Linux UART
开发指南

版本号: 1.8
发布日期: 2025.4.7

版本历史

版本号	日期	制/修订人	内容描述
1.0	2022.08.04	XAA0249	初始版本
1.1	2022.11.03	AWA1979	更新文档模板
1.2	2024.1.30	XAA0249	补充波特率配置说明
1.3	2024.3.14	XAA0312	更新文档内容
1.4	2024.12.24	XAA0249	补充两种驱动版本的使用说明
1.5	2025.2.17	XAA0249	补充软件 485 配置在不同版本驱动上的使用说明
1.6	2025.3.10	XAA0249	补充如何支持特性波特率
1.7	2025.4.8	XAA0249	修复格式问题
1.8	2025.4.7	XAA0249	补充控制台修改波特率问题

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 相关术语介绍 . . . 1
1.4.1 硬件术语 . . . 1
1.4.2 软件驱动版本介绍 . . . 2

2 模块介绍 **3**
2.1 模块功能介绍 . . . 3
2.2 模块流程介绍 . . . 3
2.3 驱动框架介绍 . . . 4
2.4 源码结构介绍 . . . 5
2.4.1 AW_UART . . . 5
2.4.2 AW_UART_NG . . . 5
2.5 模块配置介绍 . . . 6
2.5.1 AW_UART . . . 6
2.5.1.1 menuconfig 配置说明 . . . 6
2.5.1.2 设备树配置说明 . . . 9
2.5.2 AW_UART_NG . . . 11
2.5.2.1 menuconfig 配置说明 . . . 11
2.5.2.2 设备树配置说明 . . . 14
2.6 模块特殊功能配置方法 . . . 16
2.6.1 AW_UART . . . 16
2.6.1.1 dma 模式配置 . . . 16
2.6.1.2 485 模式配置 . . . 16
2.6.2 AW_UART_NG . . . 17
2.6.2.1 dma 模式配置 . . . 17
2.6.2.2 485 模式 . . . 17

3 模块接口说明 **19**
3.1 打开/关闭串口 . . . 19
3.2 读/写串口 . . . 19
3.3 设置串口属性 . . . 19
3.3.1 tcgetattr . . . 23
3.3.2 tcsetattr . . . 23
3.3.3 cfgetispeed . . . 24
3.3.4 cfgetospeed . . . 24
3.3.5 cfsetispeed . . . 25

版权所有 © 珠海全志科技股份有限公司。保留一切权利

3.3.6 cfsetospeed . . . 25
3.3.7 cfsetspeed . . . 25
3.3.8 tcflush . . . 26

4 模块使用范例 27

5 调试方法 32
5.1 调试工具 . . . 32
5.2 调试节点 . . . 32
5.2.1 /sys/devices/platform/soc*/*uart*/dev_info . . . 32
5.2.2 /sys/devices/platform/soc*/*uart*/status . . . 33
5.2.3 /sys/devices/platform/soc*/*uart*/ctrl_info . . . 33

6 FAQ 34
6.1 如何切换 uart 口为打印 conole . . . 34
6.2 如何确认当前平台支持哪些波特率 . . . 34
6.3 控制台 1.5M 波特率相关问题 . . . 34
6.3.1 原因 . . . 35
6.3.2 硬件要求 . . . 35
6.3.3 如何切换回 115200 波特率 . . . 36
6.4 关于 uart 波特率 . . . 36
6.4.1 波特率计算公式与大致原理 . . . 36
6.4.2 如何计算当前配置下的误码率 . . . 38
6.4.3 如何设置当前平台支持某个特定波特率 . . . 38
6.4.3.1 判断默认配置是否满足波特率要求 . . . 39
6.4.3.2 修改 dtsi 中的时钟配置 . . . 39
6.4.4 如何设置不同的波特率 . . . 40
6.5 如何测试 UART 回环功能 . . . 41
6.6 接收数据时为何分段 . . . 41

版权所有 © 珠海全志科技股份有限公司。保留一切权利

插　图

图 2-1　初始化流程 4
图 2-2　内核 menuconfig 根菜单 6
图 2-3　Allwinner BSP 菜单 7
图 2-4　Device Drivers 菜单 8
图 2-5　UART Drivers 配置菜单 9
图 2-6　menuconfig 根目录 12
图 2-7　Allwinner bsp 下目录 12
图 2-8　Device Driver 目录 13
图 2-9　UART Driver 目录 13
图 6-1　数据错位示例 35
图 6-2　image-20250310175031687 40
图 6-3　接收数据分段 41

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Linux内核中UART驱动的接口及使用方法，为 UART 设备的使用者提供参考。

1.2 目标读者

UART 驱动、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-5.4	sunxi-uart.c 或 sunxi-uart-ng-core.c
Linux-5.10	sunxi-uart.c 或 sunxi-uart-ng-core.c
Linux-5.15	sunxi-uart.c 或 sunxi-uart-ng-core.c
Linux-6.1	sunxi-uart.c 或 sunxi-uart-ng-core.c
Linux-6.6	sunxi-uart.c 或 sunxi-uart-ng-core.c

1.4 相关术语介绍

1.4.1 硬件术语

表 1-2: 硬件术语

术语	解释说明
sunxi	指 Allwinner 的一系列 SOC 硬件平台
UART	Universal Asynchronous Receiver/Transmitter，通用异步收发传输器
Console	控制台，Linux 内核中用于输出调试信息的 TTY 设备

版权所有 © 珠海全志科技股份有限公司。保留一切权利

术语	解释说明
TTY	TeleType/TeleTypewriters 的一个老缩写，原来指的是电传打字机，现在泛指和计算机串行端口连接的终端设备。TTY 设备还包括虚拟控制台，串口以及伪终端设备

1.4.2 软件驱动版本介绍

在 SDK 中根据 IC 设计不同有两种驱动版本，可以将其称为AW_UART与AW_UART_NG，在代码发布时已经做好配置，不需要用户自行修改所用的软件驱动版本。

用户想使用特殊功能时：

- 需要先确认自己使用的是哪一种驱动版本，详细区分方法见章节 “menuconfig 配置说明” 或 “设备树配置说明”，两种方法均可区分驱动版本
- 再按照 “模块特殊功能配置方法” 中对应章节的描述进行配置即可

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 模块功能介绍

通用异步收发器（Universal Asynchronous Receiver/Transmitter），通常称作UART，是一种串行、异步、全双工的通信协议，在嵌入式领域应用非常广泛。在 19 世纪 60 年代，为了解决计算机和电传打字机通信，Bell 发明了UART协议，将并行输入信号转换成串行输出信号。因为 UART 简单实用的特性，其已经成为一种使用非常广泛的通讯协议。全志的 uart 控制器支持以下功能：

- 兼容标准16550串口
- 支持发送接收FIFO（不同平台 FIFO 大小有差异，具体情况见 user manual/datasheet）
- 支持DMA控制接口
- 支持软件/硬件流控配置
- 支持2/4/8线串口接口
- 支持5-8数据位和1/1.5/2停止位
- 支持奇偶校验，无校验位

2.2 模块流程介绍

下图为 UART 模块初始化流程介绍：

版权所有 © 珠海全志科技股份有限公司。保留一切权利

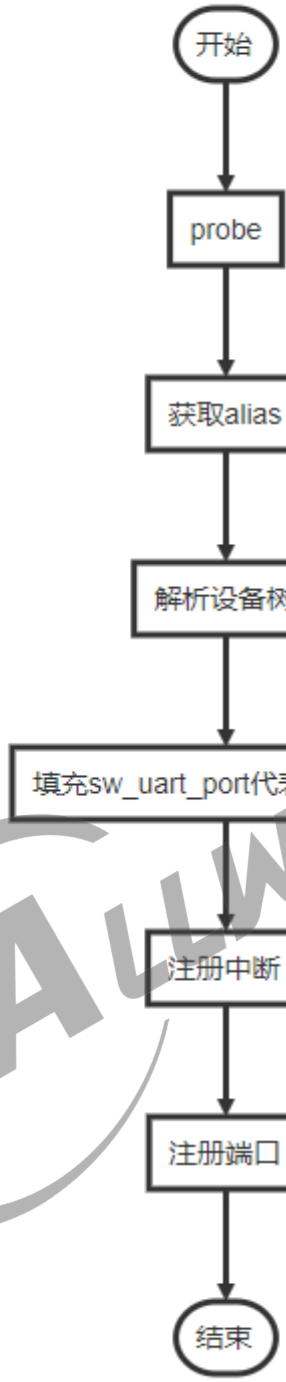

图 2-1: 初始化流程

2.3 驱动框架介绍

Linux 内核中，UART 驱动的结构可以分为三个层次:

版权所有 © 珠海全志科技股份有限公司。保留一切权利

```
TTY core
  |
UART Core
  |
Sunxi UART Driver
```

1. Sunxi UART Driver，负责 SUNXI 平台 UART 控制器的初始化、数据通信等，也是我们要实现的部分。
2. UART Core，为 UART 驱动提供了一套 API, 完成设备和驱动的注册等。
3. TTY core，实现了内核中所有 TTY 设备的注册和管理。

2.4 源码结构介绍

2.4.1 AW_UART

```
|-- bsp
|   |-- drivers
|   |   |-- uart
|   |       |-- early_printk.c
|   |       |-- Kconfig
|   |       |-- Makefile
|   |       |-- sunxi-uart.c
|   |       |-- sunxi-uart.h
```

- early_printk.c：earlyprintk功能，一般打开。
- sunxi-uart.c：sunxi平台驱动核心文件。
- sunxi-uart.h：sunxi头文件，用来描述硬件信息。

2.4.2 AW_UART_NG

```
|-- bsp
|   |-- drivers
|   |   |-- uart
|   |       |-- early_printk.c
|   |       |-- Kconfig
|   |       |-- Makefile
|   |       |-- sunxi-uart-ng-core.c
|   |       |-- sunxi-uart-ng-dma.c
|   |       |-- sunxi-uart-ng.h
```

- early_printk.c：earlyprintk功能，一般打开。
- sunxi-uart-ng-core.c：sunxi平台驱动核心文件。
- sunxi-uart-ng-dma.c：sunxi平台驱动 dma 部分核心文件。
- sunxi-uart-ng.h：sunxi头文件，用来描述硬件信息。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

