UART开发指南

文件标识：RK-KF-YF-088

发布版本：V1.5.0

日期：2021-12-22

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

本文主要说明Rockchip系列芯片UART的使用和调试方法。包括UART作为普通串口和控制台两种不同使用场景。

产品版本

芯片名称	内核版本
所有采用Linux 3.10内核的芯片	Linux Kernel 3.10
所有采用Linux 4.4及以上内核的芯片	Linux Kernel 4.4及以上

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	洪慧斌	2017-12-21	初始版本
V1.1.0	洪慧斌	2019-02-14	更新版本
V1.2.0	洪慧斌	2019-11-13	支持Linux4.19
V1.3.0	洪慧斌	2020-02-26	增加文档头
V1.4.0	刘诗舫	2021-03-15	更新版本
V1.5.0	刘诗舫	2021-12-22	更新版本

目录

功能特点

Rockchip UART (Universal Asynchronous Receiver/Transmitter) 基于16550A串口标准，完整模块支持以下功能：

- 支持5、6、7、8 bits数据位。
- 支持1、1.5、2 bits停止位。

- 支持奇校验和偶校验，不支持mark校验和space校验。
- 支持接收FIFO和发送FIFO，一般为32字节或者64字节。
- 支持最高4M波特率，实际支持波特率需要芯片时钟分频策略配合。
- 支持中断传输模式和DMA传输模式。
- 支持硬件自动流控，RTS+CTS。

注意，实际芯片中的UART支持的功能请以芯片手册中UART章节的描述为准，部分UART功能会进行适当裁剪。

作为普通串口

驱动路径

在Linux kernel 3.10中，使用以下驱动文件：

```
drivers/tty/serial/rk_serial.c
```

在Linux kernel 4.4和Linux kernel 4.19中，使用8250串口通用驱动，以下为主要驱动文件：

```
drivers/tty/serial/8250/8250_core.c     # 8250串口驱动核心
drivers/tty/serial/8250/8250_dw.c       # Synopsis DesignWare 8250串口驱动
drivers/tty/serial/8250/8250_dma.c      # 8250串口DMA驱动
drivers/tty/serial/8250/8250_port.c     # 8250串口端口操作
drivers/tty/serial/8250/8250_early.c    # 8250串口early console驱动
```

menuconfig配置

在不同版本的Linux kernel中，UART相关的menuconfig配置均在以下路径选项，选项说明十分详细，这里不再展开：

```
Device Drivers  --->
    Character devices  --->
        Serial drivers  --->
```

建议使用Rockchip SDK中提供的UART默认配置。

dts配置

在不同版本的Linux kernel中，UART的dts配置均与以下典型配置类似。以下典型配置以Linux kernel 4.19 RK3568芯片为例，在rk3568.dtsi中：

```
uart1: serial@fe650000 {
    compatible = "rockchip,rk3568-uart", "snps,dw-apb-uart";
    reg = <0x0 0xfe650000 0x0 0x100>;
    interrupts = <GIC_SPI 117 IRQ_TYPE_LEVEL_HIGH>;
    clocks = <&cru SCLK_UART1>, <&cru PCLK_UART1>;
    clock-names = "baudclk", "apb_pclk";
    reg-shift = <2>;
    reg-io-width = <4>;
    dmas = <&dmac0 2>, <&dmac0 3>;
    dma-names = "tx", "rx";
    pinctrl-names = "default";
    pinctrl-0 = <&uart1m0_xfer>;
    status = "disabled";
```

```
};
```

UART的板级dts配置只有以下参数允许修改：

- dma-names：
  - "tx" 打开tx dma
  - "rx" 打开rx dma
  - "!tx" 关闭tx dma
  - "!rx" 关闭rx dma
- pinctrl-0：
  - &uart1m0_xfer 配置tx和rx引脚为iomux group 0
  - &uart1m1_xfer 配置tx和rx引脚为iomux group 1
  - &uart1m0_ctsn和&uart1m0_rtsn 配置硬件自动流控cts和rts引脚为iomux group 0
  - &uart1m1_ctsn和&uart1m1_rtsn 配置硬件自动流控cts和rts引脚为iomux group 1
- status：
  - "okay" 打开
  - "disabled" 关闭

