 dvfs 通道优先级
- 参数说明：adap，指向所属的 I2C 总线控制器；ch, 目标通道；priority，通道优先级，配置范围 0~2，若大于 2 则等效于 2
- 返回值：成功返回 0，失败返回负数

版权所有 © 珠海全志科技股份有限公司。保留一切权利

4.3 i2c-core 接口

4.3.1 i2c_transfer

- 函数原型：int i2c_transfer(struct i2c_adapter *adap, struct i2c_msg *msgs, int num)。
- 作用：完成 TWI 总线和 TWI 设备之间的一定数目的 TWI message 交互。
- 参数：

（1）adap：指向所属的 TWI 总线控制器；

（2）msgs：i2c_msg 类型的指针；

（3）num：表示一次需要处理几个 TWI msg。

- 返回值：

（1）>0：已经处理的 msg 个数；

（2）<0：失败。

4.3.2 i2c_master_recv

- 函数原型：int i2c_master_recv(const struct i2c_client *client, char *buf, int count)。
- 作用：通过封装 i2c_transfer() 完成一次 I2c 接收操作。
- 参数：

（1）client：指向当前 TWI 设备的实例；

（2）buf：用于保存接收到的数据缓存；

（3）count：数据缓存 buf 的长度。

- 返回值：

（1）>0：成功接收的字节数；

（2）<0：失败。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

4.3.3 i2c_master_send

- 函数原型：int i2c_master_send(const struct i2c_client *client, const char *buf, int count)。
- 作用：通过封装 i2c_transfer() 完成一次 I2c 发送操作。
- 参数：

（1）client：指向当前 TWI 从设备的实例；

（2）buf：要发送的数据；

（3）count：要发送的数据长度。

- 返回值：

（1）>0：成功发送的字节数；

（2）<0：失败。

4.4 i2c 用户态调用接口

i2c 的操作在内核中是当做字符设备来操作的，可以通过利用文件读写接口（open，write，read，ioctrl）等操作内核目录中的/dev/i2c-* 文件来条用相关的接口，i2c 相关的操作定义在 i2c-dev.c 里面，本节将介绍比较重要的几个接口：

4.4.1 i2cdev_open

- 函数原型：static int i2cdev_open(struct inode *inode, struct file *file)。
- 作用：程序（C 语言等）使用 open(file) 时调用的函数。打开一个 twi 设备，可以像文件读写的方式往 twi 设备中读写数据。
- 参数：

（1）inode：inode 节点；

（2）file：file 结构体。

- 返回值：文件描述符。

4.4.2 i2cdev_read

- 函数原型：static ssize_t i2cdev_read(struct file *file, char __user *buf, size_t count,loff_t *offset)。
- 作用：程序（C 语言等）调用 read() 时调用的函数。像往文件里面读数据一样从 twi 设备中读数据。底层调用 twi_xfer 传输数据。
- 参数：

（1）file：file 结构体；

（2）buf，写数据 buf；

（3）offset, 文件偏移。

- 返回值：

（1）非空：返回读取的字节数；

（2）<0：失败。

4.4.3 i2cdev_write

- 函数原型：static ssize_t i2cdev_write(struct file *file, const char __user *buf,size_t count, loff_t *offset)。
- 作用：程序（C 语言等）调用 write() 时调用的函数。像往文件里面写数据一样往 twi 设备中写数据。底层调用 twi_xfer 传输数据。
- 参数：

（1）file：file 结构体；

（2）buf：读数据 buf；

（3）offset, 文件偏移。

- 返回值：

（1）0：成功；

（2）<0：失败。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

4.4.4 i2cdev_ioctl

- 函数原型：static long i2cdev_ioctl(struct file *file, unsigned int cmd, unsigned long arg)。
- 作用：程序（C 语言等）调用 ioctl() 时调用的函数。像对文件管理 i/o 一样对 twi 设备管理。该功能比较强大，可以修改 twi 设备的地址，往 i2 设备里面读写数据，使用 smbus 等等，详细的可以查阅该函数。
- 参数：

