UART Development Guide

ID: RK-KF-YF-089

Release Version: V1.4.0

Release Date: 2021-03-22

Security Level: □Top-Secret □Secret □Internal ■Public

Rockchip Electronics Co., Ltd.

No.18 Building, A District, No.89, software Boulevard Fuzhou, Fujian,PRC

Website: www.rock-chips.com

Customer service Tel: +86-4007-700-590

Customer service Fax: +86-591-83951833

Customer service e-Mail: fae@rock-chips.com

Preface

Overview

This article mainly explains the use and debugging method of Rockchip series chip UART. Including UART as a common serial port and console in two different usage scenarios.

Product Version

Chipset	Kernel Version
All chips using Linux Kernel 3.10	Linux Kernel 3.10
All chips using Linux Kernel 4.4	Linux Kernel 4.4
All chips using Linux Kernel 4.19	Linux Kernel 4.19

Intended Audience

This document (this guide) is mainly intended for:

Technical support engineers

Software development engineers

Revision History

Version	Author	Date	Change Description
V1.0.0	Huibin Hong	2017-12-21	Initial version
V1.1.0	Huibin Hong	2019-02-14	Updated version
V1.2.0	Huibin Hong	2019-11-13	Support Linux Kernel 4.19
V1.3.0	Huibin Hong	2020-02-26	Add the document header
V1.4.0	Steven Liu	2021-03-15	Updated version

Contents

1. Features

Rockchip UART (Universal Asynchronous Receiver/Transmitter) is based on the 16550A serial port standard. The complete module supports the following functions:

- Supports 5, 6, 7, 8 bits of data.
- Support 1, 1.5, 2 bits stop bits.
- Support odd check and even check, mark check and space check are not supported.
- Support receiving FIFO and sending FIFO, generally 32 bytes or 64 bytes.
- Supports up to 4M baud rate, the actual support of baud rate requires the cooperation of chip clock frequency division strategy.
- Support interrupt transfer mode and DMA transfer mode.
- Support hardware automatic flow control, RTS+CTS.

Note that the features supported by the UART in the actual chip are subject to the description in the UART chapter of the chip manual, and some UART features will be appropriately tailored.

2. As a normal serial port

2.1 Driver path

In Linux kernel 3.10, the following driver files are used:

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

In Linux kernel 4.4 and Linux kernel 4.19, the 8250 serial port universal driver is used. The following are the main driver files:

```
drivers/tty/serial/8250/8250_core.c     # 8250 serial driver core
drivers/tty/serial/8250/8250_dw.c       # Synopsis DesignWare 8250 serial driver
drivers/tty/serial/8250/8250_dma.c      # 8250 serial DMA driver
drivers/tty/serial/8250/8250_port.c     # 8250 serial port operation
drivers/tty/serial/8250/8250_early.c    # 8250 serial early console driver
```

2.2 menuconfig configuration

In different versions of Linux kernel, the UART-related menuconfig configuration is in the following path options. The description of the options is very detailed and will not be expanded here:

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

It is recommended to use the UART default configuration provided in Rockchip SDK.

2.3 dts configuration

In different versions of Linux kernel, the dts configuration of UART is similar to the following typical configuration. The following typical configuration takes Linux kernel 4.19 RK3568 chip as an example, in rk3568.dtsi:

```
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";
};
```

Only the following parameters of the board-level dts configuration of UART are allowed to be modified:

- dma-names：
  - "tx" Enable tx dma
  - "rx" Enable rx dma
  - "!tx" Disable tx dma
  - "!rx" Disable rx dma
- pinctrl-0：
  - &uart1m0_xfer Configure tx and rx pins as iomux group 0
  - &uart1m1_xfer Configure tx and rx pins as iomux group 1
  - &uart1m0_ctsn and &uart1m0_rtsn Configure hardware automatic flow control cts and rts pins as iomux group 0
  - &uart1m1_ctsn and &uart1m1_rtsn Configure hardware automatic flow control cts and rts pins as iomux group 1
- status：
  - "okay" Enable
  - "disabled" Disable

For example, turn on RK3568 UART1, turn on dma, configure the tx, rx, cts, rts iomux of UART1 with hardware automatic flow control turned on as group0, the configuration in the board-level dts is as follows:

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

It should be noted that the operation of the hardware automatic flow control in the parameter pinctrl-0 is only the configuration of rts and cts pins iomux. The actual operation of enabling the hardware automatic flow control is in the UART driver. If you do not need to use the hardware automatic flow control, the configuration of rts and cts pins iomux can be removed.

2.4 Baud rate configuration

UART baud rate = working clock source / internal frequency division coefficient / 16. When the working clock source is directly provided by the 24M crystal oscillator, the UART will use the internal frequency coefficient to obtain the required baud rate. When the working clock source is provided by the CRU module through PLL frequency division, the UART baud rate is generally 1/16 of the working clock source. The baud rate that UART actually allows to configure and the stability of data transmission under this baud rate are mainly determined by the UART working clock division strategy in software.

At present, the UART driver will automatically obtain the required working clock frequency according to the configured baud rate. The UART working clock frequency can be queried by the following command:

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

Rockchip UART ensures stable support for commonly used baud rates such as 115200, 460800, 921600, 1500000, 3000000, 4000000, etc. For some special baud rates, it may be necessary to modify the working clock frequency division strategy to support it.