2.5 模块配置介绍

2.5.1 AW_UART

2.5.1.1 menuconfig 配置说明

在 SDK 顶层目录，执行./build.sh menuconfig(需要先执行./build.sh config) 进入配置主界面，并按以下步骤操作：

- 首先，选择 Allwinner BSP 选项进入下一级配置，如下图所示。

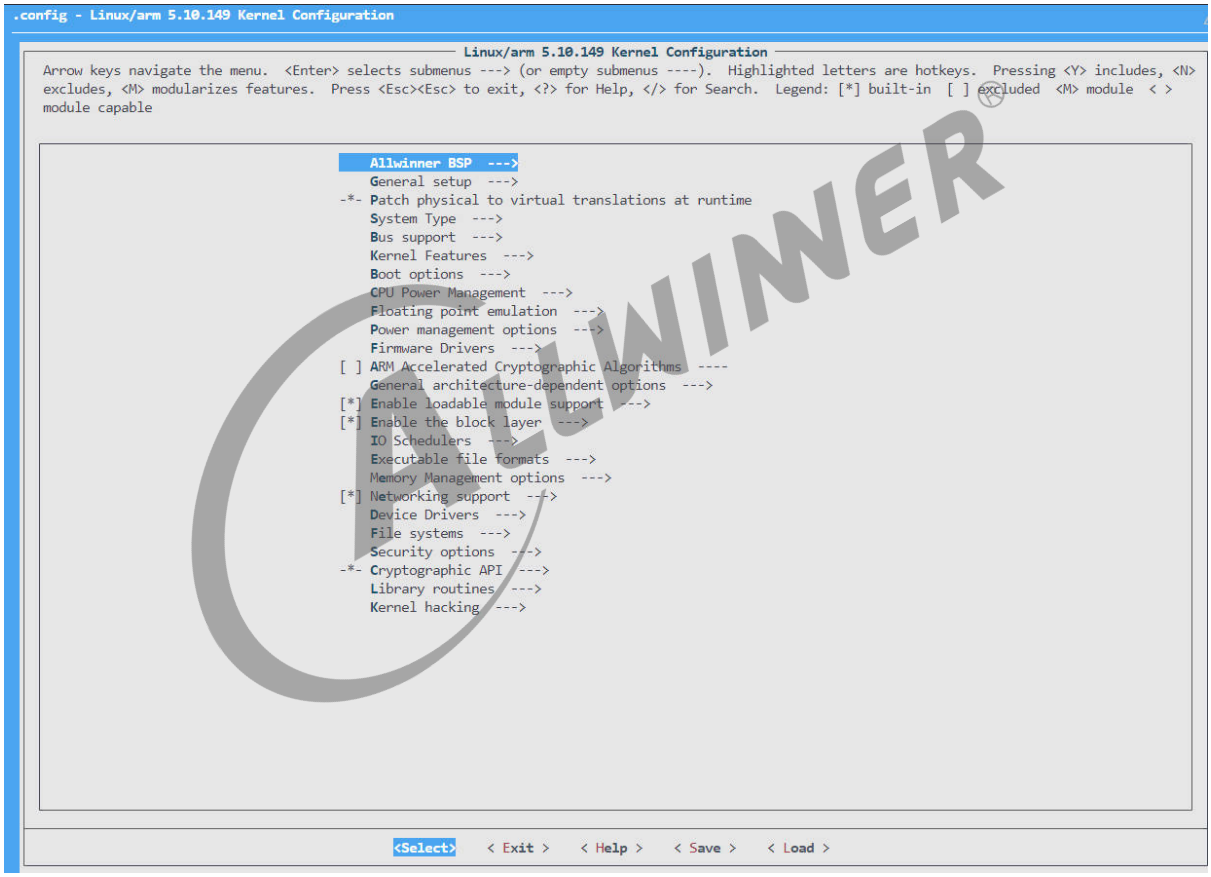

图 2-2: 内核 menuconfig 根菜单

- 选择 Device Drivers 进入下级配置，如下图所示：

版权所有 © 珠海全志科技股份有限公司。保留一切权利

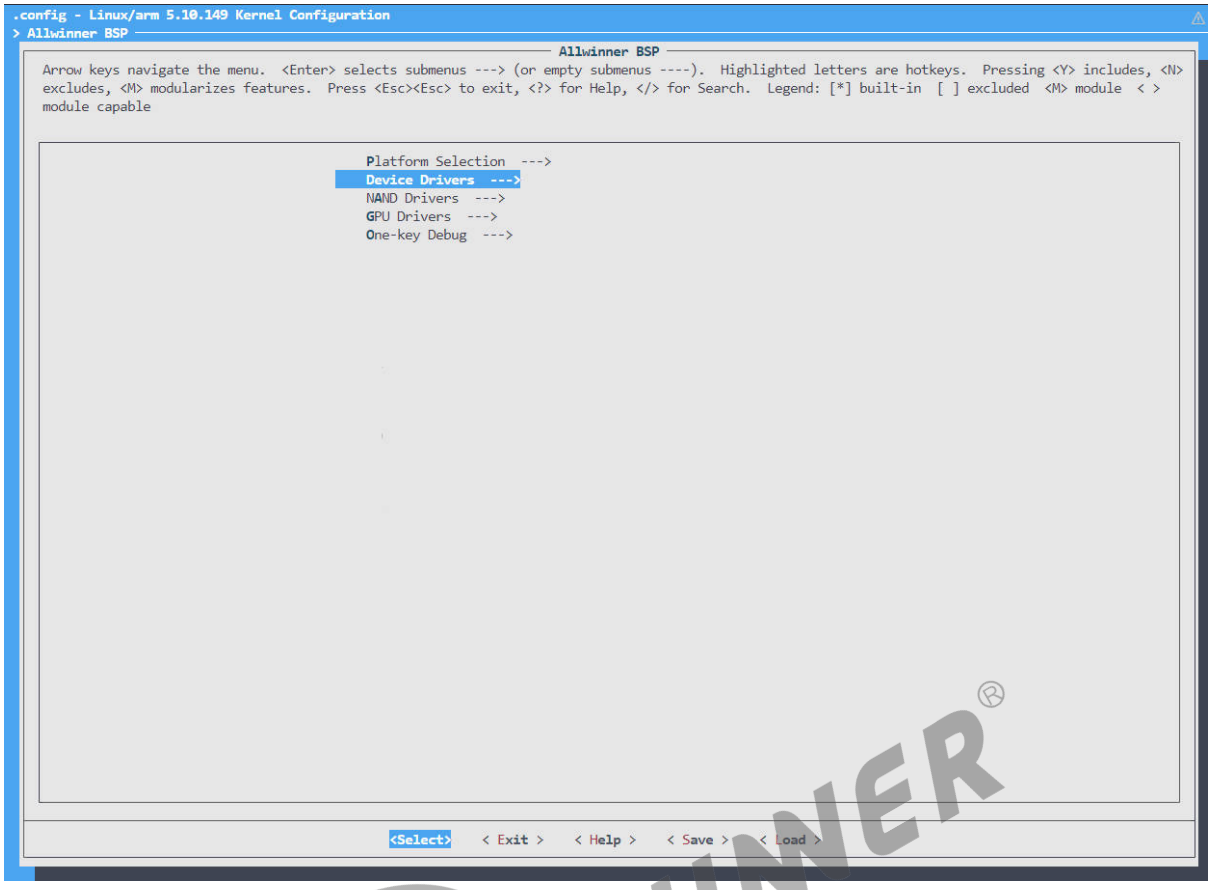

图 2-3: Allwinner BSP 菜单

- 选择 UART Drivers 进入下级配置，如下图所示：

版权所有 © 珠海全志科技股份有限公司。保留一切权利

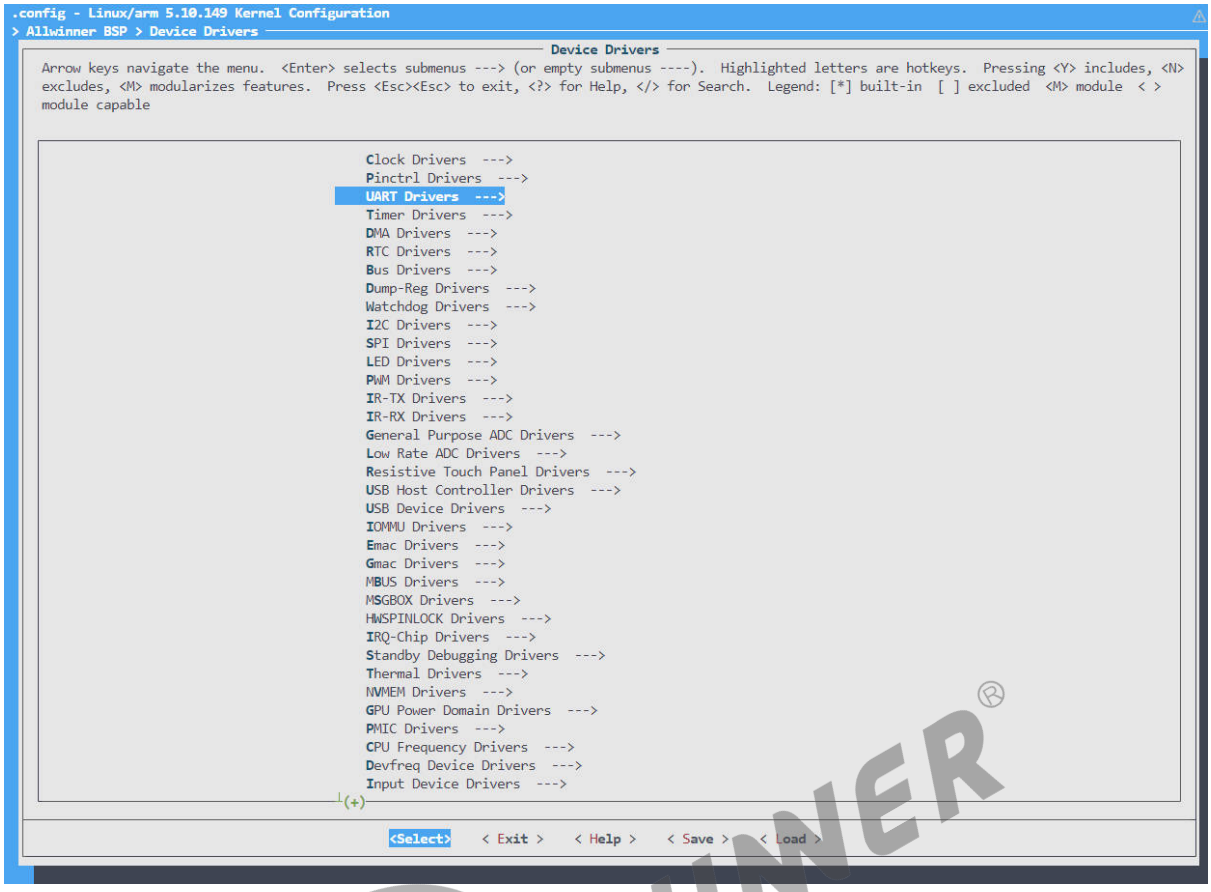

图 2-4: Device Drivers 菜单

- 选择 UART Support for Allwinner SoCs 和 Enable Console on UART 选项，如下图：
- 注意如果想使用 AW_UART 的 dma 功能，需要打开 Enable DMA for UART

版权所有 © 珠海全志科技股份有限公司。保留一切权利

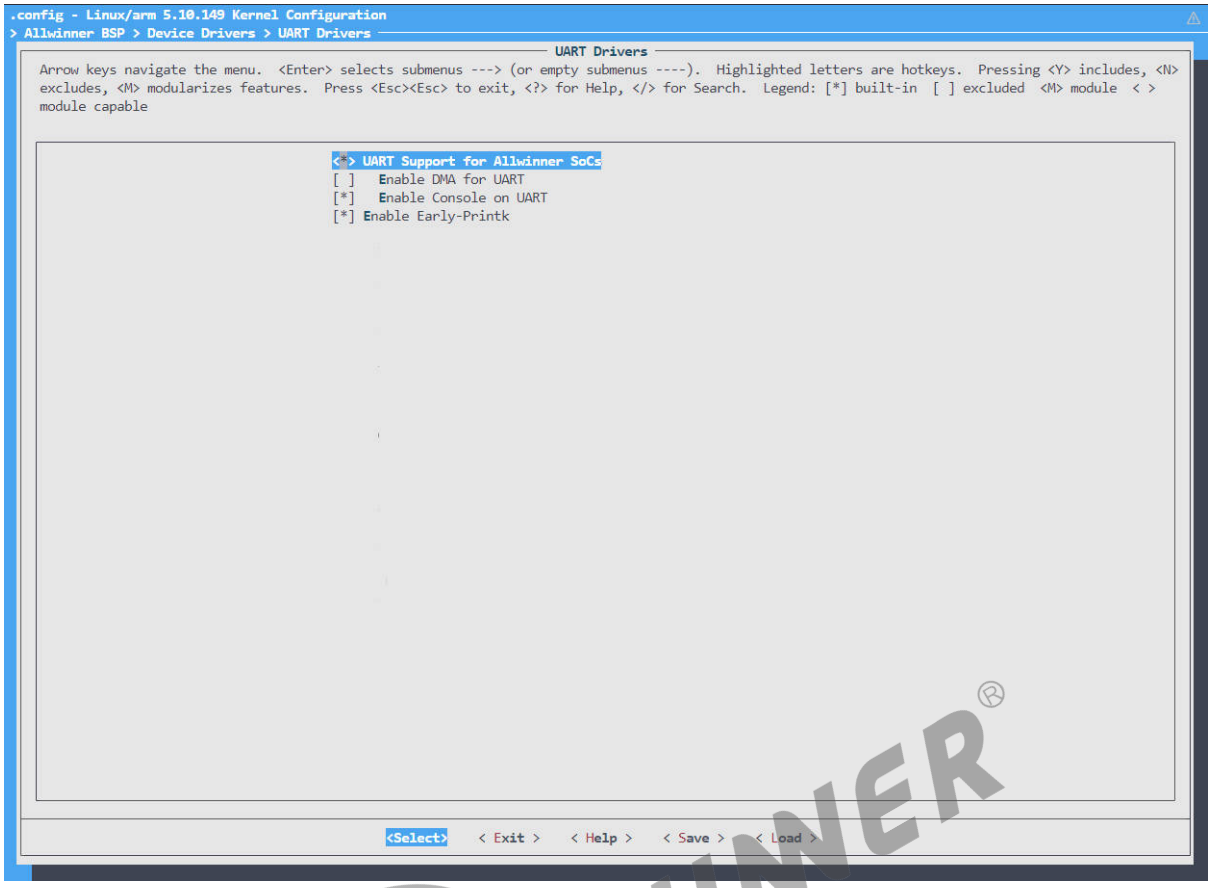

图 2-5: UART Drivers 配置菜单

2.5.1.2 设备树配置说明

- 设备树文件的配置是该SoC所有方案的通用配置，soc 设备树的路径为内核目录下：bsp/configs/linux/sun*.dtsi。
- 板级设备树路径：device/config/chips/{IC}/configs/{BOARD}/linux-{kernel-ver}/board.dts。

