UART FAQ排查指南

文件标识：RK-PC-YF-164

发布版本：V1.1.0

日期：2021-12-29

文件密级：□绝密 □秘密 □内部资料 ■公开

瑞芯微电子股份有限公司

Rockchip Electronics Co., Ltd.

地址： 福建省福州市铜盘路软件园A区18号

网址： www.rock-chips.com

客户服务电话： +86-4007-700-590

客户服务传真： +86-591-83951833

客户服务邮箱： fae@rock-chips.com

前言

概述

本文提供UART开发和维护过程中一些常见问题的解决方案，并将例举一些Rockchip芯片平台上已经解决的UART问题作为参考。读者可以根据UART问题的分类寻找对应的解决方案。

产品版本

芯片名称	内核版本
所有采用Linux内核的芯片	Linux Kernel

读者对象

本文档（本指南）主要适用于以下工程师：

技术支持工程师

软件开发工程师

修订记录

版本号	作者	修改日期	修改说明
V1.0.0	刘诗舫	2021-05-21	初始版本
V1.1.0	刘诗舫	2021-12-29	更新版本

目录

使用说明

相关文档

使用本文前，请先阅读以下文档，确保对Rockchip UART的功能特点、使用方法、调试手段等内容理解充分：

- *Rockchip_Developer_Guide_UART_CN*：Rockchip Linux Kernel UART开发指南。
- *Rockchip_Developer_Guide_UBoot_Nextdev_CN*中的UART章节：Rockchip Uboot UART开发指南。

UART根据使用场景和具体软硬件平台的不同，相同的问题在解决方案上会存在差异，请灵活处理。请相信Rockchip平台SDK中UART基础功能的健壮性，每个UART的基础数据收发功能都是经过测试的。如果正在使用版本较早的SDK，请获取最新SDK源码并对比UART驱动代码。如有疑问，请通过Rockchip Redmine平台联系FAE获取技术支持。

开机日志

根据开机日志中各个阶段启动的标志性打印，可以判断出UART问题出现的阶段，从而针对性地排查UART问题。Rockchip芯片平台开发板上电或复位后，以RK356x为例，常见的开机日志和需要注意的打印内容如下：

```
DDR Version <version information>         # DDR阶段（TPL阶段）打印
...
U-Boot SPL board init                     # Miniloader阶段（SPL阶段）打印
U-Boot SPL <version information>
...
INFO:    Preloader serial: 2              # Trust阶段（BL31阶段）打印
NOTICE:  BL31: <version information>
NOTICE:  BL31: Built : <version information>
...
I/TC: Rockchip release version: 1.0       # OP-TEE阶段（BL32阶段）打印
I/TC: OP-TEE version: <version information>
...
U-Boot <version information>              # U-Boot阶段打印
Model: Rockchip RK3568 Evaluation Board
PreSerial: 2, raw, 0xfe660000
DRAM:  2 GiB
```

```
Sysmem: init
...
Starting kernel ...
[    0.000000] Booting Linux              # Linux Kernel earlycon打印
[    0.000000] Linux version <version information>
[    0.000000] Machine model: Rockchip RK3568 EVB2 LP4X V10 Board
[    0.000000] earlycon: uart8250 at MMIO32 0x00000000fe660000
[    0.000000] bootconsole [uart8250] enabled
...
[    0.193783] console [ttyFIQ0] enabled # 切换到fiq_debugger打印
[    0.194484] bootconsole [uart8250] disabled
...
[    0.411356] Serial: 8250/16550 driver # 普通UART初始化
[    0.412586] fe650000.serial: ttyS1 at MMIO 0xfe650000 (irq = 67, base_baud =
1500000) is a 16550A
[    0.413461] fe690000.serial: ttyS5 at MMIO 0xfe690000 (irq = 68, base_baud =
1500000) is a 16550A
```

io命令

Rockchip平台中通常集成io命令用于执行直接读写寄存器的操作。io命令源代码在安卓SDK external目录下。下面以RK3568 UART5_M0为例，说明使用io命令进行调试的常用操作。

确保UART5 pclk打开，才能正确操作UART5控制器的寄存器。可以直接根据TRM手册操作CRU模块对应寄存器，也可以使用以下命令：

```
echo 1 > /sys/kernel/debug/clk/pclk_uart5/clk_enable_count
```