2.5 Use DMA

UART uses the DMA transfer mode to produce a more obvious effect of reducing the load on the CPU only when the amount of data is large. Under normal circumstances, compared with using interrupt transfer mode, UART using DMA transfer mode does not necessarily increase the data transmission speed. On the one hand, CPU performance is very high now, and the transmission bottleneck lies in the peripherals. On the other hand, starting DMA needs to consume extra resources, and because the UART data has the characteristic of uncertain length, it will reduce the DMA transmission efficiency.

Therefore, it is recommended to use the default interrupt transmission mode in general, and the following prints will be made:

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

In scenarios where DMA channel resources are tight, you can consider turning off TX DMA transmission, and the following prints will appear:

```
got rx dma channels only
```

2.6 Use hardware automatic flow control

When the UART uses hardware automatic flow control, you need to ensure that the UART driver enables the hardware automatic flow control function, and the iomux of the cts and rts flow control pins has been switched in the dts. It is recommended to use hardware automatic flow control in high baud rate (1.5M baud rate and above) and large data volume scenarios, that is, use four-wire UART.

2.7 Use serial port to wake up the system

The serial port wake-up function is to keep the serial port open when the system is in standby, and set the serial port interrupt as the wake-up source. When using, you need to add the following parameters in dts:

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

Note that the serial port wake-up system needs to modify the trust firmware at the same time, please contact Rockchip for support.

2.8 Device registration

After enabling UART in dts, you can see the following corresponding print in the system startup log, indicating that the device is registered normally:

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

Ordinary serial devices will number the serial ports according to the aliase in dts and register them as ttySx devices. The aliases in dts are as follows:

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

If you need to register uart3 as ttyS1, you can make the following modifications:

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

3. As a console

3.1 Driver path

Rockchip UART is used as the console and uses the fiq_debugger process. Rockchip SDK generally configures uart2 as a ttyFIQ0 device. Use the following driver files:

```
drivers/staging/android/fiq_debugger/fiq_debugger.c # Driver files
drivers/soc/rockchip/rk_fiq_debugger.c              # Platform implementation of
kernel 4.4 and later
arch/arm/mach-rockchip/rk_fiq_debugger.c            # Platform implementation of
kernel 3.10
```

3.2 menuconfig configuration

In different versions of Linux kernel, the menuconfig configuration related to fiq_debugger is in the following path options:

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

It is recommended to use the default configuration of Rockchip SDK.

3.3 dts configuration

Taking Linux kernel 4.19 RK3568 as an example, the fiq_debugger node configuration in dts is as follows. Since fiq_debugger and ordinary serial ports are mutually exclusive, the corresponding ordinary serial port uart node must be disabled after the fiq_debugger node is enabled.

```
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";
};
```

Several parameters are described below:

- rockchip, serial-id: UART number used. Modify the serial-id to a different UART, the fiq_debugger device will also be registered as a ttyFIQ0 device.
- rockchip, irq-mode-enable: Configure to 1 to use irq interrupt, and to 0 to use fiq interrupt.
- interrupts: Auxiliary interrupts configured, just keep the default.

3.4 parameter.txt configuration

If you use Linux kernel 3.10 and Linux kernel 4.4, you need to make sure that there are the following specific commands for the console in the parameter.txt file:

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

4. Driver debugging

Rockchip UART debugging provides a test program ts_uart.uart, two test files send_0x55 and send_00_ff, which can be obtained from Rockchip FAE.

Place the test program in an executable path on the development board through the adb tool. Place the following in the data path for example:

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

Modify the test program permissions on the development board:

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

Use the following command to get program help:

```
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
```

4.1 Test sending data

The commands sent in the test are as follows, send_0x55 and send_00_ff are the files to be sent:

```
./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
```

The successful transmission can be connected to the PC through the USB to UART board, and the serial port debugging tool on the PC can be used to verify.

4.2 Test receiving data

The commands received in the test are as follows, receive_0x55 is the received file:

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

You can use the PC-side serial port debugging tool to send data. The test program will automatically detect that U (0x55) is received correctly. If other characters are detected, the hexadecimal ASCII code value will be printed. You can check whether the reception is correct.

4.3 Test internal loopback

The commands loopback in the test are as follows:

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

Press Ctrl+C to stop the test, you can observe the end log as follows. Compare whether the sent and received data are consistent:

```
Sending data from file to port...
send:1172, receive:1172 total:1172 # Consistent data sent and received, the test
succeeded
send:3441, receive:3537 total:3441 # Inconsistent data sent and received, the
test failed
```

If the test fails, it means that there is a problem with the current serial port or other programs are using the same serial port at the same time. You can use the following command to check that these programs have opened the serial port:

```
lsof | grep ttyS1
```

4.4 Test flow control

To test CTS, first manually pull up the CTS pin level, and then use the following commands to send data:

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

When the CTS level is pulled high, sending data is blocked. When the CTS level is released to a low level, the blocked data is sent.

To test RTS, confirm by measuring whether the RTS pin level can be pulled up and down normally.