2.5.1.2.1 soc 设备树配置

```
uart0: uart@1c28000 {
    compatible = "allwinner,sun8i-uart";
    device_type = "uart0";
    reg = <0x0 0x01c28000 0x0 0x400>;
    interrupts = <GIC_SPI 1 IRQ_TYPE_LEVEL_HIGH>;
    clocks = <&ccu CLK_BUS_UART0>;    /* 设备使用的时钟 */
    resets = <&ccu RST_BUS_UART0>;    /* 设备reset时钟 */
    uart0_port = <0>;                 /* 端口号 */
    uart0_type = <2>;                 /* 该串口支持的模式，通常有2/4/8 */
    sunxi,uart-fifosize = <64>;       /* 模块FIFO深度 */
    status = "disabled";
};
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

1. compatible表征具体的设备, 用于驱动和设备的绑定；（**注：客户无需关注**）

注意当驱动版本为 AW_UART 时，可选的 compatible 目前只有以下几种

{ .compatible = "allwinner,sun8i-uart", }, { .compatible = "allwinner,sun50i-uart", }, { .compatible = "allwinner,sun55i-uart", }, { .compatible = "allwinner,sun20i-uart", }, { .compatible = "allwinner,sun251i-uart", },

2. reg设备使用的地址；（**注：客户无需关注**）
3. interrupts设备使用的中断；（**注：客户无需关注**）
4. clocks/reset设备使用的时钟；（**注：客户无需关注**）
5. uart0_port端口号；（**注：客户无需关注**）
6. uart0_type表示当前为几线模式；（**注：客户无需关注**）
7. sunxi,uart-fifosize表示fifo深度；（**注：客户无需关注**）
8. status是否使能该设备节点；（**注：客户无需关注**）

为了在UART 驱动代码中区分每一个UART 控制器，需要在Device Tree 中的 aliases 节点中为每一个UART 节点指定别名。别名形式为字符串 “serial” 加连续编号的数字，在 UART 驱动程序中可以通过of_alias_get_id() 函数获取对应的 UART 控制器的数字编号，从而区分每一个 UART 控制器。如下所示：

```
aliases {
    ......
    serial0 = &uart0;
    serial1 = &uart1;
    serial2 = &uart2;
    serial3 = &uart3;
    serial4 = &uart4;
    serial5 = &uart5;
    serial6 = &uart6;
    serial7 = &uart7;
    ......
};
```

2.5.1.2.2 板级设备树配置

board.dts 用于保存每个板级平台的设备信息 (如 demo 板、demo2.0 板等等)，board.dts 路径如下：

/device/config/chips/{IC}/configs/{BOARD}/board.dts。

在 board.dts 中的配置信息如果在 *.dtsi(如 sun50iw9p1.dtsi 等) 存在，则遵循以下规则：

1. 相同属性和结点，board.dts 的配置信息会覆盖 *.dtsi 中的配置信息。
2. 新增加的属性和结点，会添加到编译生成的 dtb 文件中。

board.dts配置示例如下：

版权所有 © 珠海全志科技股份有限公司。保留一切权利

```
&pio {
    uart0_pins_a: uart0_pins@0 {
        pins = "PB22", "PB23";
        function = "uart0";
    };

    uart0_pins_b: uart0_pins@1 {
        pins = "PB22", "PB23";
        function = "gpio_in";
    };
};

&uart0 {
    pinctrl-names = "default", "sleep";
    pinctrl-0 = <&uart0_pins_a>;
    pinctrl-1 = <&uart0_pins_b>;
    status = "okay";
};
```

1. “pinctrl-0”设备使用状态下的 GPIO 配置，“pinctrl-1”设备休眠状态下的 GPIO 配置；（**注：客户需关注，需与真正使用的 GPIO 引脚一致**）
2. status = "okay"当前uart的状态；（**注：客户需关注，当使用 uart 时需要打开**）
3. pins当前 uart 使用的硬件pin脚；**（注：若开启硬件流控时，需要添加rts及cts引脚，具体的引脚号需查看芯片手册）**

2.5.2 AW_UART_NG

2.5.2.1 menuconfig 配置说明

在 SDK 顶层目录，执行./build.sh menuconfig(需要先执行./build.sh config) 进入配置主界面，并按以下步骤操作：

- 首先，选择 Allwinner BSP 选项进入下一级配置，如下图所示。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

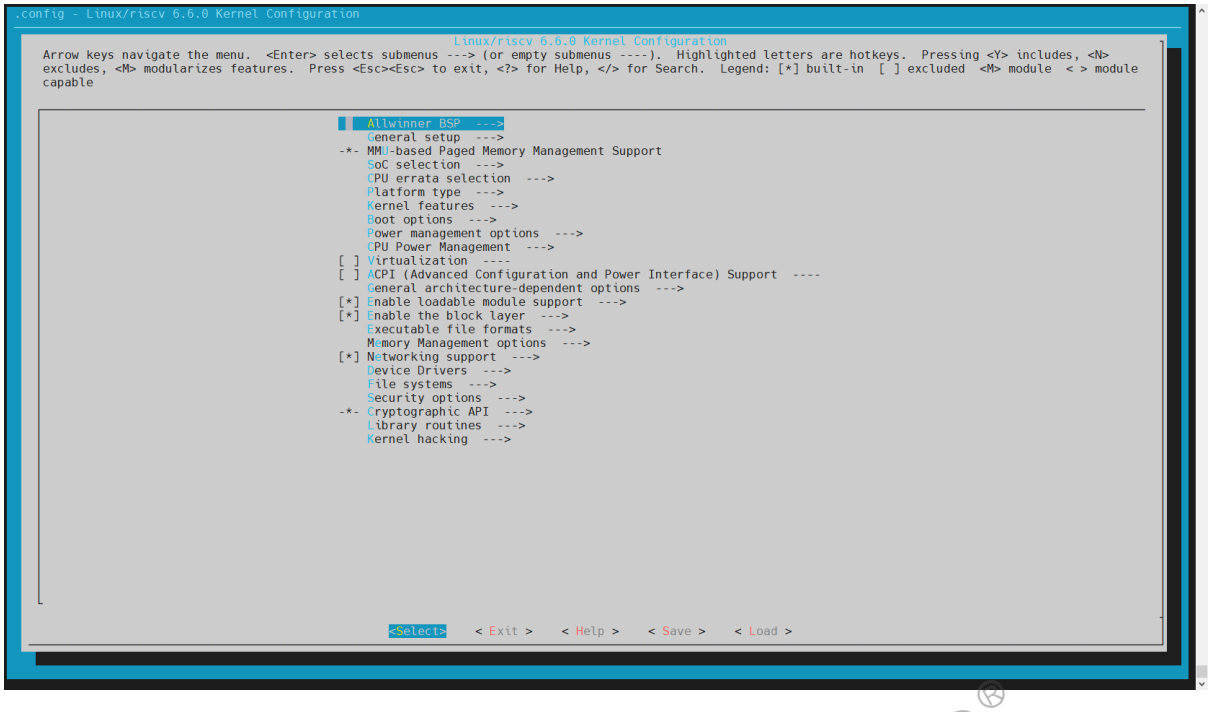

图 2-6: menuconfig 根目录

- 选择 Device Drivers 进入下级配置，如下图所示：

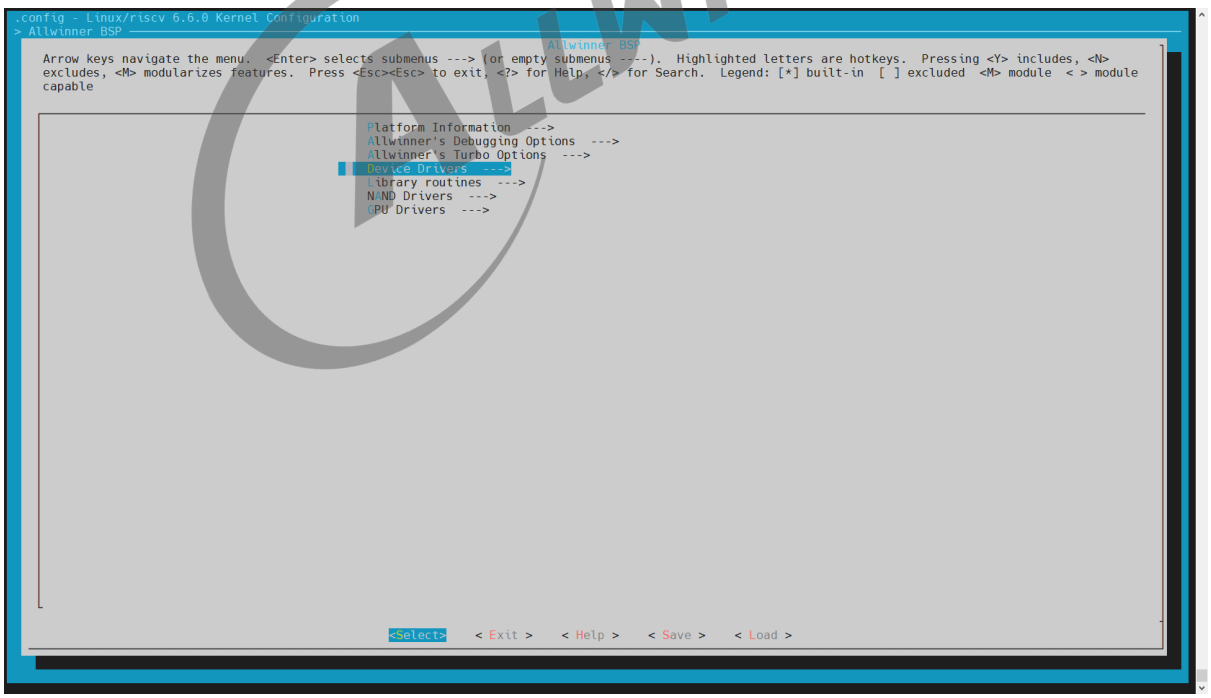

图 2-7: Allwinner bsp 下目录

- 选择 UART Drivers 进入下级配置，如下图所示：

版权所有 © 珠海全志科技股份有限公司。保留一切权利

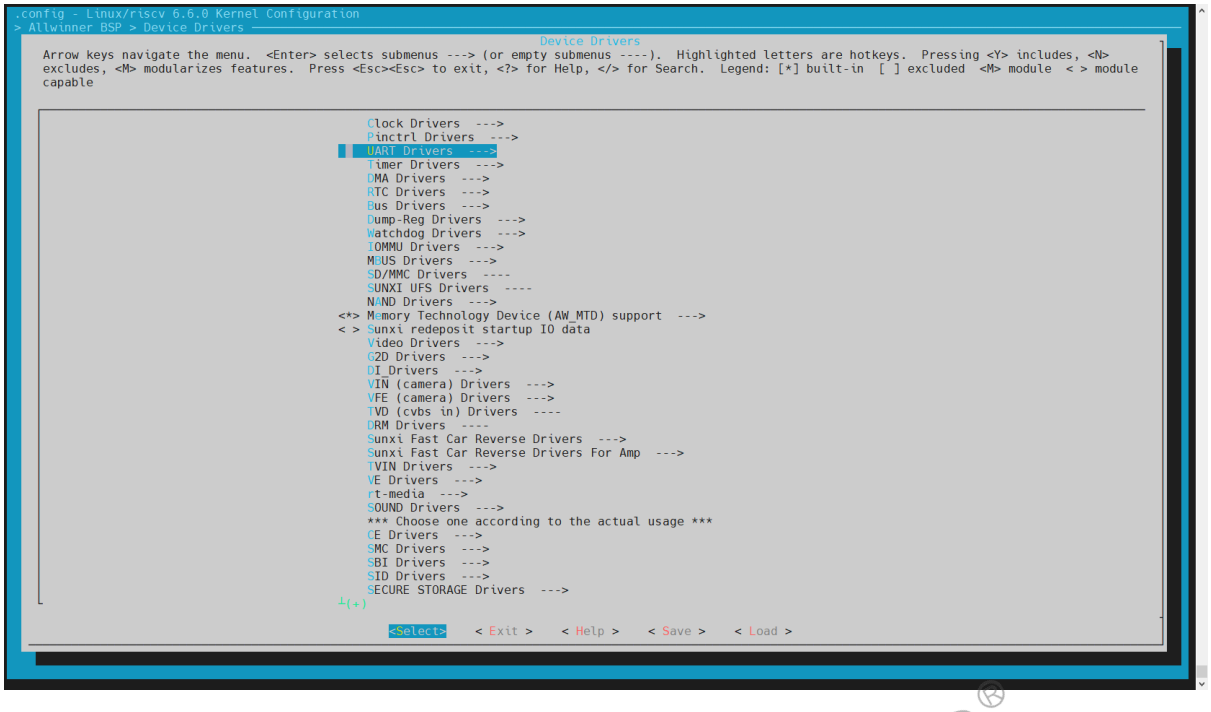

图 2-8: Device Driver 目录

- 选择 UART NG Support for Allwinner SoCs 和 Enable Console on UART 选项，如下图：

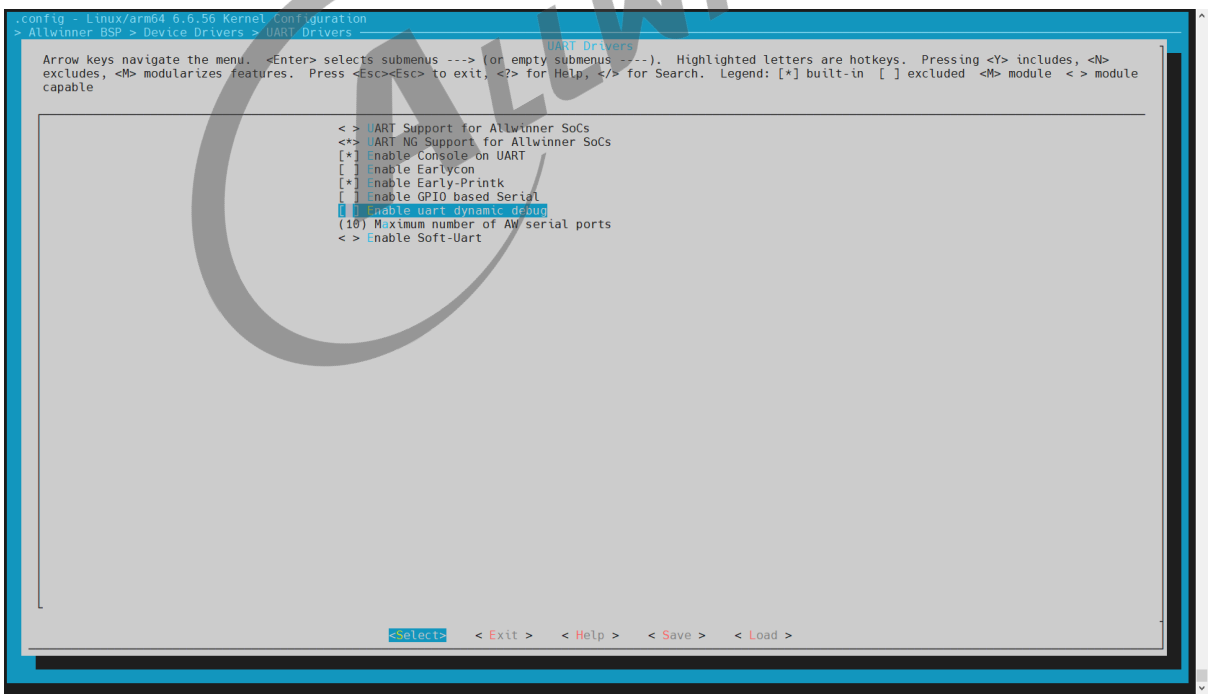

图 2-9: UART Driver 目录

版权所有 © 珠海全志科技股份有限公司。保留一切权利

2.5.2.2 设备树配置说明

- 设备树文件的配置是该SoC所有方案的通用配置，soc 设备树的路径为内核目录下：bsp/configs/linux/sun*.dtsi。
- 板级设备树路径：device/config/chips/{IC}/configs/{BOARD}/linux-{kernel-ver}/board.dts。

2.5.2.2.1 soc 设备树配置