例如，打开RK3568 UART1，打开dma，配置打开了硬件自动流控的UART1的tx、rx、cts、rts的iomux为group0，在板级dts里的配置如下：

```
&uart1 {
    dma-names = "tx", "rx";
    pinctrl-names = "default";
    pinctrl-0 = <&uart1m0_xfer &uart1m0_ctsn &uart1m0_rtsn>;
    status = "okay";
};
```

需要注意，参数pinctrl-0中对于硬件自动流控的操作仅仅是配置引脚iomux，实际使能硬件自动流控的操作在UART驱动中，如果不需要使用硬件自动流控，cts和rts引脚的iomux配置可以去掉。

波特率配置

UART波特率 = 工作时钟源 / 内部分频系数 / 16。当工作时钟源由24M晶振直接提供时，UART将使用内部分频系数得到需要的波特率。当工作时钟源由CRU模块通过PLL分频提供时，UART波特率一般为工作时钟源的1/16。UART实际允许配置的波特率和此波特率下数据传输的稳定性在软件上主要由UART工作时钟分频策略决定。

目前，UART驱动会根据配置的波特率大小自动去获取需要的工作时钟频率，可以通过以下命令查询到UART工作时钟频率：

```
cat /sys/kernel/debug/clk/clk_summary | grep uart
```

Rockchip UART对常用的波特率，如115200、460800、921600、1500000、3000000、4000000等确保稳定支持。对于一些特殊的波特率，可能需要修改工作时钟分频策略才能支持。

使用DMA

UART使用DMA传输模式只有在数据量很大时才会产生较为明显的减轻CPU负载的效果。一般情况下，和使用中断传输模式相比，UART使用DMA传输模式并不一定能提高数据传输速度。一方面，现在CPU性能都很高，传输瓶颈在外设。另一方面，启动DMA需要消耗额外的资源，并且由于UART数据存在长度不确定的特性，会使DMA传输效率下降。

因此，建议一般情况下使用默认中断传输模式，会有以下打印：

```
failed to request DMA, use interrupt mode
```

在DMA通道资源紧张的使用场景下，可以考虑关掉TX的DMA传输，会有以下打印：

```
got rx dma channels only
```

使用硬件自动流控

UART使用硬件自动流控时，需要确保UART驱动使能硬件自动流控功能，且在dts中已经切换cts和rts流控引脚的iomux。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，即使用四线UART。

使用串口唤醒系统

串口唤醒系统功能是在系统待机时串口保持打开，并且把串口中断设置为唤醒源。使用时需要在dts中增加以下参数：

```
&uart1 {
    wakeup-source;
};
```

注意，串口唤醒系统需要同时修改trust固件，请联系Rockchip以获取支持。

设备注册

在dts中使能UART后，能在系统启动的log中看到以下对应的打印，表示设备正常注册：

```
fe650000.serial: ttyS1 at MMIO 0xfe650000 (irq = 67, base_baud = 1500000) is a
16550A
```

普通串口设备将会根据dts中的aliase来对串口进行编号，对应注册成ttySx设备。dts中的aliases如下：

```
aliases {
    serial0 = &uart0;
    serial1 = &uart1;
    serial2 = &uart2;
    serial3 = &uart3;
    ......
```

如果需要把uart3注册成ttyS1，可以进行以下修改：

```
aliases {
    serial0 = &uart0;
    serial1 = &uart3;
    serial2 = &uart2;
    serial3 = &uart1;
    ......
```

作为控制台

驱动路径

Rockchip UART作为控制台，使用fiq_debugger流程。Rockchip SDK一般会将uart2配置为ttyFIQ0设备。使用以下驱动文件：

```
drivers/staging/android/fiq_debugger/fiq_debugger.c # 驱动文件
drivers/soc/rockchip/rk_fiq_debugger.c              # kernel 4.4及之后的平台实现
arch/arm/mach-rockchip/rk_fiq_debugger.c            # kernel 3.10平台实现
```

menuconfig配置

在不同版本的Linux kernel中，fiq_debugger相关的menuconfig配置均在以下路径选项：

```
Device Drivers  --->
    [*] Staging drivers  --->
        Android  --->
```