使用以下命令查看UART5控制器全部寄存器：

```
io -4 -l 0x100 0xFE690000
```

确保UART5 sclk打开，UART5才能正常工作：

```
cat /sys/kernel/debug/clk/clk_summary | grep uart5    # UART5相关时钟
cat /sys/kernel/debug/clk/sclk_uart5/clk_rate         # UART5工作时钟频率
echo 24000000 > /sys/kernel/debug/clk/sclk_uart5/clk_rate
cat /sys/kernel/debug/clk/sclk_uart5/clk_enable_count # UART5工作时钟状态
echo 1 > /sys/kernel/debug/clk/sclk_uart5/clk_enable_count
```

确保UART5_M0 IOMUX配置正确，才能从目标引脚得到相应信号，使用的寄存器需要查找TRM中GRF章节：

```
io -4 0xFDC60310    # uart5_iomux_sel，选择iomux group 0 或 group 1
fdc60310:  00000000 # bit 0为0，说明UART5_M0配置正确
io -4 0xFDC60020    # gpio2a1_sel和gpio2a2_sel，rx和tx的GPIO function
fdc60020:  00001331 # bit 7:4和bit 11:8均为3，说明UART5_M0 RX和TX配置正确
```

通过直接配置UART5控制器寄存器，实现UART5在115200波特率下的基础收发功能：

```
io -4 0xFE690088 0x00000007 # SRR寄存器，对UART和FIFO进行reset
io -4 0xFE690010 0x00000010 # MCR寄存器，将UART配置成loopback模式
io -4 0xFE69000C 0x00000080 # LCR寄存器，div_lat_access置1
io -4 0xFE690000 0x0000000D # DLL寄存器，配置波特率分频系数
io -4 0xFE690004 0x00000000 # DLH寄存器，配置波特率分频系数
io -4 0xFE69000C 0x00000003 # LCR寄存器，div_lat_access清0和配置UART协议参数
io -4 0xFE690010 0x00000000 # MCR寄存器，将UART配置成一般模式
io -4 0xFE690004 0x00000001 # IER寄存器，打开接收中断
io -4 0xFE690008 0x00000041 # FCR寄存器，打开FIFO，FIFO触发阈值配置为1/4
io -4 0xFE69009C            # SRT寄存器，读写RX FIFO触发阈值
io -4 0xFE6900A0            # STET寄存器，读写TX FIFO触发阈值
io -4 0xFE690000 0x00000055 # THR寄存器，发送“U”，0x55
io -4 0xFE690000            # RBR寄存器，上位机发送“U”，读取到0x55
```

注意，配置DLL和DLH时，需要先将LCR寄存器的bit 7 （div_lat_access）置1。在执行此操作前，又需要先将MCR寄存器的bit 4 （loopback）置1，即将UART配置为loopback模式。如果UART不在loopback模式下，当配置DLL和DLH时UART RX引脚受到外部信号干扰，将会触发UART busy状态导致LCR寄存器的bit 7 （div_lat_access）无法重新配置为0，出现错误。

关闭其它打印干扰

调试过程中可以使用以下命令关闭其它打印干扰：

```
su
echo 0 > /proc/sysrq-trigger
```

波特率相关

UART在某个波特率下能否正常工作，主要取决于UART控制器内部分频和CRU提供给UART控制器的工作时钟分频策略。UART控制器内部分频操作寄存器DLL和DLH。CRU提供给UART控制器的工作时钟通常命名为uart_sclk，根据UART控制器的要求与波特率保持16倍的关系。以下几点需要注意：

- 为了满足精度要求，请使用常见的波特率，如115200、460800、921600、1.5M、3M、4M等，Rockchip对于非典型波特率的支持不做保证。驱动框架中更多常见波特率的支持，请查询驱动代码include/uapi/asm-generic/termbits.h。
- Rockchip UART IP最高稳定支持4M波特率，对于更高波特率的支持不做保证，且需要对应修改UART驱动框架层代码。工作时钟uart_sclk的后端约束频率sign_off值通常设计为100MHz，特殊使用场景下也请不要超过6M波特率。

如果正在使用的SDK版本较为老旧，请联系Rockchip更新UART相关源码。

时钟分频策略

在Linux Kernel 4.19中，主要关注以下驱动文件和函数代码：

```
drivers/tty/serial/8250/8250_dw.c     # dw8250_set_termios
drivers/clk/clk-fractional-divider.c # clk_fd_set_rate
drivers/clk/rockchip/clk.c            # rockchip_fractional_approximation
```