```
uart0: uart@1c28000 {
    compatible = "allwinner,uart-v100";
    device_type = "uart0";
    reg = <0x0 0x01c28000 0x0 0x400>;
    interrupts = <GIC_SPI 1 IRQ_TYPE_LEVEL_HIGH>;
    clocks = <&ccu CLK_BUS_UART0>;     /* 设备使用的时钟 */
    resets = <&ccu RST_BUS_UART0>;     /* 设备reset时钟 */
    uart0_port = <0>;                  /* 端口号 */
    uart0_type = <2>;                  /* 该串口支持的模式，通常有2/4/8 */
    sunxi,uart-fifosize = <64>;        /* 模块FIFO深度 */
    status = "disabled";
};
```

1. compatible表征具体的设备, 用于驱动和设备的绑定；（**注：客户无需关注**）

注意当驱动版本为 AW_UART 时，可选的 compatible 目前只有以下几种

{ .compatible = "allwinner,uart-v100", .data = &sunxi_uart_v100_data}, { .compatible = "allwinner,uart-v101", .data = &sunxi_uart_v101_data},

2. reg设备使用的地址；（**注：客户无需关注**）
3. interrupts设备使用的中断；（**注：客户无需关注**）
4. clocks/reset设备使用的时钟；（**注：客户无需关注**）
5. uart0_port端口号；（**注：客户无需关注**）
6. uart0_type表示当前为几线模式；（**注：客户无需关注**）
7. sunxi,uart-fifosize表示fifo深度（**注：客户无需关注**）
8. status是否使能该设备节点；（**注：客户无需关注**）

为了在UART 驱动代码中区分每一个UART 控制器，需要在Device Tree 中的 aliases 节点中为每一个UART 节点指定别名。别名形式为字符串 “serial” 加连续编号的数字，在 UART 驱动程序中可以通过of_alias_get_id() 函数获取对应的 UART 控制器的数字编号，从而区分每一个 UART 控制器。如下所示：

版权所有 © 珠海全志科技股份有限公司。保留一切权利

```
aliases {
    ......
    serial0 = &uart0;
    serial1 = &uart1;
    serial2 = &uart2;
    serial3 = &uart3;
    serial4 = &uart4;
    serial5 = &uart5;
    serial6 = &uart6;
    serial7 = &uart7;
    ......
};
```

2.5.2.2.2 板级设备树配置

board.dts 用于保存每个板级平台的设备信息 (如 demo 板、demo2.0 板等等)，board.dts 路径如下：

/device/config/chips/{IC}/configs/{BOARD}/board.dts。

在 board.dts 中的配置信息如果在 *.dtsi(如 sun50iw9p1.dtsi 等) 存在，则遵循以下规则：

1. 相同属性和结点，board.dts 的配置信息会覆盖 *.dtsi 中的配置信息。
2. 新增加的属性和结点，会添加到编译生成的 dtb 文件中。

board.dts配置示例如下：

```
&pio {
    uart0_pins_a: uart0_pins@0 {
        pins = "PB22", "PB23";
        function = "uart0";
    };

    uart0_pins_b: uart0_pins@1 {
        pins = "PB22", "PB23";
        function = "gpio_in";
    };
};

&uart0 {
    pinctrl-names = "default", "sleep";
    pinctrl-0 = <&uart0_pins_a>;
    pinctrl-1 = <&uart0_pins_b>;
    status = "okay";
};
```

1. “pinctrl-0”设备使用状态下的 GPIO 配置，“pinctrl-1”设备休眠状态下的 GPIO 配置；（**注：客户需关注，需与真正使用的 GPIO 引脚一致**）
2. status = "okay"当前uart的状态；（**注：客户需关注，当使用 uart 时需要打开**）
3. pins当前 uart 使用的硬件pin脚；**（注：若开启硬件流控时，需要添加rts及cts引脚，具体的引脚号需查看芯片手册）**

版权所有 © 珠海全志科技股份有限公司。保留一切权利

2.6 模块特殊功能配置方法

2.6.1 AW_UART

2.6.1.1 dma 模式配置

使用前提：

当使用cpu传输，出现系统负载过大、响应不及时、overrun 等问题时，可以使用DMA模式来经降低CPU负载。

未出现上述异常情况，请使用默认的 cpu传输模式，因为dma模式不一定能够提升传输效率。

dma模式配置流程如下：

1. 在内核配置菜单打开CONFIG_AW_SERIAL_DMA配置。
2. 在对应dts配置开启dma，如下所示:

```
&uart6 {
    ......
    use_dma = <3>; /* 是否采用DMA 方式传输，0：不启用，1：只启用TX，2：只启用RX，3：启用TX 与RX */
    dmas = <&dma 14>, <&dma 14>;   /* 14表示DRQ, 查看dma spec */
    dma-names = "tx", "rx";
    ......
};
```

2.6.1.2 485 模式配置

AW_UART 版本驱动只支持软件控制的 485 管脚，需要添加以下成员：

- sunxi,uart-rs485
  - 0：485 模式关闭
  - 1：485 模式使能
- sunxi,uart-485fl
  - 0：485 gpio 管脚电平为低时表示接收状态
  - 1：485 gpio 管脚电平为高时表示接收状态
- sunxi,uart-485oe-gpios
  - 用于外部转换芯片使能信号的 gpio 引脚，GPIO_ACTIVE_HIGH 含义为默认为高电平

示例：

版权所有 © 珠海全志科技股份有限公司。保留一切权利

```
uart1: uart@2500400 {
        ...
        status = "okay";
        //添加以下变更
        sunxi,uart-rs485 = <1>;
        sunxi,uart-485fl = <1>;
        sunxi,uart-485oe-gpios = <&pio PG 8 GPIO_ACTIVE_HIGH>;
    };
```

2.6.2 AW_UART_NG

2.6.2.1 dma 模式配置

使用前提：

当设备传输数据量**很大**时，若使用cpu传输，会导致系统负载过大，响应不及时。因此可以使用DMA模式来经降低CPU负载。

若传输数据量较小时，请使用默认的 cpu传输模式，因为dma模式不一定能够提升传输效率。

dma模式配置流程如下：

1. dma 传输模式下设备树节点需要增加以下配置

```
&uart6 {
   ......
   dmas = <&dma 15>, <&dma 15>;
   dma-names = "tx", "rx"; /* 需要修改为tx/rx，和非dma传输模式有差异 */
   status = "disabled";
   ......
};
```

2.6.2.2 485 模式

需要添加以下成员：

- sunxi,uart-485pin_auto = <1>;（与 sunxi,uart-rs485 都需要配置，是为了兼容不同小版本的驱动代码）
  - 0：485 模式关闭
  - 1：485 模式使能
- sunxi,uart-rs485
  - 0：485 模式关闭
  - 1：软件控制 485 传输方向模式使能

版权所有 © 珠海全志科技股份有限公司。保留一切权利

- sunxi,uart-485fl
  - 0：485 gpio 管脚电平为低时表示接收状态
  - 1：485 gpio 管脚电平为高时表示接收状态
- sunxi,uart-485oe-gpios
  - 用于外部转换芯片使能信号的 gpio 引脚，GPIO_ACTIVE_HIGH 含义为默认为高电平

示例：

```
uart1: uart@2500400 {
        ...
        status = "okay";
        //添加以下变更
        sunxi,uart-485pin_auto = <1>;
        sunxi,uart-rs485 = <1>;
        sunxi,uart-485fl = <1>;
        sunxi,uart-485oe-gpios = <&pio PG 8 GPIO_ACTIVE_HIGH>;
    };
```

在 dtsi 中需对 compatible 进行配置

示例：

```
uart4: uart@2604000 {

    compatible = "allwinner,uart-v100";

    device_type = "uart4";

     .....