（1）file：file 结构体；

（2）cmd：指令；

（3）arg：其他参数。

- 返回值：

（1）0：成功；

（2）<0：失败。

5 用户态 TWI 功能开发

5.1 功能概述

用户态 TWI 功能是在 Linux 用户空间与 I2C 设备通信，实现 I2C 设备的访问、数据读写等功能。

5.2 开发流程

- 步骤 1：调用 open 打开 I2C 设备文件，获取文件描述符。

```
fd = open("/dev/i2c-1", O_RDWR);
```

- 步骤 2：调用 ioctl 设置 I2C 从设备地址。

```
ioctl(fd, TWI_SLAVE_FORCE, 0x50);
```

- 步骤 3：调用 read 和 write 函数读写 I2C 设备。

```
write(fd, wrbuf, 3);
read(fd, &rddata, 1);
```

说明

示例代码，简要说明读写函数使用方法，不提供实际应用功能

```
int main() {
 ......
   unsigned char rddata;
   unsigned char rdaddr[2] = {0, 0}; /* 将要读取的数据在芯片中的偏移量 */
   unsigned char wrbuf[3] = {0, 0, 0x3c}; /* 要写的数据，头两字节为偏移量 */

   printf("input a char you want to write to E2PROM\n");
   wrbuf[2] = getchar();
   printf("write return:%d, write data:%x\n", write(fd, wrbuf, 3), wrbuf[2]);
   sleep(1);
   printf("write address return: %d\n",write(fd, rdaddr, 2)); /* 读取之前首先设置读取的偏移量 */
   printf("read data return:%d\n", read(fd, &rddata, 1));
   printf("rddata: %c\n", rddata);
......
}
```

- 步骤 4：调用 close 关闭 I2C 设备。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

```
close(fd);
```

5.3 注意事项

- 确认使用的 I2C 总线编号（例如/dev/i2c-1），确保与实际硬件连接匹配。
- 确保正确设置 I2C 从设备地址，避免与其他设备地址冲突。

5.4 编程示例

```
#include <sys/ioctl.h>
#include <fcntl.h>
#include <linux/i2c-dev.h>
#include <linux/i2c.h>
#define CHIP "/dev/i2c-1"
#define CHIP_ADDR 0x50
int main()
{
  unsigned char rddata;
  unsigned char rdaddr[2] = {0, 0}; /* 将要读取的数据在芯片中的偏移量 */
  unsigned char wrbuf[3] = {0, 0, 0x3c}; /* 要写的数据，头两字节为偏移量 */
  int fd = open(CHIP, O_RDWR);
  if (fd < 0)
  {
    printf("open "CHIP"failed\n");
    goto exit;
  }
  if (ioctl(fd, TWI_SLAVE_FORCE, CHIP_ADDR) < 0)
  { /* 设置芯片地址 */
    printf("oictl:set slave address failed\n");
    goto close;
  }
  printf("input a char you want to write to E2PROM\n");
  wrbuf[2] = getchar();
  printf("write return:%d, write data:%x\n", write(fd, wrbuf, 3), wrbuf[2]);
  sleep(1);
  printf("write address return: %d\n",write(fd, rdaddr, 2)); /* 读取之前首先设置读取的偏移量 */
  printf("read data return:%d\n", read(fd, &rddata, 1));
  printf("rddata: %c\n", rddata);
  close(fd);
  exit:
  return 0;
}
```

6 内核态 TWI 功能开发

6.1 功能概述

内核态 TWI 功能是在 Linux 内核空间与 I2C 设备通信，实现 I2C 设备的访问、数据读写等功能。

6.2 开发流程

- 步骤 1：调用 i2c_add_driver 注册 I2C 驱动

```
i2c_add_driver(&eeprom_driver);
```

- 步骤 2：实现读写函数

读操作：

调用 i2c_master_send(client, &addr, 1); 发送从 eeprom 读取的起始地址

调用 i2c_master_recv(client, data, length); 读取指定长度的数据

写操作:

构建数据缓冲区，将地址和数据一起写入

调用 i2c_master_send(client, buffer, length + 1); 发送地址和数据

说明

示例代码，简要说明读写函数使用方法，不提供实际应用功能

```
static int eeprom_read(struct i2c_client *client, u8 addr, u8 *data, size_t length) {
    int ret;

    // 发送地址
    ret = i2c_master_send(client, &addr, 1);
    if (ret < 0) return ret;

    // 读取数据
    ret = i2c_master_recv(client, data, length);
    if (ret < 0) return ret;

    return 0;
}
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

```
static int eeprom_write(struct i2c_client *client, u8 addr, u8 *data, size_t length) {
    int ret;
    u8 *buffer;

    // 分配缓冲区
    buffer = kmalloc(length + 1, GFP_KERNEL);
    if (!buffer) return -ENOMEM;

    buffer[0] = addr;
    memcpy(&buffer[1], data, length);

    // 写入数据
    ret = i2c_master_send(client, buffer, length + 1);
    kfree(buffer);
    if (ret < 0) return ret;

    return 0;
}
```

- 步骤 3：调用 i2c_del_driver 注销 I2C 驱动

```
i2c_del_driver(&eeprom_driver);
```

6.3 注意事项

- 确保设备树中正确配置 I2C 的相关信息，如地址、大小等。

6.4 编程示例

在内核源码中有现成的 i2c 设备驱动实例：sdk/kernel/linux-{KERN_VER}/drivers/misc/eeprom/at24.c , 这是一个 EEPROM 的 TWI 设备驱动，为了验证 TWI 总线驱动，所以其中通过 sysfs 节点实现读写访问。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

7 调试方法

7.1 i2c-tools 调试工具

i2c-tools 是一个开源工具，专门用来调试 TWI 设备。可以用 i2c-tools 来获取 twi 设备的相关信息（默认集成在内核里面），并且读写相关的 twi 设备的数据。

i2c-tools 主要是通过读写/dev/i2c-* 文件获取 TWI 设备，所以需要在 menuconfig 里面把 TWI 的 device interface 节点打开，具体的 i2c-tools 使用方法如下：

7.1.1 i2cdetect

7.1.1.1 参数说明

```
# ./i2cdetect
Usage: i2cdetect [-y] [-a] [-q|-r] TWIBUS [FIRST LAST]
    i2cdetect -F TWIBUS
    i2cdetect -l
 TWIBUS is an integer or an TWI bus name
 If provided, FIRST and LAST limit the probing range.
```

- -y: 关闭交互式，不会显示警告信息
- -a: 扫描总线上所有设备
- -q: 使用 SMBus 的 “quick write” 命令进行检测，不建议使用
- -r: 使用 SMBus 的 “receive byte” 命令进行检测，不建议使用
- TWIBUS: 指定查询某个总线编号
- FIRST LAST：扫描的地址范围

7.1.1.2 使用示例

在小机端运行:

```
./i2cdetect -l
```

运行结果如下图所示，探测设备中所有的 i2c 总线：

```
root@TinaLinux:/# ./i2cdetect -l
i2c-0    i2c          SUNXI I2C(0x05002000)            I2C adapter
i2c-1    i2c          SUNXI I2C(0x05002400)            I2C adapter
```

图 7-1: i2cdetect 命令结果

7.1.2 i2cdump

7.1.2.1 参数说明

```
# ./i2cdump
Usage: i2cdump [-f] [-y] [-r first-last] [-a] TWIBUS ADDRESS [MODE [BANK [BANKREG]]]
  TWIBUS is an integer or an TWI bus name
  ADDRESS is an integer (0x08 - 0x77, or 0x00 - 0x7f if -a is given)
  MODE is one of:
    b (byte, default)
    w (word)
    W (word on even register addresses)
    s (SMBus block, deprecated)
    i (TWI block)
    c (consecutive byte)
    Append p for SMBus PEC
```

- -r: 指定寄存器范围，只能扫描从 first 到 last 区域;
- -f: 强制访问设备;
- -y: 关闭人机交互模式
- TWIBUS: 总线编号
- ADDRESS: 指定设备地址
- MODE: 指定读取的大小，b 字节，w 字，s 是 SMBus 块，i 是 twi 块

7.1.2.2 使用示例

在小机端运行:

```
# ./i2cdump -f -y 0 0x50
```

运行结果如下图所示，打印出 i2c-0 总线上 0x50 设备中所有寄存器的值:

版权所有 © 珠海全志科技股份有限公司。保留一切权利

```
root@TinaLinux:/# ./i2cdump -f -y 0 0x50
No size specified (using byte-data access)
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f    0123456789abcdef
00: 30 31 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f    01"#$%&'()*+,-./
10: 1a 0b 0c 0d 0e 0f 10 11 12 13 14 15 16 17 18 19    ????????????????
20: 2e 1f 20 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d    .? !"#$%&'()*+,-
30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
40: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
80: 00 40 2c e7 c2 00 00 00 00 28 00 00 00 00 00 00    .@,??....(......
90: 00 00 00 00 e7 c2 d4 25 e7 c2 94 29 e7 c2 00 00    ....???%???)??..
a0: 00 00 00 a1 00 00 00 e7 c2 00 00 00 00 00 00 00    ...?...??.......
b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
```

图 7-2: i2cdump 结果分析

ALLWINNER®

7.1.3 i2cset

7.1.3.1 参数说明

```
# ./i2cset
Usage: i2cset [-f] [-y] [-m MASK] [-r] [-a] TWIBUS CHIP-ADDRESS DATA-ADDRESS [VALUE] ... [MODE]
  TWIBUS is an integer or an TWI bus name
  ADDRESS is an integer (0x08 - 0x77, or 0x00 - 0x7f if -a is given)
  MODE is one of:
    c (byte, no value)
    b (byte data, default)
    w (word data)
    i (TWI block data)
    s (SMBus block data)
    Append p for SMBus PEC
```

- -r: 写入后立即回读寄存器的值，并将结果与写入的值进行比较;
- TWIBUS：总线编号;
- CHIP-ADDRESS:twi 设备地址;
- DATA-ADDRESS:twi 寄存器地址;
- VALUE: 要写入的值
- MODE：指定读取的大小，b 字节，w 字，s 是 SMBus 块，i 是 twi 块

版权所有 © 珠海全志科技股份有限公司。保留一切权利

7.1.3.2 使用示例

在小机端运行:

```
# ./i2cset -f -y 0 0x50 0x00 0xff
```

运行结果如下图所示，向 i2c-0 总线上 0x50 设备中地址为 0x00 寄存器中写入 0xff 数据:

```
root@TinaLinux:/# ./i2cset -f -y 0 0x50 0x00 0xff
root@TinaLinux:/#
```

图 7-3: i2cset 运行结果

7.1.4 i2cget

7.1.4.1 参数说明

```
# ./i2cget
Usage: i2cget [-f] [-y] [-a] TWIBUS CHIP-ADDRESS [DATA-ADDRESS [MODE [LENGTH]]]
  TWIBUS is an integer or an TWI bus name
  ADDRESS is an integer (0x08 - 0x77, or 0x00 - 0x7f if -a is given)
  MODE is one of:
    b (read byte data, default)
    w (read word data)
    c (write byte/read byte)
    s (read SMBus block data)
    i (read TWI block data)
    Append p for SMBus PEC
  LENGTH is the TWI block data length (between 1 and 32, default 32)
```

- -f: 强制访问
- -y: 关闭交互模式，不会提示警告信息
- TWIBUS: 总线编号
- CHIP-ADDRESS:twi 设备地址
- DATA-ADDRESS:twi 寄存器地址
- MODE: 指定读取的大小，b 字节，w 字，s 是 SMBus 块，i 是 twi 块

7.1.4.2 使用示例

在小机端运行:

```
# ./i2cget -f -y 0 0x50 0x00
```

运行结果如下图所示，读取出 i2c-0 总线上 0x50 设备中地址为 0x00 寄存器的值:

```
root@TinaLinux:/# ./i2cget -f -y 0 0x50 0x00
0xff
```

图 7-4: i2cget 运行结果

7.1.5 i2ctransfer

注意：i2ctransfer 接口是在 v4.0 版本之后才支持的，因要使用该命令需要获取 4.0 以上的 i2c-tools 工具；源码包可在下方网站中进行获取 (需注意，要获取 4.0 以及以上版本)：

```
https://mirrors.edge.kernel.org/pub/software/utils/i2c-tools/
```

7.1.5.1 参数说明

```
# ./i2ctransfer
Usage: i2ctransfer [-f] [-y] [-v] [-V] [-a] TWIBUS DESC [DATA] [DESC [DATA]]...
 TWIBUS is an integer or an TWI bus name
 DESC describes the transfer in the form: {r|w}LENGTH[@address]
  1) read/write-flag 2) LENGTH (range 0-65535, or '?')
  3) TWI address (use last one if omitted)
 DATA are LENGTH bytes for a write message. They can be shortened by a suffix:
  = (keep value constant until LENGTH)
  + (increase value by 1 until LENGTH)
  - (decrease value by 1 until LENGTH)
  p (use pseudo random generator until LENGTH with value as seed)

Example (bus 0, read 8 byte at offset 0x64 from EEPROM at 0x50):
 # i2ctransfer 0 w1@0x50 0x64 r8
Example (same EEPROM, at offset 0x42 write 0xff 0xfe ... 0xf0):
 # i2ctransfer 0 w17@0x50 0x42 0xff-
```

7.1.5.2 使用示例

在小机端运行

```
# ./i2ctransfer -f -y 0 w42@0x50 0x00 0xaa=
```

运行结果如下图所示，向 i2c-0 总线的 0x50 设备的 0x00 地址写入 41 个字节的数据，数据内容为 0xaa(因为是 drv-mode，且传输数据数量大于 32 个字节，因此采用 DMA 传输)：

```
root@TinaLinux:/# ./i2ctransfer -f -y 0 w42@0x50 0x00 0x00+
[  472.057010] sunxi-i2c 5002000.twi: drv-mode: dma write data end
```

图 7-5: i2ctransfer 运行结果

在小机端运行

```
./i2ctransfer -f -y 6 w1@0x36 0x00 r50
```

运行结果如下图所示，向 i2c-6 总线的 0x36 设备的 0x00 地址读 50 个字节的数据 (因为是 drv-mode，且传输数据数量大于 32 个字节，因此采用 DMA 传输)：

```
/ # i2ctransfer -f -y 6 w1@0x36 0x00 r50
0x10 0x00 0x00 0x49 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x07 0x00 0x01 0x41 0xbc 0x28 0x0d 0x1c 0x00 0x00 0x22 0x06 0x00 0x1e 0x80 0x
x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
/ #
```

图 7-6: i2ctransfer 读结果

8 FAQ

8.1 TWI 数据未完全发送

问题现象：incomplete xfer。具体的 log 如下所示：

```
[ 1658.926643] sunxi_twi_do_xfer()1936 - [twi0] incomplete xfer (status: 0x20, dev addr: 0x50)
[ 1658.926643] sunxi_twi_do_xfer()1936 - [twi0] incomplete xfer (status: 0x48, dev addr: 0x50)
```

问题分析：此错误表示主控已经发送了数据（status 值为 0x20 时，表示发送了 SLAVE ADDR + WRITE；status 值为 0x48 时，表示发送了 SLAVE ADDR + READ），但是设备没有回 ACK，这表明设备无响应，应该检查是否未接设备、接触不良、设备损坏和上电时序不正确导致的设备未就绪等问题。

问题排查步骤：

步骤 1：通过设备树里面的配置信息，核对引脚配置是否正确。每组 TWI 都有好几组引脚配置。

步骤 2：更换 TWI 总线下的设备为 at24c16，用 i2ctools 读写 at24c16 看看是否成功，成功则表明总线工作正常；

步骤 3：排查设备是否可以正常工作以及设备与 TWI 之间的硬件接口是否完好；

步骤 4：详细了解当前需要操作的设备的初始化方法，工作时序，使用方法，排查因初始化设备不正确导致通讯失败；

步骤 5：用示波器检查 TWI 引脚输出波形，查看波形是否匹配。

8.2 TWI 起始信号无法发送

问题现象：START can’t sendout!。具体的 log 如下所示：