在drivers/tty/serial/8250/8250_dw.c的dw8250_set_termios函数中，波特率为115200及以下的时钟源选择24M晶振，波特率在115200之上的时钟源将选择PLL，由CRU控制。对于某些特殊波特率，CRU经过分频后提供的工作时钟可能误差较大，这时在UART控制器向CRU请求工作时钟时可以通过再乘以2（sclk_rate = baud_rate x 16 x 2）来改善。

为了减少CRU输入到UART控制器的工作时钟的jitter，CRU的小数分频器存在一个20倍关系的限制。但由于UART控制器内部存在分频器，即使解除20倍限制后，信号也是满足一般需求的。解除20倍限制请查看drivers/clk/rockchip/clk.c的rockchip_fractional_approximation函数中的标志CLK_FRAC_DIVIDER_NO_LIMIT。注意，某些以前的SDK中可能不包含此功能的补丁，请联系Rockchip FAE获取。

案例

波特率测量方法错误

出现问题时，波特率测量请一定使用示波器。示波器抓取UART TX信号波形才能得到真实的波特率。使用逻辑分析仪、USB转UART小板等方式进行测量的结果都是不一定可靠。通常获取并对比以下三个值就能定位问题：

- 使用示波器抓取UART TX实际输出波形的波特率。
- 打印对应UART的工作时钟频率。
- 根据时钟分频策略计算出的理论UART的工作时钟频率。

打印出的时钟频率正常，测量波特率异常，请检查测量波特率的过程是否有误。时钟分频策略的理论值与打印出来实际输入UART控制器的值有差异，请尝试修改时钟分频策略或更换其它波特率。

使用过程中出错，重启后正常

在使用过程中，例如老化测试等，一段时间后会出现数据传输错误，请根据时钟分频策略检查时钟源。如果时钟源选择CRU提供的PLL，请查看出现问题前后PLL是否出现变化。UART控制器波特率由工作时钟频率决定。使用过程中的某些操作，例如休眠唤醒、变频等会导致PLL实际频率发生变化，进而影响UART波特率，造成数据传输过程中实际波特率改变。

波特率由高到低切换后异常

如果出现从低波特率切换到高波特率正常，从高波特率切换到低波特率异常。请查看出现问题前后，UART工作时钟源是否在CRU提供和晶振提供之间切换正常。通常Rockchip使用24MHz晶振，根据UART控制器和工作时钟的16倍关系换算，1.5M及以下波特率将选择24MHz晶振作为时钟源，1.5M以上波特率将选择CRU提供的PLL时钟作为时钟源。通过打印问题出现前后的UART工作时钟信息可以进行定位。

引脚复用相关

一个UART控制器的信号在引脚分配上通常有多个iomux。配置UART iomux需要同时配置GRF中的两处寄存器，出现问题时也需要通过io命令检查这两处寄存器的值是否正确：

- 选择GPIO引脚的function：同一个引脚会与多个模块的功能进行复用，需要确认该寄存器的值是否已经配置成UART的对应功能。
- 选择uart_iomux_sel寄存器中的group：一个UART控制器需要整体配置group才能实现整体切换所有信号引脚。通常情况下在GRF的pinctrl代码中能找到相应配置，但是由于这部分代码是手动添加，Rockchip无法保证iomux group配置的完备性和准确性，需要确认该寄存器的值是否已经配置成想要切换的iomux group。

案例

不同iomux的TX可以同时使用但RX不能

Rockchip设计iomux是为了更好地利用引脚资源，同一个UART控制器只能选择一组iomux group。iomux group是否成功切换需要检查UART RX功能。因为在UART M0和M1两组iomux的TX引脚中并未加入选择开关，所以可能出现只要对应GPIO引脚的iomux的function都配置成UART功能，两个TX的引脚都会有数据输出的情况。但是在UART M0和M1两组iomux的RX引脚中是存在选择开关的，所以不会

出现两个RX的引脚都能接收UART数据的情况。

开启其他模块后串口异常

由于存在引脚复用，某些模块开启后会去重新配置iomux相关寄存器，导致串口传输错误。常见的PWM（显示设备中使用的PWM背光）、SDMMC（系统启动的存储位置）、JTAG（force jtag位需要保持关闭）等功能，在使用不当的情况下都会与UART的iomux冲突。尤其是用于console的UART的对应引脚，其iomux function涉及的模块使用时需要特别注意，配置冲突会导致打印出错。出现此类问题时，请确认对应引脚的iomux function寄存器的值来定位。

