

UART开发指南

发布版本：1.1

作者邮箱：hhb@rock-chips.com

日期：2019.01

文件密级：公开资料

前言

概述

产品版本

芯片名称	内核版本
全部采用linux4.4内核的RK芯片	Linux4.4

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

日期	版本	作者	修改说明
2017-12-21	V1.0	洪慧斌	初始发布
2019-02-14	V1.1	洪慧斌	更新版本

UART开发指南

1 Rockchip UART功能特点

2 内核软件

2.1 代码路径

2.2 内核配置

2.3 使能串口设备

2.3.1 使能uart0

2.3.2 驱动设备注册log

2.3.3 串口设备

2.4 DTS节点配置

2.4.1 pinctrl配置

2.4.2 关于DMA的使用

2.4.3 波特率配置说明

2.4.4 串口唤醒系统

3 Linux串口打印

- 3.1 FIQ debugger, ttyFIQ0设备作为console
 - 3.1.1 DTS使能fiq_debugger节点, 禁止对应uart节点
 - 3.1.2 使能early printk功能
 - 3.1.3 安卓 parameter.txt 配置console设备
- 3.2 ttySx设备作为console
 - 3.2.1 uart2作为console
 - 3.2.2 使能early printk功能
 - 3.2.3 安卓 parameter.txt 配置console设备
- 3.3 关掉串口打印功能
 - 3.3.1 去掉或禁止3.1或3.2的所有配置
 - 3.3.2 去掉8250驱动console的配置
 - 3.3.3 安卓去掉recovery对console的使用, 否则恢复出场设置的时候会卡住
- 4 调试串口设备

1 Rockchip UART功能特点

UART (Universal Asynchronous Receiver/Transmitter), 以下是linux 4.4 uart驱动支持的一些特性:

- 最高支持4M波特率
- 部分串口支持硬件自动流控, 部分不支持, 详细参看数据手册
- 支持中断传输模式和DMA传输模式

2 内核软件

2.1 代码路径

采用的是8250通用驱动, 类型是16550A

1	drivers/tty/serial/8250/8250_core.c	
2	drivers/tty/serial/8250/8250_dma.c	dma实现
3	drivers/tty/serial/8250/8250_dw.c	design ware ip相关操作
4	drivers/tty/serial/8250/8250_early.c	early console实现
5	drivers/tty/serial/8250/8250_fsl.c	
6	drivers/tty/serial/8250/8250.c	
7	drivers/tty/serial/8250/8250_port.c	端口相关的接口
8	drivers/tty/serial/earlycon.c	解析命令行参数, 并提供注册early con接口

2.2 内核配置

```

1 Device Drivers --->
2   Character devices --->
3     Serial drivers --->
4       [*] 8250/16550 and compatible serial support
5       [ ] Support 8250_core.* kernel options (DEPRECATED)
6       [*] Console on 8250/16550 and compatible serial port      8250串口开启
console功能
7       [ ] DMA support for 16550 compatible UART controllers
8       (5) Maximum number of 8250/16550 serial ports            一般填最大串
口数
9       (5) Number of 8250/16550 serial ports to register at runtime 一般填最大串
口数
10      [ ] Extended 8250/16550 serial driver options
11      [*] Support for Synopsys Designware 8250 quirks

```

2.3 使能串口设备

2.3.1 使能uart0

在板级DTS文件里添加以下代码：

```

1 &uart0 {
2     status = "okay";
3 };

```

2.3.2 驱动设备注册log

```

1 [0.464875] Serial: 8250/16550 driver, 5 ports, IRQ sharing disabled
2 [0.466561] ff180000.serial: ttyS0 at MMIO 0xff180000 (irq = 36, base_baud = 1500000) is
a 16550A
3 [0.467112] ff1a0000.serial: ttyS2 at MMIO 0xff1a0000 (irq = 37, base_baud = 1500000) is
a 16550A
4 [0.467702] ff370000.serial: ttyS4 at MMIO 0xff370000 (irq = 40, base_baud = 1500000) is
a 16550A

```

设备正常注册就是以上log，如果pinctrl跟其他驱动有冲突的话，会报pinctrl配置失败的log。

2.3.3 串口设备

旧的驱动起来后会先注册5个ttySx设备。但如果没有经过2.3.1使能的串口，虽然也有设备节点，但是是不能操作的。

```

1 |root@android:/ # ls /dev/tty
2 ttyS0  ttyS1  ttyS2  ttyS3  ttyS4