} ;
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

3 模块接口说明

UART 驱动会注册生成串口设备/dev/ttySx，应用层的使用只需遵循Linux系统中的标准串口编程方法即可。

3.1 打开/关闭串口

使用标准的文件打开函数：

```
int  open(const char *pathname, int flags);
int  close(int  fd);
```

需要引用头文件：

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
```

3.2 读/写串口

同样使用标准的文件读写函数：

```
ssize_t  read(int  fd,  void *buf,  size_t  count);
ssize_t  write(int  fd,  const void *buf,  size_t  count);
```

需要引用头文件：

```
#include <unistd.h>
```

3.3 设置串口属性

串口属性包括波特率、数据位、停止位、校验位、流控等，这部分是串口设备特有的接口。串口属性的数据结构 termios 定义如下：（terminos.h）。

```
#define NCCS 19
struct  termios {
    tcflag_t  c_iflag;          /*  input mode flags */
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

```
    tcflag_t  c_oflag;      /*  output mode flags */
    tcflag_t  c_cflag;      /*  control mode flags */
    tcflag_t  c_lflag;      /*  local  mode flags */
    cc_t c_line;            /*  line  discipline  */
    cc_t c_cc[NCCS];        /*  control characters */
};
```

其中，c_iflag 的标志常量定义如下：

标志	说明
IGNBRK	忽略输入中的 BREAK 状态。
BRKINT	如果设置了 IGNBRK，将忽略 BREAK。如果没有设置，但是设置了 BRKINT，那么 BREAK 将使得输入和输出队列被刷新，如果终端是一个前台进程组的控制终端，这个进程组中所有进程将收到 SIGINT 信号。如果既未设置 IGNBRK 也未设置 BRKINT，BREAK 将视为与 NUL 字符同义，除非设置了 PARMRK，这种情况下它被视为序列 \377 \0 \0。
IGNPAR	忽略桢错误和奇偶校验错。
PARMRK	如果没有设置 IGNPAR，在有奇偶校验错或桢错误的字符前插入 \377 \0。如果既没有设置 IGNPAR 也没有设置 PARMRK，将有奇偶校验错或桢错误的字符视为 \0。
INPCK	启用输入奇偶检测。
ISTRIP	去掉第八位。
INLCR	将输入中的 NL 翻译为 CR。
IGNCR	忽略输入中的回车。
ICRNL	将输入中的回车翻译为新行 (除非设置了 IGNCR)。
IUCLC	(不属于 POSIX) 将输入中的大写字母映射为小写字母。
IXON	启用输出的 XON/XOFF 流控制。
IXANY	(不属于 POSIX.1；XSI) 允许任何字符来重新开始输出。
IXOFF	启用输入的 XON/XOFF 流控制。
IMAXBEL	(不属于 POSIX) 当输入队列满时响零。Linux 没有实现这一位，总是将它视为已设置。

c_oflag 的标志常量定义如下：

标志	说明
OLCUC	(不属于 POSIX) 将输出中的小写字母映射为大写字母。
ONLCR	(XSI) 将输出中的新行符映射为回车-换行。
OCRNL	将输出中的回车映射为新行符。
ONOCR	不在第 0 列输出回车。
ONLRET	不输出回车。
OFILL	发送填充字符作为延时，而不是使用定时来延时。
OFDEL	(不属于 POSIX) 填充字符是 ASCII DEL (0177)。如果不设置，填充字符则是 ASCII NUL。
NLDLY	新行延时掩码。取值为 NL0 和 NL1。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

标志	说明
CRDLY	回车延时掩码。取值为 CR0, CR1, CR2, 或 CR3。
TABDLY	水平跳格延时掩码。取值为 TAB0, TAB1, TAB2, TAB3 (或 XTABS)。取值为 TAB3，即 XTABS，将扩展跳格为空格 (每个跳格符填充 8 个空格)。
BSDLY	回退延时掩码。取值为 BS0 或 BS1。(从来没有被实现过)。
VTDLY	竖直跳格延时掩码。取值为 VT0 或 VT1。
FFDLY	进表延时掩码。取值为 FF0 或 FF1。

c_cflag 的标志常量定义如下：

标志	说明
CBAUD	(不属于 POSIX) 波特率掩码 (4+1 位)。
CBAUDEX	(不属于 POSIX) 扩展的波特率掩码 (1 位)，包含在 CBAUD 中。(POSIX 规定波特率存储在 termios 结构中，并未精确指定它的位置，而是提供了函数 cfgetispeed() 和 cfsetispeed() 来存取它。一些系统使用 c_cflag 中 CBAUD 选择的位，其他系统使用单独的变量，例如 sg_ispeed 和 sg_ospeed 。)
CSIZE	字符长度掩码。取值为 CS5, CS6, CS7, 或 CS8。
CSTOPB	设置两个停止位，而不是一个。
CREAD	打开接受者。
PARENB	允许输出产生奇偶信息以及输入的奇偶校验。
PARODD	输入和输出是奇校验。
HUPCL	在最后一个进程关闭设备后，降低 modem 控制线 (挂断)。
CLOCAL	忽略 modem 控制线。
LOBLK	(不属于 POSIX) 从非当前 shell 层阻塞输出 (用于 shl)。
CIBAUD	(不属于 POSIX) 输入速度的掩码。CIBAUD 各位的值与 CBAUD 各位相同，左移了 IBSHIFT 位。
CRTSCTS	(不属于 POSIX) 启用 RTS/CTS (硬件) 流控制。

c_lflag 的标志常量定义如下：

标志	说明
ISIG	当接受到字符 INTR, QUIT, SUSP, 或 DSUSP 时，产生相应的信号。
ICANON	启用标准模式 (canonical mode)。允许使用特殊字符 EOF, EOL, EOL2, ERASE, KILL, LNEXT, REPRINT, STATUS, 和 WERASE，以及按行的缓冲。
XCASE	(不属于 POSIX; Linux 下不被支持) 如果同时设置了 ICANON，终端只有大写。输入被转换为小写，除了以 前缀的字符。输出时，大写字符被前缀，小写字符被转换成大写。
ECHO	回显输入字符。
ECHOE	如果同时设置了 ICANON，字符 ERASE 擦除前一个输入字符，WERASE 擦除前一个词。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

标志	说明
ECHOK	如果同时设置了 ICANON，字符 KILL 删除当前行。
ECHONL	如果同时设置了 ICANON，回显字符 NL，即使没有设置 ECHO。
ECHOCTL	(不属于 POSIX) 如果同时设置了 ECHO，除了 TAB, NL, START, 和 STOP 之外的 ASCII 控制信号被回显为 ^X, 这里 X 是比控制信号大 0x40 的 ASCII 码。例如，字符 0x08 (BS) 被回显为 ^H。
ECHOPRT	(不属于 POSIX) 如果同时设置了 ICANON 和 IECHO，字符在删除的同时被打印。
ECHOKE	(不属于 POSIX) 如果同时设置了 ICANON，回显 KILL 时将删除一行中的每个字符，如同指定了 ECHOE 和 ECHOPRT 一样。
DEFECHO	(不属于 POSIX) 只在一个进程读的时候回显。
FLUSHO	(不属于 POSIX; Linux 下不被支持) 输出被刷新。这个标志可以通过键入字符 DISCARD 来开关。
NOFLSH	禁止在产生 SIGINT, SIGQUIT 和 SIGSUSP 信号时刷新输入和输出队列。
PENDIN	(不属于 POSIX; Linux 下不被支持) 在读入下一个字符时，输入队列中所有字符被重新输出。(bash 用它来处理 typeahead)
TOSTOP	向试图写控制终端的后台进程组发送 SIGTTOU 信号。
IEXTEN	启用实现自定义的输入处理。这个标志必须与 ICANON 同时使用，才能解释特殊字符 EOL2，LNEXT，REPRINT 和 WERASE，IUCLC 标志才有效。

c_cc 数组定义了特殊的控制字符。符号下标 (初始值) 和意义为：

标志	说明
VINTR	(003, ETX, Ctrl-C, or also 0177, DEL, rubout) 中断字符。发出 SIGINT 信号。当设置 ISIG 时可被识别，不再作为输入传递。
VQUIT	(034, FS, Ctrl-) 退出字符。发出 SIGQUIT 信号。当设置 ISIG 时可被识别，不再作为输入传递。
VERASE	(0177, DEL, rubout, or 010, BS, Ctrl-H, or also #) 删除字符。删除上一个还没有删掉的字符，但不删除上一个 EOF 或行首。当设置 ICANON 时可被识别，不再作为输入传递。
VKILL	(025, NAK, Ctrl-U, or Ctrl-X, or also @) 终止字符。删除自上一个 EOF 或行首以来的输入。当设置 ICANON 时可被识别，不再作为输入传递。
VEOF	(004, EOT, Ctrl-D) 文件尾字符。更精确地说，这个字符使得 tty 缓冲中的内容被送到等待输入的用户程序中，而不必等到 EOL。如果它是一行的第一个字符，那么用户程序的 read() 将返回 0，指示读到了 EOF。当设置 ICANON 时可被识别，不再作为输入传递。
VMIN	非 canonical 模式读的最小字符数。
VEOL	(0, NUL) 附加的行尾字符。当设置 ICANON 时可被识别。
VTIME	非 canonical 模式读时的延时，以十分之一秒为单位。
VEOL2	(not in POSIX; 0, NUL) 另一个行尾字符。当设置 ICANON 时可被识别。
VSTART	(021, DC1, Ctrl-Q) 开始字符。重新开始被 Stop 字符中止的输出。当设置 IXON 时可被识别，不再作为输入传递。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

标志	说明
VSTOP	(023, DC3, Ctrl-S) 停止字符。停止输出，直到键入 Start 字符。当设置 IXON 时可被识别，不再作为输入传递。
VSUSP	(032, SUB, Ctrl-Z) 挂起字符。发送 SIGTSTP 信号。当设置 ISIG 时可被识别，不再作为输入传递。
VLNEXT	(not in POSIX; 026, SYN, Ctrl-V) 字面上的下一个。引用下一个输入字符，取消它的任何特殊含义。当设置 IEXTEN 时可被识别，不再作为输入传递。
VWERASE	(not in POSIX; 027, ETB, Ctrl-W) 删除词。当设置 ICANON 和 IEXTEN 时可被识别，不再作为输入传递。
VREPRINT	(not in POSIX; 022, DC2, Ctrl-R) 重新输出未读的字符。当设置 ICANON 和 IEXTEN 时可被识别，不再作为输入传递。