```
sunxi_twi_do_xfer()1865 - [twi1] START can't sendout!
```

问题分析：此错误表示 TWI 无法发送起始信号，一般跟 TWI 总线的引脚配置以及时钟配置有关。应该检查引脚配置是否正确，时钟配置是否正确，引脚是否存在上拉电阻等等。

问题排查步骤：

步骤 1：重新启动内核，通过查看 log，分析 TWI 是否成功初始化，如若存在引脚配置问题，应核对引脚信息是否正确；

版权所有 © 珠海全志科技股份有限公司。保留一切权利

步骤 2：根据原理图，查看 TWI-SCK 和 TWI-SDA 是否经过合适的上拉电阻接到 3.3v 电压；

步骤 3：用万用表量 SDA 与 SCL 初始电压，看电压是否在 3.3V 附近（断开此 TWI 控制器所有外设硬件连接与软件通讯进程）；

步骤 4：核查引脚配置以及 clk 配置是否进行正确设置；

步骤 5：测试 PIN 的功能是否正常，利用寄存器读写的方式，将 PIN 功能直接设为 INPUT 功能（echo [reg] [val] > /sys/class/sunxi_dump/write），然后将 PIN 上拉和接地改变 PIN 状态，读 PIN 的状态 (echo [reg,reg] > /sys/class/sunxi_dump/dump;cat dump)，看是否匹配。

步骤 6：测试 CLK 的功能是否正常，利用寄存器读写的方式，将 TWI 的 CLK gating 等打开，（echo [reg] [val] > /sys/class/sunxi_dump/write），然后读取相应 TWI 的寄存器信息，读 TWI 寄存器的数据（echo [reg],[len]> /sys/class/sunxi_dump/dump)，查看寄存器数据是否正常。

8.3 TWI 终止信号无法发送

问题现象：STOP can’t sendout。具体的 log 如下所示：

```
twi_stop()511 - [twi4] STOP can't sendout!
sunxi_twi_core_process()1726 - [twi4] STOP failed!
```

问题分析：此错误表示 TWI 无法发送终止信号，一般跟 TWI 总线的引脚配置。应该检查引脚配置是否正确，引脚电压是否稳定等等。

问题排查步骤：

步骤 1：根据原理图，查看 TWI-SCK 和 TWI-SDA 是否经过合适的上拉电阻接到 3.3v 电压；

步骤 2：用万用表量 SDA 与 SCL 初始电压，看电压是否在 3.3V 附近（断开此 TWI 控制器所有外设硬件连接与软件通讯进程）；

步骤 3：测试 PIN 的功能是否正常，利用寄存器读写的方式，将 PIN 功能直接设为 INPUT 功能（echo [reg] [val] > /sys/class/sunxi_dump/write），然后将 PIN 上拉和接地改变 PIN 状态，读 PIN 的状态 (echo [reg,reg] > /sys/class/sunxi_dump/dump;cat dump)，看是否匹配；

步骤 4: 查看设备树配置，把其他用到 SCK/SDA 引脚的节点关闭，重新测试 TWI 通信功能。

8.4 TWI 传送超时

问题现象：xfer timeout。具体的 log 如下所示：

```
[123.681219] sunxi_twi_do_xfer()1914 - [twi3] xfer timeout (dev addr:0x50)
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