```

如果内核包含以下补丁，则串口驱动只会生成dts有使能的串口。

```

1 commit a997ba744c6b001b8a8033aaacc65d6f4ce849a2
2 Author: Huibin Hong <huibin.hong@rock-chips.com>
3 Date: Mon Nov 5 15:56:03 2018 +0800
4

```

```

5 serial: 8250: add /dev/ttySx when uart is enable
6
7 before the patch:
8 ls /dev/ttyS
9 ttyS0 ttyS1 ttyS2 ttyS3 ttyS4 ttyS5 ttyS6 ttyS7
10
11 after the patch:
12 ls /dev/ttyS
13 ttyS3 ttyS4 ttyS6
14
15 Change-Id: I844523408751cb579bbfb50fafb7923d5c2cafd
16 Signed-off-by: Huibin Hong <huibin.hong@rock-chips.com>

```

驱动会根据alias，来对应串口编号，如下：serial0最终会生成ttyS0，serial3会生成ttyS3设备。

```

1 aliases {
2     serial0 = &uart0;
3     serial1 = &uart1;
4     serial2 = &uart2;
5     serial3 = &uart3;
6     serial4 = &uart4;
7 };

```

2.4 DTS节点配置

以uart0 DTS节点为例：

dtsi文件里：

```

1 uart0: serial@fff180000 {
2     compatible = "rockchip,rk3399-uart", "snps,dw-apb-uart";
3     reg = <0x0 fff180000 0x0 0x100>;
4     clocks = <&cru SCLK_UART0>, <&cru PCLK_UART0>;
5     clock-names = "baudclk", "apb_pclk";
6     interrupts = <GIC_SPI 99 IRQ_TYPE_LEVEL_HIGH 0>;
7     reg-shift = <2>;
8     reg-io-width = <4>;
9     dmas = <&dmac_peri 0>, <&dmac_peri 1>;
10    dma-names = "tx", "rx";
11    pinctrl-names = "default";
12    pinctrl-0 = <&uart0xfer &uart0cts &uart0rts>;
13    status = "disabled";
14 };

```

板级dts文件添加：

```

1 &uart0 {
2     status = "okay";
3 };

```

2.4.1 pinctrl配置

有时一个串口有多组IOMUX配置，需要根据实际使用配置

```
1 | pinctrl-names = "default";
2 | pinctrl-0 = <&uart0xfer &uart0cts &uart0_rts>;
```

其中uart0_cts和uart0_rts是硬件流控脚，这只代表引脚有配置为相应的功能脚，并不代表使能硬件流控。使能硬件流控需要从运用层设置下来。需要注意的是，如果使能流控，uart0_cts和uart0_rts必须同时配上。如果不需要流控，可以把uart0_cts和uart0_rts去掉。

2.4.2 关于DMA的使用

和中断传输模式相比，使用DMA并不一定能提高传输速度，相反可能略降低传输速度。

因为现在CPU的性能都很高，传输瓶颈在外设，而且启动DMA还会消耗额外的资源。

但整体上看中断模式会占用更多的CPU资源。只有传输数据量很大时，DMA的使用

对CPU负载的减轻效果才会比较明显。

关于DMA使用的几点建议：

如果外接的设备传输数据量不大，请使用默认的中断模式。

如果外接的设备传输数据量较大，可以使用DMA。

如果串口没接自动流控脚，可以使用DMA作为FIFO缓冲，防止数据丢失

需要使用DMA时需要以下配置，如果没有需要自己手动添加：

dma-names = "tx", "rx"; 使能DMA发送和接收

dma-names = "!tx", "!rx"; 禁止DMA发送和接收

dmass = <&dmac_peri 0>, <&dmac_peri 1>; 这里的0和1是外设和DMAC连接的通道号，DMAC通过这个号来识别外设。通过手册查找Req number，如下图

Table 12-2 DMAC1 Request Mapping Table

Req number	Source	Polarity
0	UART0 tx	High level
1	UART0 rx	High level
2	UART1 tx	High level
3	UART1 rx	High level
4	UART2 tx	High level
5	UART2 rx	High level

