UART开发指南

发布版本：1.00

作者邮箱：hhb@rock-chips.com

日期：2017.12

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

---

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

```
drivers/tty/serial/8250/8250_core.c
drivers/tty/serial/8250/8250_dma.c          dma实现
drivers/tty/serial/8250/8250_dw.c           design ware ip相关操作
drivers/tty/serial/8250/8250_early.c        early console实现
drivers/tty/serial/8250/8250_fsl.c
drivers/tty/serial/8250/8250.c
drivers/tty/serial/8250/8250_port.c         端口相关的接口
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

```
Device Drivers  --->
    Character devices  --->
        Serial drivers  --->
         [*] 8250/16550 and compatible serial support
         [ ]   Support 8250_core.* kernel options (DEPRECATED)
          [*]   Console on 8250/16550 and compatible serial port        8250串口开启console功能
          [ ]   DMA support for 16550 compatible UART controllers
          (5)   Maximum number of 8250/16550 serial ports                    一般填最大串口数
          (5)   Number of 8250/16550 serial ports to register at runtime     一般填最大串口数
          [ ]   Extended 8250/16550 serial driver options
          [*] 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：

```
&uart0 {
        status = "okay";
};
```

2.3.2 驱动设备注册log

```
[0.464875] Serial: 8250/16550 driver, 5 ports, IRQ sharing disabled
[0.466561] ff180000.serial: ttyS0 at MMIO 0xff180000 (irq = 36, base_baud = 1500000) is a 16550A
[0.467112] ff1a0000.serial: ttyS2 at MMIO 0xff1a0000 (irq = 37, base_baud = 1500000) is a 16550A
[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

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/tt
ttyS0   ttyS1   ttyS2   ttyS3  ttyS4
```

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

```
    aliases {
        serial0 = &uart0;
        serial1 = &uart1;
        serial2 = &uart2;
        serial3 = &uart3;
        serial4 = &uart4;
    };
```

2.4 DTS节点配置

以uart0 DTS节点为例：

dtsi文件里：

```
uart0: serial@ff180000 {
        compatible = "rockchip,rk3399-uart", "snps,dw-apb-uart";
        reg = <0x0 0xff180000 0x0 0x100>;
        clocks = <&cru SCLK_UART0>, <&cru PCLK_UART0>;
        clock-names = "baudclk", "apb_pclk";
        interrupts = <GIC_SPI 99 IRQ_TYPE_LEVEL_HIGH 0>;
        reg-shift = <2>;
        reg-io-width = <4>;
        dmas = <&dmac_peri 0>, <&dmac_peri 1>;
        dma-names = "tx", "rx";
        pinctrl-names = "default";
        pinctrl-0 = <&uart0xfer &uart0cts &uart0_rts>;
        status = "disabled";
    };
```

板级dts文件添加：

```
&uart0 {
        status = "okay";
};
```

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

```
pinctrl-names = "default";
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

dmas = <&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：

```
amba {
compatible = "arm,amba-bus";
        #address-cells = <2>;
        #size-cells = <2>;
        ranges;

        dmac_bus: dma-controller@ff6d0000 {
            compatible = "arm,pl330", "arm,primecell";
            reg = <0x0 0xff6d0000 0x0 0x4000>;
            interrupts = <GIC_SPI 5 IRQ_TYPE_LEVEL_HIGH 0>,
                         <GIC_SPI 6 IRQ_TYPE_LEVEL_HIGH 0>;
            #dma-cells = <1>;
            clocks = <&cru ACLK_DMAC0_PERILP>;
            clock-names = "apb_pclk";
            peripherals-req-type-burst;
        };

        dmac_peri: dma-controller@ff6e0000 {
```

```
            compatible = "arm,pl330", "arm,primecell";
            reg = <0x0 0xff6e0000 0x0 0x4000>;
            interrupts = <GIC_SPI 7 IRQ_TYPE_LEVEL_HIGH 0>,
                     <GIC_SPI 8 IRQ_TYPE_LEVEL_HIGH 0>;
            #dma-cells = <1>;
            clocks = <&cru ACLK_DMAC1_PERILP>;
            clock-names = "apb_pclk";
            peripherals-req-type-burst;
        };
    };
```

注意：没开DMA的时候，在使用过程中，会报以下log，也不影响正常使用。

```
[54696.575402] ttyS0 - failed to request DMA
```

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。

```
[54131.273012] rockchip_fractional_approximation parent_rate(676000000) is low than
    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%。

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

3 Linux串口打印

3.1 FIQ debugger, ttyFIQ0设备作为console

3.1.1 使能DTS节点

```
fiq_debugger: fiq-debugger {
        compatible = "rockchip,fiq-debugger";
        rockchip,serial-id = <2>;    /*设置串口id，如果想换不同的串口就改这个ID*/
        rockchip,wake-irq = <0>;
        rockchip,irq-mode-enable = <0>;  /* If 1， uart uses irq instead of fiq */
        rockchip,baudrate = <1500000>;  /* Only 115200 and 1500000 */
        pinctrl-names = "default";
        pinctrl-0 = <&uart2c_xfer>;     /*换了不同的串口后，需要配置iomux*/
        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 */
        status = "okay";
    };
```

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 firmeware的平台就需要谨慎用，这可能会因为trust firmeware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。

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

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。

```
commandline：androidboot.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。

```
chosen {
    bootargs ="console=uart8250,mmio32,0xff1a0000";
};

&uart2 {
  status = "okay";
};
```

3.2.2 使能early printk功能

```
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。

```
commandline：androidboot.console=ttyS2
```

注意 ：3.1和3.2不能同时存在，否则打印有问题

3.3 关掉串口打印功能

3.3.1 去掉或禁止3.1和3.2的所有配置

3.3.2 去掉8250驱动console的配置

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

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

```
android/device/rockchip/common/recovery/etc/init.rc
service recovery /sbin/recovery
#console  这个注释掉
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|root@android:/ # ts_uart
 Use the following format to run the HS-UART TEST PROGRAM
```

```
 ts_uart v1.0
 For sending data:
 ./ts_uart <tx_rx(s/r)> <file_name> <baudrate> <flow_control(0/1)> <max_delay(0-100)>
<random_size(0/1)>
 tx_rx : send data from file (s) or receive data (r) to put in file
 file_name : file name to send data from or place data in
 baudrate : baud rate used for TX/RX
 flow_control : enables (1) or disables (0) Hardware flow control using RTS/CTS lines
 max_delay : defines delay in seconds between each data burst when sending. Choose 0 for
continuous stream.
 random_size : enables (1) or disables (0) random size data bursts when sending. Choose 0
for max size.
 max_delay and random_size are useful for sleep/wakeup over UART testing. ONLY meaningful
when sending data
 Examples:
 Sending data (no delays)
 ts_uart s init.rc 1500000 0 0 0 /dev/ttyS0
 loop back mode:
 ts_uart m init.rc 1500000 0 0 0 /dev/ttyS0
 receive then send
 ts_uart r init.rc 1500000 0 0 0 /dev/ttyS0
```

5 常见问题

详细见另一份文档