问题分析: 此错误表示主控已经发送完起始信号，但是在与设备通信的过程中无法正常完成数据发送与接收，导致最终没有发出终止信号来结束 TWI 传输，导致的传输超时问题。应该检查引脚配置是否正常，CLK 配置是否正常，TWI 寄存器数据是否正常，是否有其他设备干扰，中断是否正常等问题。

问题排查步骤:

步骤 1：核实 TWI 控制器配置是否正确；

步骤 2：根据原理图，查看 TWI-SCK 和 TWI-SDA 是否经过合适的上拉电阻接到 3.3v 电压；

步骤 3：用万用表量 SDA 与 SCL 初始电压，看电压是否在 3.3V 附近（断开此 TWI 控制器所有外设硬件连接与软件通讯进程）；

步骤 4：关闭其他 TWI 设备，重新进行烧录测试 TWI 功能是否正常；

步骤 5：测试 PIN 的功能是否正常，利用寄存器读写的方式，将 PIN 功能直接设为 INPUT 功能（echo [reg] [val] > /sys/class/sunxi_dump/write），然后将 PIN 上拉和接地改变 PIN 状态，读 PIN 的状态 (echo [reg,reg] > /sys/class/sunxi_dump/dump;cat dump)，看是否匹配；

步骤 6：测试 CLK 的功能是否正常，利用寄存器读写的方式，将 TWI 的 CLK gating 等打开，（echo [reg] [val] > /sys/class/sunxi_dump/write），然后读取相应 TWI 的寄存器信息，读 TWI 寄存器的数据（echo [reg] ,[len]> /sys/class/sunxi_dump/dump)，查看寄存器数据是否正常；

步骤 7：根据相关的 LOG 跟踪 TWI 代码执行流程，分析报错原因。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

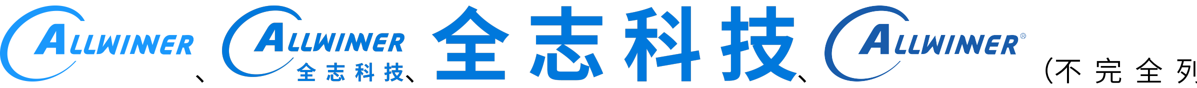
著作权声明

版权所有 ©2025 珠海全志科技股份有限公司。保留一切权利。

本文档及内容受著作权法保护，其著作权由珠海全志科技股份有限公司（“全志”）拥有并保留一切权利。

本文档是全志的原创作品和版权财产，未经全志书面许可，任何单位和个人不得擅自摘抄、复制、修改、发表或传播本文档内容的部分或全部，且不得以任何形式传播。

商标声明



（不完全列举）均为珠海全志科技股份有限公司的商标或者注册商标。在本文档描述的产品中出现的其它商标，产品名称，和服务名称，均由其各自所有人拥有。

免责声明

您购买的产品、服务或特性应受您与珠海全志科技股份有限公司（“全志”）之间签署的商业合同和条款的约束。本文档中描述的全部或部分产品、服务或特性可能不在您所购买或使用的范围内。使用前请认真阅读合同条款和相关说明，并严格遵循本文档的使用说明。您将自行承担任何不当使用行为（包括但不限于如超压，超频，超温使用）造成的不利后果，全志概不负责。

本文档作为使用指导仅供参考。由于产品版本升级或其他原因，本文档内容有可能修改，如有变更，恕不另行通知。全志尽全力在本文档中提供准确的信息，但并不确保内容完全没有错误，因使用本文档而发生损害（包括但不限于间接的、偶然的、特殊的损失）或发生侵犯第三方权利事件，全志概不负责。本文档中的所有陈述、信息和建议并不构成任何明示或暗示的保证或承诺。

本文档未以明示或暗示或其他方式授予全志的任何专利或知识产权。在您实施方案或使用产品的过程中，可能需要获得第三方的权利许可。请您自行向第三方权利人获取相关的许可。全志不承担也不代为支付任何关于获取第三方许可的许可费或版税（专利税）。全志不对您所使用的第三方许可技术做出任何保证、赔偿或承担其他义务。