&dmac_peri要根据手册确认外设属于哪个DMAC，来选择，一般DMAC1是dmac_peri，

DMAC0是dmac_bus。

如下：

```
1 | amba {
2 |     compatible = "arm,amba-bus";
3 |     #address-cells = <2>;
4 |     #size-cells = <2>;
```

```

5     ranges;
6
7     dmac_bus: dma-controller@ff6d0000 {
8         compatible = "arm,p1330", "arm,primecell";
9         reg = <0x0 0xff6d0000 0x0 0x4000>;
10        interrupts = <GIC_SPI 5 IRQ_TYPE_LEVEL_HIGH 0>,
11                   <GIC_SPI 6 IRQ_TYPE_LEVEL_HIGH 0>;
12        #dma-cells = <1>;
13        clocks = <&cru ACLK_DMAC0_PERILP>;
14        clock-names = "apb_pclk";
15        peripherals-req-type-burst;
16    };
17
18    dmac_peri: dma-controller@ff6e0000 {
19        compatible = "arm,p1330", "arm,primecell";
20        reg = <0x0 0xff6e0000 0x0 0x4000>;
21        interrupts = <GIC_SPI 7 IRQ_TYPE_LEVEL_HIGH 0>,
22                   <GIC_SPI 8 IRQ_TYPE_LEVEL_HIGH 0>;
23        #dma-cells = <1>;
24        clocks = <&cru ACLK_DMAC1_PERILP>;
25        clock-names = "apb_pclk";
26        peripherals-req-type-burst;
27    };
28 };

```

注意:

有些不需要使用DMA的场景，也可以考虑收发都关闭DMA，如下

```
1 dma-names = "!tx", "!rx";
```

会有以下log:

```
1 [54696.575402] ttys0 - failed to request DMA, use interrupt mode
```

由于DMA通道资源有限，在通道资源紧张的情况下，可以考虑关掉TX的DMA传输，如下

```
1 dma-names = "!tx", "rx";
```

会有以下log:

```
1 [ 498.889713] dw-apb-uart ff0a0000.serial: got rx dma channels only
```

2.4.3 波特率配置说明

波特率=时钟源/16/DIV。（DIV是分频系数）

目前的代码会根据波特率大小来设置时钟，一般1.5M以下的波特率都可以分出来。1.5M以上的波特率，可能会经过小数分频或整数分频。如果以上都分不出来，则需要修改PLL。但修改PLL有风险，会影响其他模块。可以通过redmine提需求。

如果在操作串口的时候出现以下log，需要通过打印时钟树来确定串口的时钟设置是否正确。

```
1 [54131.273012] rockchip_fractional_approximation parent_rate(676000000) is low than
2 rate(48000000)*20, fractional div is not allowed
```

注意以下命令必须在串口打开的时候打，否则clk可能不准。本次例子串口设置的是3M的波特率，从以下log可以看出，串口走的是clk_uart4_pmu 整数分频，由676M PLL分出来接近48M的的clk（48M根据上面的公式，是分出3M波特率的最小时钟）。这虽然有误差，但在允许范围内，这个误差的大小驱动里设定为正负2%。

```
1 root@android:/ # cat /sys/kernel/debug/clk/clk_summary | grep uart
2         clk_uart4_src           1           1   676000000           0 0
3         clk_uart4_div          1           1   48285715            0 0
4         clk_uart4_pmu          1           1   48285715            0 0
5         clk_uart4_frac         0           0    285257             0 0
6         pclk_uart4_pmu         1           1   48285715            0 0
```

2.4.4 串口唤醒系统

内核需要打补丁，对应的SOC的trust firmware也可能需要修改，这块需要咨询维护trust firmware的人员。

```
1 &uart0 {
2     wakeup-source;           使能串口唤醒功能，作用是待机时不去关闭串口，并把串口中断设置为唤醒源
3     status = "okay";
4 };
```

3 Linux串口打印

3.1 FIQ debugger, ttyFIQ0设备作为console

3.1.1 DTS使能fiq_debugger节点，禁止对应uart节点

```
1 fiq_debugger: fiq-debugger {
2     compatible = "rockchip,fiq-debugger";
3     rockchip,serial-id = <2>; /*设置串口id, 如果想换不同的串口就改这个ID*/
4     rockchip,wake-irq = <0>;
5     rockchip,irq-mode-enable = <0>; /* If 1, uart uses irq instead of fiq */
6     rockchip,baudrate = <1500000>; /* only 115200 and 1500000 */
7     pinctrl-names = "default";
8     pinctrl-0 = <&uart2c_xfer>; /*换了不同的串口后，需要配置iomux*/
9     interrupts = <GIC_SPI 150 IRQ_TYPE_LEVEL_HIGH 0>; /* 配置signal irq, 一般可以是
   该SOC最大中断号加1 */
10    status = "okay";
11 };
12 禁止对应uart节点
13 &uart2 {
14     status = "disabled";
15 };
```