引脚电平相关

UART引脚电平配置错误会导致通讯失败。通常为了符合UART协议，引脚RX和TX都会配置为内部上拉，某些Rockchip旧平台上可以配置为RX上拉，TX保持芯片默认上下拉配置不变。出现问题时，软件上需要检查：

- 确认kernel dts中UART RX和TX对应引脚电平配置是否正确。
- 根据芯片的datasheet，检查电源域io domain的配置（1v8或3v0）是否正确。
- 直接测量UART RX和TX引脚电平是否为高电平，且电压值是否正确。

硬件上的技术支持请联系Rockchip硬件工程师。

案例

硬件PCB设计错误

如果确认软件配置正确后仍无法解决问题，请检查硬件PCB设计，包括但不止以下几种情况：

- 硬件PCB RX和TX信号线接反、接到低电平或高电平。
- UART打开流控，CTS引脚被外部电平强制拉高。
- 外设实际电平不匹配、电压值错误、没有做漏电保护等。

硬件设计与实际使用的芯片型号不一致

Rockchip平台的芯片通常会按照系列推出，同一系列的芯片存在不同型号的细分，在命名上会在系列名称后增加型号尾缀。需要确认硬件参考设计时使用的芯片手册是否是实际使用的型号。遇到此类问题，可以使用Rockchip提供的EVB板进行相同测试，并关注引脚电平情况。

外设TX无法拉低主控RX

出现外设TX无法拉低主控RX从而无法启动UART传输的问题时，请先检查外设是否能拉低其它GPIO引脚。进一步检查外设和主控中间是否存在电平转换芯片、外设引脚驱动能力是否足够等。

中断相关

UART中断包括TX发送中断和RX接收中断。因为UART控制器一般都会使用FIFO，所以中断相关问题需要和FIFO配置结合分析。主要关注以下几组寄存器：

- IER寄存器：中断使能寄存器。
    - 0x80 PTIME：Programmable THRE Interrupt Mode Enable。提高中断发送传输效率，在TX FIFO中数据触发TX FIFO水线时，提前产生THRI中断，准备下一笔发送数据。
    - 0x08 MSI：Modem Status Interrupt。第四优先级，UART Modem触发的中断，相关寄存器为MCR和MSR。
    - 0x04 RLSI：Receive Line Status Interrupt。最高优先级，UART Line触发的中断，包括overrun/parity/framing errors、break interrupt、address received interrupt。相关寄存器为LCR和LSR。

    - 0x02 THRI：Transmitter Holding Register Interrupt。第三优先级，发送中断，当PTIME未使能，发送数据TX FIFO为空时触发。当PTIME使能，发送数据到达或低于TX FIFO水线时触发。
    - 0x01 RDI：Receive Data Interrupt。第二优先级，接收中断，接收数据到达或高于RX FIFO水线时触发。或者当RX FIFO中已经有数据，但在最近的4个字符的时间内没有接收到新的数据，将会产生Timeout触发中断。
- IIR寄存器：中断识别寄存器，读取触发中断的子中断号。除了IER寄存器中使能的四类子中断，还包括第五类最低优先级的Busy Detect Indication中断。
- FCR寄存器：FIFO控制寄存器，操作RX FIFO和TX FIFO的水线配置等的控制。

案例

配置接收中断的触发水线

UART数据传输时，接收中断的触发水线，即FCR寄存器中RX FIFO Trigger Level的配置很关键。Rockchip UART RX FIFO通常为64 Bytes或32 Bytes，可以配置为以下四种触发水线：

- 2'b00: 1 character in the FIFO
- 2'b01: FIFO 1/4 full
- 2'b10: FIFO 1/2 full
- 2'b11: FIFO 2 less than full

UART RX FIFO触发水线越低，接收数据处理越及时，但是将会产生较多中断消耗CPU资源。通常默认配置为FIFO 1/2 full，在Linux Kernel UART驱动代码drivers/tty/serial/8250/8250_port.c中的serial8250_do_set_termios()函数里修改UART_FCR_R_TRIG_10参数。如果出现UART接收数据错误，可以尝试修改RX FIFO触发水线为2'b00，即每收到一个字符就产生一次中断。