3.3.1 tcgetattr

- 作用：获取串口设备的属性。
- 参数：
  - fd，串口设备的文件描述符。
  - termios_p，用于保存串口属性。
- 返回：
  - 成功，返回 0。
  - 失败，返回-1，errnor 给出具体错误码。

3.3.2 tcsetattr

- 作用：设置串口设备的属性。
- 参数：
  - fd，串口设备的文件描述符。
  - optional_actions，本次设置什么时候生效。
  - termios_p，指向要设置的属性结构。
- 返回：
  - 成功，返回 0。
  - 失败，返回-1，errnor 给出具体错误码

说明

其中，optional_actions 的取值有：
TCSANOW：会立即生效。
TCSADRAIN：当前的输出数据完成传输后生效，适用于修改了输出相关的参数。
TCSAFLUSH：当前的输出数据完成传输，如果输入有数据可读但没有读就会被丢弃。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

3.3.3 cfgetispeed

- 作用：返回串口属性中的输入波特率。
- 参数：
  - termios_p，指向保存有串口属性的结构。
- 返回：
  - 成功，返回波特率，取值是一组宏，定义在 terminos.h。
  - 失败，返回-1，errnor 给出具体错误码。

说明

波特率定义如下所示：

```
#define B0 0000000
#define B50 0000001
#define B75 0000002
#define B110 0000003
#define B134 0000004
#define B150 0000005
#define B200 0000006
#define B300 0000007
#define B600 0000010
#define B1200 0000011
#define B1800 0000012
#define B2400 0000013
#define B4800 0000014
#define B9600 0000015
#define B19200 0000016
#define B38400 0000017
#define B57600 0010001
#define B115200 0010002
#define B230400 0010003
#define B460800 0010004
#define B500000 0010005
#define B576000 0010006
#define B921600 0010007
#define B1000000 0010010
#define B1152000 0010011
#define B1500000 0010012
#define B2000000 0010013
#define B2500000 0010014
#define B3000000 0010015
#define B3500000 0010016
#define B4000000 0010017
```

3.3.4 cfgetospeed

- 作用：返回串口属性中的输出波特率。
- 参数：
  - termios_p，指向保存有串口属性的结构。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

- 返回：
  - 成功，返回波特率，取值是一组宏，定义在 terminos.h。
  - 失败，返回-1，errnor 给出具体错误码。

3.3.5 cfsetispeed

- 作用：设置输入波特率到属性结构中。
- 参数：
  - termios_p，指向保存有串口属性的结构。
  - speed，波特率。
- 返回：
  - 成功，返回 0。
  - 失败，返回-1，errnor 给出具体错误码。

3.3.6 cfsetospeed

- 作用：设置输出波特率到属性结构中。
- 参数：
  - termios_p，指向保存有串口属性的结构。
  - speed，波特率。
- 返回：
  - 成功，返回 0。
  - 失败，返回-1，errnor 给出具体错误码

3.3.7 cfsetspeed

- 作用：同时设置输入和输出波特率到属性结构中。
- 参数：
  - termios_p，指向保存有串口属性的结构。
  - speed，波特率。
- 返回：
  - 成功，返回 0。
  - 失败，返回-1，errnor 给出具体错误码

版权所有 © 珠海全志科技股份有限公司。保留一切权利

3.3.8 tcflush

- 作用：清空输出缓冲区、或输入缓冲区的数据，具体取决于参数 queue_selector。
- 参数：
  - fd，串口设备的文件描述符。
  - queue_selector，清空数据的操作。
- 返回：
  - 成功，返回 0。
  - 失败，返回-1，errnor 给出具体错误码。

说明

参数 queue_selector 的取值有三个：
TCIFLUSH：清空输入缓冲区的数据。
TCOFLUSH：清空输出缓冲区的数据。
TCIOFLUSH：同时清空输入/输出缓冲区的数据。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

4 模块使用范例

此 demo 程序是打开一个串口设备，然后侦听这个设备，如果有数据可读就读出来并打印。设备名称、侦听的循环次数都可以由参数指定。

```
#include    <stdio.h>     /*标准输入输出定义*/
#include    <stdlib. h>   /*标准函数库定义*/
#include    <unistd.h>    /*Unix 标准函数定义*/
#include    <sys/types.h>
#include    <sys/stat. h>
#include    <fcntl. h>    /*文件控制定义*/
#include    <termios.h>   /*PPSIX 终端控制定义*/
#include    <errno.h>     /*错误号定义*/
#include <string.h>

enum parameter_type {
    PT_PROGRAM_NAME = 0,
    PT_DEV_NAME,
    PT_CYCLE,

    PT_NUM
};

#define DBG(string, args...)   \
    do { \
        printf("%s, %s()%u---", __FILE__, __FUNCTION__, __LINE__); \
        printf(string, ##args); \
        printf("\n"); \
    } while (0)

void usage(void)
{
    printf("You should input as: \n");
    printf("\t select_test [/ dev/name] [Cycle Cnt]\n");
}

int OpenDev(char *name)
{
    int fd = open(name, O_RDWR );       //| O_NOCTTY | O_NDELAY
    if (-1 == fd)
        DBG("Can't Open(%s)!", name);

    return fd;
}

/**
*@brief 设置串口通信速率
*@param fd     类型 int   打开串口的文件句柄
*@param speed 类型 int 串口速度
*@return void
*/
void set_speed(int fd, int speed){
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

```
    int    i;
    int    status;
    struct termios Opt = {0};
    int speed_arr[] = { B38400, B19200, B9600, B4800, B2400, B1200, B300,
                B38400, B19200, B9600, B4800, B2400, B1200, B300, };
    int name_arr[] = {38400, 19200, 9600, 4800, 2400, 1200, 300, 38400,
                19200, 9600, 4800, 2400, 1200, 300, };

    tcgetattr(fd, &Opt);

    for ( i = 0;  i < sizeof(speed_arr) / sizeof(int);  i++) {
        if  (speed == name_arr[i])
            break;
    }

    tcflush(fd, TCIOFLUSH);
    cfsetispeed(&Opt, speed_arr[i]);
    cfsetospeed(&Opt, speed_arr[i]);

    Opt.c_lflag  &= ~(ICANON | ECHO | ECHOE | ISIG); /*Input*/
    Opt.c_oflag  &= ~OPOST;  /*Output*/

    status = tcsetattr(fd, TCSANOW, &Opt);
    if  (status != 0) {
        DBG("tcsetattr fd");
        return;
    }
    tcflush(fd, TCIOFLUSH);
}

/**
*@brief   设置串口数据位，停止位和效验位
*@param fd     类型  int   打开的串口文件句柄
*@param databits 类型 int 数据位   取值 为 7 或者8
*@param stopbits 类型 int 停止位   取值为 1 或者2
*@param parity 类型 int   效验类型 取值为N,E,O,,S
*/
int set_Parity(int fd,int databits,int stopbits,int parity)
{
    struct termios options;

    if  ( tcgetattr( fd,&options) != 0) {
        perror("SetupSerial 1");
        return -1;
    }
    options.c_cflag &= ~CSIZE;

    switch (databits) /*设置数据位数*/
    {
    case 7:
        options.c_cflag |= CS7;
        break;
    case 8:
        options.c_cflag |= CS8;
        break;
    default:
        fprintf (stderr, "Unsupported data size\n");
        return -1;
    }
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

```
    switch (parity)
    {
    case 'n':
    case 'N':
        options.c_cflag &= ~PARENB;  /* Clear parity enable */
        options.c_iflag &= ~INPCK;    /* Enable parity checking */
        break;
    case 'o':
    case 'O':
        options.c_cflag |= (PARODD | PARENB); /* 设置为奇效验*/
        options.c_iflag |= INPCK;             /* Disnable parity checking */
        break;
    case 'e':
    case 'E':
        options.c_cflag |= PARENB;    /* Enable parity */
        options.c_cflag &= ~PARODD;   /* 转换为偶效验*/
        options.c_iflag |= INPCK;     /* Disnable parity checking */
        break;
    case 'S':
    case 's':  /*as no parity*/
        options.c_cflag &= ~PARENB;
        options.c_cflag &= ~CSTOPB;break;
    default:
        fprintf (stderr, "Unsupported parity\n");
        return -1;
    }

    /* 设置停止位*/
    switch (stopbits)
    {
        case 1:
            options.c_cflag &= ~CSTOPB;
            break;
        case 2:
            options.c_cflag |= CSTOPB;
           break;
        default:
             fprintf (stderr, "Unsupported stop bits\n");
             return -1;
    }

    /* Set input parity option */
    if (parity != 'n')
        options.c_iflag |= INPCK;
    tcflush(fd, TCIFLUSH);
    options.c_cc[VTIME] = 150; /* 设置超时15 seconds*/
    options.c_cc[VMIN] = 0; /* Update the options and do it NOW */
    if (tcsetattr (fd, TCSANOW,&options) != 0)
    {
        perror("SetupSerial 3");
        return -1;
    }
    return 0;
}

void str_print(char *buf, int len)
{
    int i;

    for (i=0; i<len; i++) {
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

```
168             if (i%10 == 0)
169                 printf("\n");
170
171             printf("0x%02x ", buf[i]);
172         }
173         printf("\n");
174     }
175
176     int main(int argc, char **argv)
177     {
178         int i = 0;
179         int fd = 0;
180         int cnt = 0;
181         char buf[256];
182
183         int ret;
184         fd_set rd_fdset;
185         struct timeval dly_tm;      // delay time in select()
186
187         if (argc != PT_NUM) {
188             usage();
189             return -1;
190         }
191
192         sscanf(argv[PT_CYCLE], "%d", &cnt);
193         if (cnt == 0)
194             cnt = 0xFFFF;
195
196         fd = OpenDev(argv[PT_DEV_NAME]);
197         if (fd < 0)
198             return -1;
199
200         set_speed(fd,19200);
201         if (set_Parity(fd,8,1,'N') == -1)  {
202             printf("Set Parity Error\n");
203             exit(0);
204         }
205
206         printf("Select(%s), Cnt %d. \n", argv[PT_DEV_NAME], cnt);
207         while (i<cnt) {
208             FD_ZERO(&rd_fdset);
209             FD_SET(fd, &rd_fdset);
210
211             dly_tm.tv_sec = 5;
212             dly_tm.tv_usec = 0;
213             memset(buf, 0, 256);
214
215             ret = select(fd+1, &rd_fdset, NULL, NULL, &dly_tm);
216     //      DBG("select() return %d, fd = %d", ret, fd);
217             if (ret == 0)
218                 continue;
219
220             if (ret < 0) {
221                 printf("select(%s) return %d. [%d]: %s \n", argv[PT_DEV_NAME], ret, errno, strerror(errno));
222                 continue;
223             }
224
225             i++;
226             ret = read(fd, buf, 256);
227             printf("Cnt%d: read(%s) return %d.\n", i, argv[PT_DEV_NAME], ret);
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

```
228             str_print(buf, ret);
229         }
230
231         close(fd);
232         return 0;
233     }
```

版权所有 © 珠海全志科技股份有限公司。保留一切权利

5 调试方法

- 通过 debugfs 使用命令打开调试开关

注：内核需打开CONFIG_DYNAMIC_DEBUG宏定义

```
1.挂载debugfs。
mount -t debugfs none /sys/kernel/debug
2.打开uart模块所有打印。
echo "module sunxi_uart +p" > /mnt/dynamic_debug/control
3.打开指定文件的所有打印。
echo "file  sunxi-uart.c  +p" > /mnt/dynamic_debug/control
4.打开指定文件指定行的打印。
echo "file  sunxi-uart.c  line  615 +p" > /mnt/dynamic_debug/control
5.打开指定函数名的打印。
echo "func sw_uart_set_termios +p" > /mnt/dynamic_debug/control
6.关闭打印。
把上面相应命令中的+p 修改为-p 即可。
更多信息可参考linux 内核文档：linux-3.10/Documentation/dynamic-debug-howto.txt。
```