建议使用Rockchip SDK默认配置。

dts配置

以Linux kernel 4.19 RK3568为例，在dts中fiq_debugger节点配置如下。由于fiq_debugger和普通串口互斥，在使能fiq_debugger节点后必须禁用对应的普通串口uart节点。

```
chosen: chosen {
    bootargs = "earlycon=uart8250,mmio32,0xfe660000 console=ttyFIQ0";
};

fiq-debugger {
    compatible = "rockchip,fiq-debugger";
    rockchip,serial-id = <2>;
    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 = <&uart2m0_xfer>;
    status = "okay";
};

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

以下对几个参数进行说明：

- rockchip,serial-id：使用的UART编号。修改serial-id到不同UART，fiq_debugger设备也会注册成ttyFIQ0设备。
- rockchip,irq-mode-enable：配置为1使用irq中断，配置为0使用fiq中断。
- interrupts：配置的辅助中断，保持默认即可。

parameter.txt配置

如果使用Linux kernel 3.10和Linux kernel 4.4，需要确保在parameter.txt文件中有以下对于控制台的指定命令：

```
CMDLINE: console=ttyFIQ0 androidboot.console=ttyFIQ0
```

驱动调试

Rockchip UART调试提供一个测试程序ts_uart.uart、两个测试用文件send_0x55和send_00_ff，该程序可以向Rockchip FAE获取。

通过adb工具将测试程序放在开发板上一个可执行的路径下，以下放在data路径：

```
adb root
adb remount
adb push ts_uart.uart /data
adb push send_0x55 /data
adb push send_00_ff /data
```

在开发板上修改测试程序权限：

```
su
chmod +x /data/ts_uart.uart
```

使用以下命令可以获取程序帮助：

```
console:/ # ./data/ts_uart.uart

 Use the following format to run the HS-UART TEST PROGRAM
 ts_uart v1.1
 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, data must be 0x55
 ts_uart r init.rc 1500000 0 0 0 /dev/ttyS0
```

测试发送

测试发送的命令如下，send_0x55和send_00_ff为发送的文件：

```
./data/ts_uart.uart s ./data/send_0x55 1500000 0 0 0 /dev/ttyS1
./data/ts_uart.uart s ./data/send_00_ff 1500000 0 0 0 /dev/ttyS1
```

发送成功可以通过USB转UART小板连接PC端，使用PC端串口调试工具验证。

测试接收

测试接收的命令如下，receive_0x55为接收的文件：

```
./data/ts_uart.uart r ./data/receive_0x55 1500000 0 0 0 /dev/ttyS1
```

可以使用PC端串口调试工具发送数据，测试程序将自动检测，检测到U（0x55）接收正确，检测到其它字符将打印16进制ASCII码值，可以对照查询接收是否正确。

测试内部自发自收

测试内部自发自收的命令如下：

```
./data/ts_uart.uart m ./data/send_00_ff 1500000 0 0 0 /dev/ttyS1
```

按下Ctrl+C停止测试，可以观察到结束log如下。比较发送和接收的数据是否一致：

```
Sending data from file to port...
send:1172, receive:1172 total:1172 # 收发数据一致，测试成功
send:3441, receive:3537 total:3441 # 收发数据不一致，测试失败
```

如果测试失败，说明当前串口存在问题或者有其他程序也在使用同一个串口。可以使用以下命令查看哪些程序打开了这个串口：

```
lsof | grep ttyS1
```

测试流控

验证CTS，先手动拉高CTS引脚电平，再使用以下命令发送数据：

```
./data/ts_uart.uart s ./data/send_0x55 1500000 1 0 0 /dev/ttyS1
```

当CTS电平被拉高时，发送数据阻塞。当释放CTS电平为低电平时，被阻塞的数据完成发送。

验证RTS，通过测量RTS引脚电平是否能够正常拉高和拉低来确认。