DMA相关

Rockchip芯片平台的DMA控制器通常为PL330，部分型号的芯片使用DW的控制器。实际DMA控制器的型号请查阅对应芯片的TRM手册，本章内容如果没有特殊说明，均针对PL330 DMA控制器。因为UART存在传输数据量不定长的特征，所以在使用PL330控制器的芯片中，UART对于DMA的配置策略比较特殊。以下几点需要注意：

- 默认配置DMA传输为burst模式，但burst length为1，即1个byte触发一次DMA请求。UART的这种配置方式是由PL330控制器的特性决定的。由于在UART中断传输模式下，FCR寄存器中的FIFO水线默认配置为2'b10，即1/2 FIFO触发，这与DMA配置不匹配。因此，需要确认FCR寄存器中的FIFO水线是否已经修改配置为2'b00，即1个byte触发一次。详细配置UART FIFO的触发水线方法请参考中断相关章节。
- 可以构造burst length对齐的传输数据，为了提高UART DMA的传输效率，可以让传输双方严格按照burst length的大小进行对齐，一次传输中的最后一笔数据按照这种对齐方式进行数据补齐。注意，burst length需要跟UART FIFO的触发水线相匹配。
- UART在使用DMA模式接收时，由于接收中断存在Timeout机制。如果配置的burst length和UART FIFO的触发水线超过1个byte，每一笔数据的最后一包又没有补充冗余数据进行对齐。那么，每一笔数据的最后一包数据将会通过Timeout机制使用CPU中断进行接收。这种配置下，CPU中断接收数据和DMA请求接收数据存在竞争关系，有一定的接收数据丢失的风险。

使用DW的DMA控制器不会出现以上问题，但为了统一DMA控制器的配置策略，仍使用同种方案。在使用DMA的过程中，需要注意以下几点：

- 确认每一笔DMA数据的开始位置和结束位置。
- 确认下一笔DMA数据传输开始前，DMA控制器的开关状态。

案例

不同版本kernel的DMA配置可能不同

此类问题通常出现在客户升级系统版本，移植DTS中的DMA配置参数时，直接沿用之前的配置而没有确认不同版本kernel的DMA配置差异。例如，在kernel 4.4中，burst模式的配置参数为peripherals-req-type-burst，而在kernel 4.19中，burst模式的配置参数为arm,pl330-periph-burst。

高波特率下DMA模式接收数据出现概率性丢失

在高波特率下使用DMA接收数据时，由于DMA接收配置的RX buffer采用循环覆盖的方式，如果RX buffer配置较小，还未收走的数据会被覆盖，从而出现概率性丢失的情况。可以尝试通过修改驱动代码drivers/tty/serial/8250/8250_dma.c中serial8250_request_dma函数的rx_size大小来解决，其中一个PAGE_SIZE大小为4k。

硬件流控相关

建议在高波特率（1.5M波特率及以上）、大数据量的场景下都使用硬件自动流控模式，即使用四线UART。硬件自动流控模式增加RTS和CTS引脚，相比DMA模式的传输可靠性更高。注意，在Rockchip平台上使用UART，请不要使用软件流控。

案例

打开硬件流控后无法发送数据

此类问题通常是硬件问题，可以从以下方向进行排查：

- 开发板硬件设计错误、硬件故障等导致CTS引脚被强制拉高。
- 对方RTS引脚无法正常拉高或拉低，导致连接后一直将我方CTS引脚拉高。

数据接收错误相关

UART数据接收错误的问题分为数据接收不到和数据接收出现乱码两类，可以从以下方向进行排查：

- 硬件层面：确认RX引脚是否配置为内部上拉、外围电路是否有电压电流异常、芯片引脚电平是否正确配置等。如果使用USB转串口小板，请尝试更换更优质的串口线和工具。
- 驱动层面：使用io命令，关闭IER寄存器中断，运行接收测试程序，检查RFL寄存器中是否有数据停留。确认FCR寄存器配置的RX FIFO触发水线是否满足需求。
- 应用层面：确认应用程序是否有取走UART FIFO中的数据、是否有其它应用程序也在使用同一个UART、应用程序处理上报数据的策略是否与RX FIFO触发水线匹配、应用程序配置给UART接收的buffer是否满足需求等。