5.1 调试工具

5.2 调试节点

5.2.1 /sys/devices/platform/soc*/*uart*/dev_info

从该节点可以看到 UART 端口的一些硬件资源信息。以内核为 linux6.6 的 a733 下的uart0为例：

```
# cat /sys/devices/platform/soc@3000000/2b00000.uart/dev_info
id    = 0
name  = uart0
irq   = 247
io_num = 2
port->mapbase = 0x0000000005000000
port->membase = 0xffffff800b005000
port->iobase  = 0x00000000
pdata->regulator   = 0x       (null)
pdata->regulator_id =
```

- id：uart编号
- name：uart名称
- irq：uart中断号
- io_num：表明 uart0 目前被配置为为 2 线模式

版权所有 © 珠海全志科技股份有限公司。保留一切权利

5.2.2 /sys/devices/platform/soc*/*uart*/status

此节点可以打印出 UART 端口的一些运行状态信息，包括控制器的各寄存器值。以内核为 linux6.6 的 a733 下的uart0为例：

```
# cat /sys/devices/platform/soc@3000000/2b00000.uart/status
uartclk = 24000000
The Uart controller register[Base: 0xffffff800b005000]:
[RTX] 0x00 = 0x0000000d, [IER] 0x04 = 0x00000005, [FCR] 0x08 = 0x000000c1
[LCR] 0x0c = 0x00000013, [MCR] 0x10 = 0x00000003, [LSR] 0x14 = 0x00000060
[MSR] 0x18 = 0x00000000, [SCH] 0x1c = 0x00000000, [USR] 0x7c = 0x00000006
[TFL] 0x80 = 0x00000000, [RFL] 0x84 = 0x00000000, [HALT] 0xa4 = 0x00000002
```

5.2.3 /sys/devices/platform/soc*/*uart*/ctrl_info

此节点可以打印出软件中保存的一些控制信息，如当前 UART 端口的寄存器值、收发数据的统计等。以内核为 linux6.6 的 a733 下的uart0为例：

```
# cat /sys/devices/platform/soc@3000000/2b00000.uart/ctrl_info
 ier : 0x05
 lcr : 0x13
 mcr : 0x03
 fcr : 0xb1
 dll : 0x0d
 dlh : 0x00
 last baud : 115384 (dl = 13)

TxRx Statistics:
 tx    : 61123
 rx    : 351
 parity : 0
 frame  : 0
 overrun: 0
```

重要信息如下：

- tx：当前发出的数据总量
- rx：接收到的数据总量
- frame：frame 错误的次数。
- overrun：overrun 的次数，代表当前 fifo 溢出。

版权所有 © 珠海全志科技股份有限公司。保留一切权利

6 FAQ

6.1 如何切换 uart 口为打印 conole

1. 从 dts 确保设置为console的uart已经使能，如 uart1。

```
uart1: uart@05000400 {
  compatible = "allwinner,sun50i-uart";
  device_type = "uart1";
  reg = <0x0 0x05000400 0x0 0x400>;
  interrupts = <GIC_SPI 1 IRQ_TYPE_LEVEL_HIGH>;
  clocks = <&clk_uart1>;
  pinctrl-names = "default", "sleep";
  pinctrl-0 = <&uart1_pins_a>;
  pinctrl-1 = <&uart1_pins_b>;
  uart1_port = <1>;
  uart1_type = <4>;
  status = "okay"; /* 确保该uart已经使能 */
};
```

2. 修改方案使用的env*.cfg文件，如下所示：

```
console=ttyS1,115200

说明:
ttyS0 <===> uart0
ttyS1 <===> uart1
...
```

6.2 如何确认当前平台支持哪些波特率

波特率的选择取决于 IC 设计，软件只能在限定的范围内选择，可以在 uart spec 的 “波特率设置” 章节中查看当前平台支持的最高波特率及其他常用波特率。

6.3 控制台 1.5M 波特率相关问题

已在部分平台上将控制台波特率调整至 1.5M，在使用这些平台时需要严格遵守以下注意事项

版权所有 © 珠海全志科技股份有限公司。保留一切权利

6.3.1 原因

过往项目的串口打印波特率均为 115200，打印效率低，大约 10 个字符耗时 1ms，存在诸多性能问题。故在 A537/A333 上开始，推行 1.5M 波特率，后续使用需要注意波特率的切换，否则波特率不匹配会出现串口卡住或乱码。

当前各平台串口波特率配置如下：

芯片	波特率
A733	115200
A523	115200
A527	115200
A537	1500000
A333	1500000

6.3.2 硬件要求

使用 1.5M 波特率需要注意如下事项

1. **尽量选择 FT232 的串口线，其次是 CH340，若未使用要求用线，可能会出现乱码、丢数、行数混乱的情况，以下为数据错位示例；**

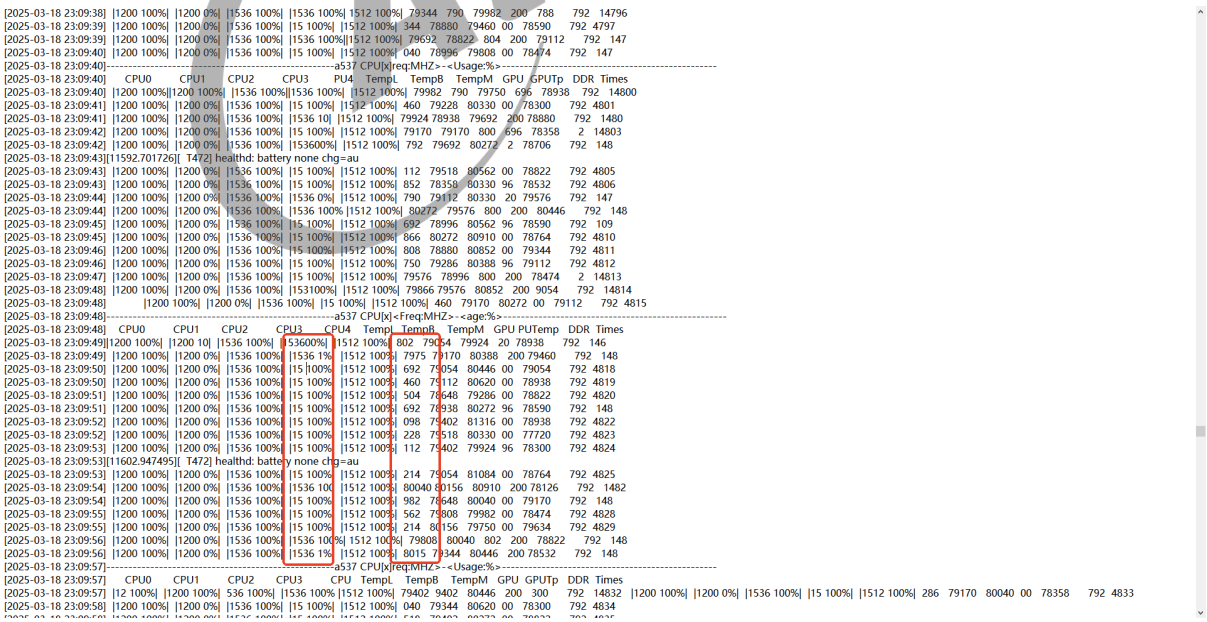

图 6-1: 数据错位示例

2. **串口工具设置为 1.5M 波特率，mobaxterm 和 secureCRT 的波特率选项中没有 1.5M 的选项，可手动输入设置 1.5M。**

版权所有 © 珠海全志科技股份有限公司。保留一切权利

⚠ 注意

PL2303 的串口线不可用

6.3.3 如何切换回 115200 波特率

考虑到终端部分串口线不达要求，存在需要将波特率切回 115200 的需求，而串口波特率切换较为繁琐，故提供下列快速切换方式，便于客户端修改。

1. 修改路径

当前修改波特率只需要修改 sysconfig.fex 中的配置即可，sysconfig.fex 路径如下。

```
longan/device/config/chips/${chip}/configs/${board}/sysconfig.fex
```

⚠ 注意

如果是 GO 平台，如 A333-64bit-go 方案，则使用的 sysconfig.fex 为 sys_config_ddr_3GB.fex。

2. 修改方法

修改 uart_baudrate 的值为 115200，即可将串口输出切换回 115200 波特率，如无该配置，则说明不支持从 sysconfig.fex 中进行配置波特率。

```
[uart_para]
uart_debug_port = 0
uart_debug_tx  = port:PB9<2><1><default><default>
uart_debug_rx  = port:PB10<2><1><default><default>
uart_baudrate  = 1500000
```

⚠ 注意

当前仅 A537 和 A333 平台支持快速切换，后续会支持更多平台。

6.4 关于 uart 波特率

6.4.1 波特率计算公式与大致原理

uart 波特率计算公式为：

apb clk rate = apb parent clk / apb div

baudrate = apb clk rate/ uart divisor / 16

版权所有 © 珠海全志科技股份有限公司。保留一切权利

如果已经理解以上公式可以直接跳转查看 “如何设置当前平台支持某个特定波特率” 章节，如果不理解请先看完本章节，了解基本原理

在不同的 Sunxi 硬件平台中，UART 控制器的时钟源选择、配置略有不同，总体上的时钟关系如下：

```
hosc--------|
clk32K------|
clk16M_RC---|->apb mux->/apb div->apb_clk->uart_clk->/uart divisor-->/16->baud
pll_periX1--|
pll_periX2--|
...---------|
```

uart 波特率依赖 uart 时钟及其上层时钟，整体分为两个部分

- apb 部分
- apb 只是一个代称，在不同平台中名称可以是 apb1、apb_uart、apbs1 等，可以在时钟代码中查看

“ ‘以 1912 平台为例，ccu 文件路径为 SDK/bsp/drivers/clk/sunxi-ng/ccu-sun55iw6.c 在文件中搜索 uart，跳转至任意一个 uart 的时钟描述，可以从下列代码中看出此平台 uart 时钟的时钟源为 apb-uart

static SUNXI_CCU_GATE(uart1_clk, “uart1”, “apb-uart”, 0x0E04, BIT(0), 0);

以上查询方法也可换成在 ccmu spec 中搜索 uart 时钟，并根据 spec 描述确定所用时钟源 “ ‘

- apb 时钟多有种时钟源可以选择，发布 SDK 时，一般选择 hosc 做时钟源，后期用户可在合理范围内自由调整

“ ‘在同一个文件中继续搜索 “apb-uart”, 跳转至时钟定义处，可以看到 apb_uart_clk 可选择的所有时钟源有 5 个

static const char * const apb_uart_parents[] = { “dcxo24M”, “osc32k”, “iosc”, “pll-peri0-600m”, “pll-peri0-480m” }; static SUNXI_CCU_M_WITH_MUX(apb_uart_clk, “apb-uart”, apb_uart_parents, 0x0538, 0, 5, 24, 2, CLK_IS_CRITICAL);

当想要使用 “pll-peri0-480m” 时钟作为 apb_uart_clk 的父时钟时（1）在代码中搜索 “pll-peri0-480m”，跳转至时钟定义处 static SUNXI_CCU_M(pll_peri0_480m_clk, “pll-peri0-480m”, “pll-peri0-parent”, 0x00A0, 2, 3, 0);

（2）根据时钟定义处的时钟变量名称 “pll_peri0_480m_clk” 继续跳转，找到宏定义名称，将 CLK_PLL_PERI0_480M 填入 dtsi [CLK_PLL_PERI0_480M] = &pll_peri0_480m_clk.common.hw