该节点驱动加载后会注册/dev/ttyFIQ0设备，需要注意的是rockchip,serial-id 即便改了，注册的也是ttyFIQ0。

rockchip,irq-mode-enable = <0>; 这个如果为1，串口中断方式采用的是irq，一般不会遇到问题。但如果是0，用的是FIQ模式，有些带有trust firmware的平台就需要谨慎用，这可能会因为trust firmware版本和内核版本不匹配出问题。

3.1.2 使能early printk功能

添加一下参数，其中0xff1a0000是uart2的物理基地址，不同的串口基地址不一样。

一般后面参数不加115200等波特率，用uboot或loader起来后配置的波特率即可。

如果配了波特率可能会出问题，因为内核early con对这块的支持不是很好。

```
1 chosen {
2     bootargs = "earlycon=uart8250,mmio32,0xff1a0000";
3 };
```

3.1.3 安卓 parameter.txt 配置console设备

一般以下参数可以不指定，会用默认的console device，比如上面注册的ttyFIQ0。但如果指定为ttyS2的话，就不能敲命令了。

```
1 | cmdline: androidboot.console=ttyFIQ0 console=ttyFIQ0
```

3.2 ttySx设备作为console

3.2.1 uart2作为console

添加以下配置，其中0xff1a0000是uart2的物理基地址，不同的串口基地址不一样。

一般后面参数不加115200等波特率，用uboot或loader起来后配置的波特率即可。

如果配了波特率可能会出问题，因为内核early con对这块的支持不是很好。

```
1 chosen {
2     bootargs = "console=uart8250,mmio32,0xff1a0000";
3 };
4
5 &uart2 {
6     status = "okay";
7 };
```

3.2.2 使能early printk功能

```
1 | console=uart8250,mmio32,0xff1a0000 已经包含early printk的功能
```

3.2.3 安卓 parameter.txt 配置console设备

一般以下参数可以不指定，会用默认的console device，比如上面注册的ttyS2。单如果指定为ttyFIQ0的话，就不能敲命令了。


```
1 | cmdline: androidboot.console=ttyS2 console=ttyS2
```

注意：3.1和3.2不能同时存在，否则打印有问题。fiq debugger的rockchip,serial-id = ;与ttySx互斥，就是说某个串口被fiq debugger驱动用了，就不能作为普通串口用。

3.3 关掉串口打印功能

3.3.1 去掉或禁止3.1或3.2的所有配置

3.3.2 去掉8250驱动console的配置

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

如果不想修改这个配置的，需要再command line 增加console=，什么都不指定，表示不适用console。

3.3.3 安卓去掉recovery对console的使用，否则恢复出场设置的时候会卡住

```
1 | android/device/rockchip/common/recovery/etc/init.rc
2 | service recovery /sbin/recovery
3 | #console 这个注释掉
4 | seclabel u:r:recovery:s0
```

4 调试串口设备

调试串口设备最好不要用echo cat等命令来粗鲁地调试，最好用测试的APK软件，或找我司FAE获取ts_uart测试bin文件。在命令行输入ts_uart会有使用帮助。

```
1 | 1|root@android:/ # ts_uart
2 | Use the following format to run the HS-UART TEST PROGRAM
3 | ts_uart v1.0
4 | For sending data:
5 | ./ts_uart <tx_rx(s/r)> <file_name> <baudrate> <flow_control(0/1)> <max_delay(0-100)>
   | <random_size(0/1)>
6 | tx_rx : send data from file (s) or receive data (r) to put in file
7 | file_name : file name to send data from or place data in
8 | baudrate : baud rate used for TX/RX
9 | flow_control : enables (1) or disables (0) Hardware flow control using RTS/CTS lines
10 | max_delay : defines delay in seconds between each data burst when sending. Choose 0
   | for continuous stream.
11 | random_size : enables (1) or disables (0) random size data bursts when sending.
   | Choose 0 for max size.
12 | max_delay and random_size are useful for sleep/wakeup over UART testing. ONLY
   | meaningful when sending data
13 | Examples:
14 | Sending data (no delays)
15 | ts_uart s init.rc 1500000 0 0 0 /dev/ttyS0
```

```
16 | Loop back mode:
17 | ts_uart m init.rc 1500000 0 0 0 /dev/ttys0
18 | receive then send
19 | ts_uart r init.rc 1500000 0 0 0 /dev/ttys0
```

如果串口APK无法打开串口设备，那可能是权限问题，需要修改/dev/ttySx的设备权限为0666。

以安卓为例，在ueventd.rc里添加以下配置，如果还是不行请联系安卓开发人员修改权限。

```
1 | /dev/ttySx          0666   system   system
```