如果正在使用的SDK版本较为老旧，请联系Rockchip FAE更新UART相关源码。请仔细阅读本文档其它章节，结合*Rockchip_Developer_Guide_UART_CN*中的调试方式进行问题排查。

案例

对方数据发送错误

一定要确认导致UART出现错误的原因，是我方数据接收错误还是对方数据发送错误。可以使用示波器、USB转UART小板等工具进行检测。如果使用以上排查手段进行测试后，我方UART数据接收均正常。请仔细检查对方数据发送是否符合要求。

一次接收大量数据会出现分段

遇到此类问题，请使用UART中断传输模式，并确认FCR寄存器中FIFO触发水线的大小。UART中断接收机制中，通过读取IIR寄存器触发中断的ID号，会对4'b0100接收数据触发水线产生的中断和4'b1100接收数据Timeout产生的中断两种情况分别处理并上报给应用层。

请查看UART驱动代码中数据上报逻辑是否符合要求。注意以下代码中参数max_count的值是否与循环接收字符的次数一致：

- kernel 3.10在drivers/tty/serial/rk_serial.c的receive_chars函数中的max_count。
- kernel 4.4和4.19在drivers/tty/serial/8250/8250_port.c的serial8250_rx_chars函数中max_count。

应用层读取数据出现延迟

由于Linux Kernel并非实时系统，UART驱动通过Work Queue的方式上报数据给应用层。如果Work Queue被阻塞，可能会导致应用层读取数据出现延迟。在kernel 3.10中，UART驱动框架提供了中断上报的方式，可以提高实时性。修改补丁如下：

```
diff --git a/drivers/tty/serial/rk_serial.c b/drivers/tty/serial/rk_serial.c
index 9870873c1200..ff81be78a53a 100644
--- a/drivers/tty/serial/rk_serial.c
+++ b/drivers/tty/serial/rk_serial.c
@@ -2074,6 +2074,9 @@ static int serial_rk_probe(struct platform_device *pdev)
        up->port.uartclk = 24000000;
 #endif

+       /* update recv data quickly instead of workqueue */
+       up->port.flags |= UPF_LOW_LATENCY;
```

在kernel 4.4和kernel 4.19中，UART驱动框架只支持Work Queue上报数据给应用层。如果对实时性有较高要求，请使用RTOS。

连接外设或其它控制器相关

UART连接蓝牙、WIFI、NFC等外设或其它控制器出现数据传输异常时，可以从以下方向进行排查：

- 确认是主控端出错还是设备端出错：可以通过对应设备的协议分析仪或者使用两个USB转UART小板分别连接到主控端UART的RX和TX引脚上，抓取传输过程中的往来数据。如果是主控端出错，再进一步分析原因。如果是设备端出错，请联系设备供应商协同解决问题。
- 确认错误数据是否存在规律：通过分析获取到的数据，根据错误数据出现的位置、数量以及数值，推理可能的原因。
    - 多出数据导致设备通信失败时，可以分析多出的数据是否是某个进程在特殊情况下会向这个UART发送数据。特别是在使用由console调试UART切换为普通UART时，特殊情况下的前级打印会导致此类问题。
    - 丢失数据导致设备通信失败时，可以分析是否有别的模块或进程读走了数据、UART配置是否正确等。特别是在使用DMA模式进行传输时，请根据DMA相关章节的说明，对配置进行检查。

案例

蓝牙传输异常

UART连接蓝牙模块，在高波特率（1.5M及以上）、大数据量的使用场景下，请选择带硬件流控的四线蓝牙。经过长时间的测试和论证，两线蓝牙即使在使用DMA模式传输的情况下，连接如蓝牙耳机、蓝牙音箱等产品时也会出现播放卡顿，使用体验不佳。且由于传输瓶颈主要在蓝牙模块端，主控端无法进行优化。

蓝牙传输异常的问题，请先确认蓝牙模块供应商提供的固件是否正确。特别是在系统版本升级时，需要注意同步更新蓝牙固件。主控端排查问题请参考本文其它章节，引脚电平相关、引脚复用相关等。以下几点需要注意：

- 硬件流控模式：四线蓝牙硬件流控的dts配置中，RTS引脚pinctrl的切换是在蓝牙节点做的，CTS引脚pinctrl的切换是在UART节点做的。这种处理方式主要是由休眠唤醒策略决定的，可以增加唤醒时蓝牙数据接收的稳定性。
- DMA模式：两线蓝牙配置为DMA模式，可能会出现蓝牙没有环形buffer的警告，需要自行根据使用场景配置DMA buffer。