以上查询方法也可换成在 ccmu spec 中搜索 apb_uart 时钟，并根据 spec 描述确定可选时钟源及 div 范围 “ ‘

版权所有 © 珠海全志科技股份有限公司。保留一切权利

- apb 时钟有 div 可对时钟源进行除频，可以将较大的频点整除成较小的频点，一般范围为 1-32，具体除频范围需要查阅 ccmu spec
- uart 部分
- uart clk 直接继承 apb 时钟频点，可以理解为 uart clk rate = apb clk rate
- uart 驱动内部有 uart divisor div 可以对 uart clk 再进行分频，此参数在 uart 内部自动设置，无法也无需手动调整，分频范围是 0-65535

6.4.2 如何计算当前配置下的误码率

需要知道以下信息：

- apbrate（apb 时钟频点）
- baudrate（期望波特率）
- 期望分频系数 uart divisor = apbrate / 16 / baudrate（可能是小数）
- 实际分频数 actual divisor = 四舍五入 uart divisor

误码率 = （uart divisor - actual divisor）/ uart divisor

连续传输数据量越大，对误码率要求越高，推荐不超过 1.725%，如果超过，可以考虑使用其他大小接近但误码率更小的波特率

```
示例1：
apb=24M
baudrate=115200
uart divisor = apbrate / 16 / baudrate = 24000000/16/115200=13.0208
actual divisor =13
误码率 = （13.0208 - 13）/ 13.0208 = 0.16%

示例2：
apb=160M
baudrate=10M
uart divisor = apbrate / 16 / baudrate = 160M/16/10M=1
actual divisor =1
误码率 = （1 - 1）/ 1 = 0%
```

6.4.3 如何设置当前平台支持某个特定波特率

上一章节已经描述了波特率的计算公式，并讲解了其计算原理，本章节将描述如何判断是否需要修改 dtsi 以及怎样修改

版权所有 © 珠海全志科技股份有限公司。保留一切权利

6.4.3.1 判断默认配置是否满足波特率要求

考虑到功耗，UART dts 中默认使用 hosc 做时钟源，一般是 24M

当不设置 apb 分频以及 uart divisor 时，baudrate = 24MHz/1/1/16=1.5MHz，即此时最大仅可支持 1.5M 的波特率

注意：最大支持 1.5M 不等于 1.5M 以下的波特率都可以支持！频点与波特率是离散的对应关系！

此时有两种情况需要修改 dtsi:

- 想要使用的波特率大于 1.5M
- 虽然想要使用的波特率小于 1.5M，但当前配置计算出来的波特率与目标波特率相差过大，误码率过大

6.4.3.2 修改 dtsi 中的时钟配置

```
        device_type = "uart2";
        reg = <0x0 0x05000800 0x0 0x400>;
        interrupts = <GIC_SPI 2 IRQ_TYPE_LEVEL_HIGH>;
-               clocks = <&ccu CLK_BUS_UART0>;
+               clocks = <&ccu CLK_BUS_UART0>,
+                        <&ccu CLK_APB2>,
+                        <&ccu CLK_PSI_AHB1_AHB2>;
+               clock-frequency = <50000000>;
        pinctrl-names = "default", "sleep";
```

以上述代码修改为例讲解 dtsi 中有关时钟的两个重要节点

- clocks 节点
  - 当填写三个时钟时，依次为 uart 时钟名称、apb 时钟名称、apb 时钟源名称，需要按照 ccmu 驱动代码中的命名来填写时钟宏名
- clock-frequency 节点
  - 其中数值为 **apb 时钟频点**，需要依照波特率公式 + 想要设置的波特率倒推符合要求的时钟频点
  - 波特率不在这里指定，而是在使用时通过接口灵活配置
- 重要注意事项！！！
  - 由于所有 uart 的时钟源都来自于同一个 apb 时钟，因此 dtsi 的修改需要同步到所有的 uart 节点中，不可以制只修改其中有一个，否则会造成乱码等问题
  - 修改 APB 总线时钟，可能会影响到使用到 APB 总线时钟的相应模块，例如 twi 等，需要根据 ccmu spec 查询是否会影响到其他模块

示例 1：

版权所有 © 珠海全志科技股份有限公司。保留一切权利

例如需要配置uart2的波特率为460800HZ，先按照波特率公式倒推符合要求的时钟频点
baudrate = apb clk rate/ uart divisor / 16可以转换成apb clk rate = baudrate * uart divisor * 16，可以先假设uart divisor=1，最小的符合要求的apb clk rate
apb clk rate = 460800 * 1 * 16 = 7372800
只要apb时钟是7372800的整数倍就可以支持到此频点，由于7372800无法直接用常用的480M、600M等整除，只能将时钟设置到接近的频点

例如：
50M接近7372800*7=51609600，在apb时钟可分出50M的情况下（如600M/12），就可以选取50M 作为apb的时钟频点，此时真实波特率为50M/7/16=446428
30M接近7372800*4=29491200，在apb时钟可分出30M的情况下（如480M/16），也可以选取30M作为apb的时钟频点，此时真实波特率为30M/4/16=468750

可以看出不同时钟选择造成误码率不同，符合外设要求即可使用
如不想进行计算，可以在驱动代码中找到注释，已给出一些常用波特率与对应的时钟频点,注意需要自行计算误码率判断是否可以使用

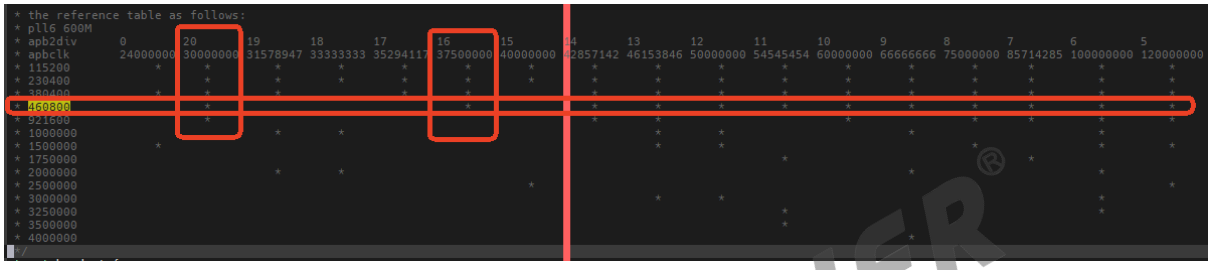

图 6-2: image-20250310175031687

示例 2：

例如需要配置uart2的波特率为10M，先按照波特率公式倒推符合要求的时钟频点
baudrate = apb clk rate/ uart divisor / 16可以转换成apb clk rate = baudrate * uart divisor * 16，可以先假设uart divisor=1，最小的符合要求的apb clk rate
apb clk rate = 10000000 * 1 * 16 = 16000000
只要apb时钟是16000000的整数倍就可以支持到此频点，由于160000000可以直接用常用的480M整除，可以将时钟设置到精确的频点
此时需要先确定平台是否支持将apb parent选到pll_peri_480M，若支持，可将apb时钟频点设置到160M，波特率为160M/1/16=10M
若不支持，则设计上无法支持到10M波特率

6.4.4 如何设置不同的波特率

注意 “设置波特率” 和 “设置支持某个波特率的时钟” 是不同的概念

- 设置波特率

指的是在用户使用 uart 时，可以通过cfsetispeed、cfseospeed等接口设置某笔传输的波特率，是可以灵活配置的，每一次传输都可以重新设置

- 设置支持某个波特率的时钟

版权所有 © 珠海全志科技股份有限公司。保留一切权利

指的是在用户有想要使用的特定波特率的情况下，发现当前时钟无法支持到这个特定的波特率，就需要提前在 dtsi 文件中设置 uart 父时钟及频点、父时钟的父时钟，设置后需要重新编译，生成固件后就不可再更改

6.5 如何测试 UART 回环功能

回环测试分为硬件回环与软件回环：

- 若需要使用硬件回环，需要短接 uart 的 tx 与 rx 引脚
- 若需要使用软件回环，可以在/sys/devices/platform路径下找到uart的loopback节点，使用如下指令即可配置软件回环：

```
echo enable > loopback
```

- 关闭软件回环

```
echo disable > loopback
```

若需要测试uart回环模式是否正常工作，可向我司获取uarttest文件进行测试。或者参考 “模块使用范例” 章节。

注意：除了"enable"，其他所有字符都表示不开启回环

6.6 接收数据时为何分段

当波特率小于等于 9600 时，uart 接收数据可能会分段。

比如一次性发送数据 “12345678”，但是接收会分多次，第一次接收 “1234”，第二次接收 “5678”。现象类似下图：

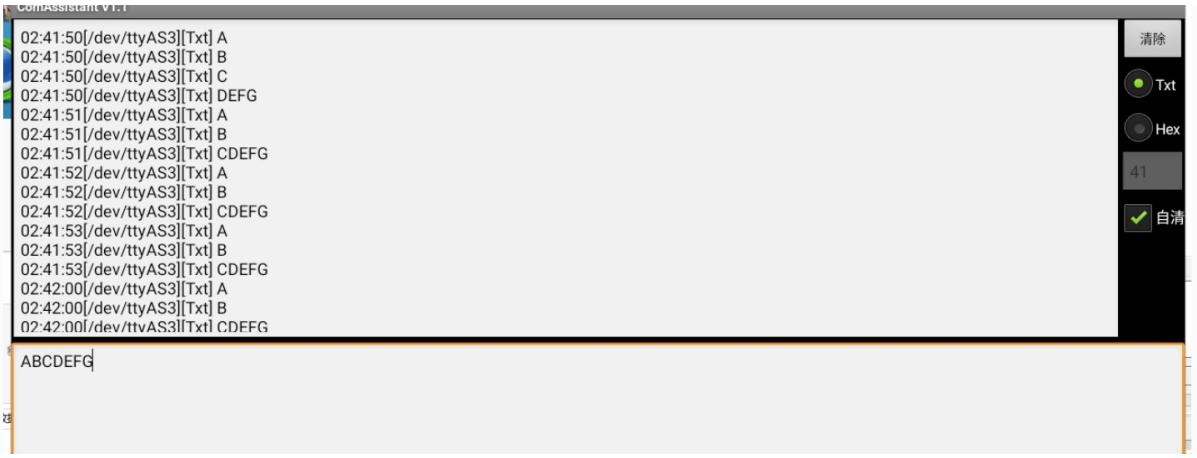

图 6-3: 接收数据分段

版权所有 © 珠海全志科技股份有限公司。保留一切权利

根本原因：

- 当波特率小于等于 9600 时：

硬件接收到 1 个字节数据时，即触发接收中断。此时 cpu 运行速率远大于 uart 传输速率，因此 cpu 读取完一个字节之后就会立刻上报。此时用户层 read 线程被唤醒，大概率只能读到一个字节数据。

- 大于 9600 波特率时：

触发接收中断的场景有两种：（1）当接收数据到达 1/2 FIFO

（2）未到达 1/2 FIFO，但在一定的时间未接收到数据

此时 cpu 触发中断的时间会有一定延迟，FIFO 中会有多个数据存留。此时用户层 read 线程被唤醒，大概率会完整读到整包数据。

修改方法：

```
@@ -1245,14 +1245,9 @@ static void sw_uart_set_termios(struct uart_port *port, struct ktermios *termios
      */
     sw_uart_reset(sw_uport);

-    if (baud <= 9600)
-        sw_uport->fcr = SUNXI_UART_FCR_RXTRG_1CH
-                | SUNXI_UART_FCR_TXTRG_1_2
-                | SUNXI_UART_FCR_FIFO_EN;
-    else
-        sw_uport->fcr = SUNXI_UART_FCR_RXTRG_1_2
-                | SUNXI_UART_FCR_TXTRG_1_2
-                | SUNXI_UART_FCR_FIFO_EN;
+    sw_uport->fcr = SUNXI_UART_FCR_RXTRG_1_2
+            | SUNXI_UART_FCR_TXTRG_1_2
+            | SUNXI_UART_FCR_FIFO_EN;

     serial_out(port, sw_uport->fcr, SUNXI_UART_FCR);
```

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

、、全志科技、（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。