使用RS485出现异常

Rockchip UART模块原生不支持RS485，仅支持RS232。因此，如果需要使用RS485需要外接转换模块。使用RS485出现异常时，请检查硬件电路设计以及相关电平状态。

控制台打印相关

Rockchip控制台打印使用fiq_debugger。由于fiq会关闭全局中断，因此Rockchip使用线程化打印，先将打印信息写进内存，再打印出来。配置中对应打开宏CONFIG_RK_CONSOLE_THREAD=y。相关代码和配置请参考文档*Rockchip_Developer_Guide_UART_CN*。如果正在使用的SDK版本较为老旧，请联系Rockchip FAE更新fiq_debugger相关源码。

使用控制台打印时，以下几点需要注意：

- 每一级的打印波特率需要保持一致，具体配置参考本文的关闭所有打印或切换所有打印到其他UART章节。
- 如果需要打印大量数据，在驱动代码中，增大FIFO_SIZE可以提高线程打印的缓存能力。

案例

使用控制台发送大量测试命令时出现shell卡死

此类问题通常出现在客户进行自动化压力测试时，使用控制台发送大量测试命令，shell会卡死。这是安卓shell造成的，建议更换基于busybox的shell或者bash。

回车和换行的问题

控制台打印通常会对回车字符'\r'（0x0D）和换行字符'\n'（0x0A）进行特殊处理。一般会在UART向终端输出字符时进行判断，识别到输出字符是'\n'时，在此字符前增加输出字符'\r'。在检查ASCII码形式的UART传输数据时，需要特别注意0x0D和0x0A这两个值。

休眠唤醒和功耗相关

Rockchip UART休眠唤醒的实现是在trust中，kernel中需要添加wakeup source，从而在休眠状态下让UART作为唤醒源保持运行。所有使用UART作为唤醒源时出现的问题，都需要结合trust和kernel来排查。出现此类问题，请更新SDK最新的trust固件，如果仍然无法解决，请联系负责Rockchip Trust的工程师以获取技术支持。

如果对于休眠状态下的功耗有严格要求，作为唤醒源的UART需要使用晶振作为工作时钟输入，以节省CRU小数分频器带来的功耗。

关闭所有打印或切换所有打印到其他UART

Rockchip UART打印通常包括DDR阶段、Miniloader阶段、TF-A (Trusted Firmware-A)阶段、OP-TEE阶段、Uboot阶段和Kernel阶段。具体打印信息和阶段标志请参考本文开机日志章节。关闭所有打印或切换所有打印到其他UART有以下两种做法：

- 每一级均使用关闭打印或切换打印到其他UART的特殊固件：通常Rockchip早期平台只支持这种修改方式。请联系Rockchip以获取特殊前级固件支持。
- 使用传参机制关闭打印或切换打印到其他UART：Rockchip前级打印使用传参机制，只需要修改DDR Loader的UART打印参数，即可在前级所有阶段生效。Kernel阶段需要单独进行修改。

目前，Rockchip平台主要采用使用传参机制关闭打印或切换打印到其他UART的方案。如果正在使用的SDK版本为早期版本，前级中某个阶段的UART打印功能可能没有开启传参机制，请联系Rockchip以获取技术支持。

DDR Loader修改方法

DDR Loader中关闭或切换打印，需要修改DDR Loader中的UART打印配置，修改文件rkbin/tools/ddrbin_param.txt中的以下参数：

```
uart id=          # UART控制器id，配置为0xf为关闭打印
uart iomux=       # 复用的IOMUX引脚
uart baudrate=    # 115200 or 1500000
```

详细使用说明请参考文档：ddrbin_tool_user_guide.txt。修改完成后，使用以下命令重新生成ddr.bin固件。具体ddr.bin固件名请替换为实际SDK中使用的ddr.bin固件名。

```
./ddrbin_tool ddrbin_param.txt path/to/ddr.bin
```

Uboot修改方法

Uboot中关闭打印，需要在menuconfig中，打开配置CONFIG_DISABLE_CONSOLE，保存到.config文件。

```
console  --->
    disable console in&out
```

Uboot中切换打印，由传参机制决定，不需要进行额外修改。uboot解析传参机制相关代码在arch/arm/mach-rockchip/board.c的board_init_f_init_serial()函数中。

Kernel修改方法

Kernel中关闭打印

需要在menuconfig中，关闭配置CONFIG_SERIAL_8250_CONSOLE。

```
Device Drivers  --->
    Character devices  --->
        Serial drivers  --->
            [ ] Console on 8250/16550 and compatible serial port
```

在dts配置中找到类似以下内容，并去掉UART基地址和console相关配置参数。

```
chosen: chosen {
    bootargs = "earlycon=uart8250,mmio";
}
```

找到fiq-debugger节点，修改serial-id为0xffffffff，去掉UART引脚复用相关配置。注意，需要保持fiq-debugger节点使能，保持fiq-debugger流程系统才能正常启动。

```
fiq-debugger {
    compatible = "rockchip,fiq-debugger";
    rockchip,serial-id = <0xffffffff>;
    rockchip,wake-irq = <0>;
    /* If enable uart uses irq instead of fiq */
    rockchip,irq-mode-enable = <1>;
    rockchip,baudrate = <1500000>;  /* Only 115200 and 1500000 */
    interrupts = <GIC_SPI 252 IRQ_TYPE_LEVEL_LOW>;
    status = "okay";
    };
```

Kernel中切换打印

例如将Kernel打印从UART2切换到UART3，在dts配置中找到类似以下内容，将UART基地址由UART2改为UART3.

```
chosen: chosen {
    bootargs = "earlycon=uart8250,mmio32,0xfe670000 console=ttyFIQ0";
}
```

找到fiq-debugger节点，修改serial-id为3，修改UART3引脚复用配置。注意，同时需要将切换为打印串口的UART3作为普通串口的节点禁用。

```
fiq-debugger {
    compatible = "rockchip,fiq-debugger";
    rockchip,serial-id = <3>;
    rockchip,wake-irq = <0>;
    /* If enable uart uses irq instead of fiq */
    rockchip,irq-mode-enable = <1>;
    rockchip,baudrate = <1500000>;  /* Only 115200 and 1500000 */
    interrupts = <GIC_SPI 252 IRQ_TYPE_LEVEL_LOW>;
    pinctrl-names = "default";
    pinctrl-0 = <&uart3m0_xfer>;
    status = "okay";
    };
```

将原控制台UART作为普通UART使用

进行完上述关闭或切换控制台UART后，需要重新将该UART作为普通UART使能。注意，在dts配置流程下一定是先关闭或切换控制台UART，将其释放后再作为普通UART在对应UART节点使能。

Android修改方法

安卓需要去掉recovery中对console的使用，否则恢复出厂设置的时候会卡住。请修改文件android/device/rockchip/common/recovery/etc/init.rc，注释掉console。

```
service recovery /sbin/recovery
    #console
    seclabel u:r:recovery:s0
```

案例

由打印串口切换成普通串口后出现的问题

使用由打印串口切换成的普通串口出现问题时，请先使用另一个UART进行相同的测试。如果其它UART不会出现同样的问题，请联系Rockchip获取直接在源码中关闭UART打印的特殊固件，再进行问题排查。在特殊情况下，例如DDR变频等前级操作会导致由打印串口切换成的普通串口出现数据增加或丢失。

其它问题

案例

使用某些上位机时Kernel无法正常启动

某些上位机串口调试软件，如MobaXterm、Putty等，默认开启UART软件流控，即Xon/Xoff。系统启动时，在某些情况下，如DDR变频操作，会从UART TX引脚发送大量打印数据，当这些打印数据量超过上位机设置的接收buffer时，就会触发软件流控，一直向UART RX引脚发送Xoff（对应Control+S字符）。而Rockchip平台在系统启动过程中，接收到Control+S字符时会触发Uboot Debug模式导致无法正常开机。

这些上位机默认串口配置打开软件流控是不合理的。因为软件流控并不可靠，且在Rockchip平台上并未使用软件流控。如果出现此类问题，请检查上位机串口工具的配置。

更换UART设备节点号

如果需要更换UART设备节点号，以ttyS3和ttyS4交换为例。请在dts中修改以下内容：

```
aliases {
    serial3 = &uart4;
    serial4 = &uart3;
}
```

如果是将作为控制台的UART和普通UART设备节点号交换，请直接修改drivers/tty/tty_io.c 中的tty_line_name()函数，对需要修改的编号进行判断。