D1-H Linux UART
开发指南

版本号: 1.1
发布日期: 2021.5.21

版本历史

版本号	日期	制/修订人	内容描述
1.0	2021.4.23	XAA0191	添加初版
1.1	2021.5.21	XAA0191	删除与本平台无关的描述

目　录

1 概述 **1**
1.1 编写目的 . . . 1
1.2 适用范围 . . . 1
1.3 相关人员 . . . 1

2 模块介绍 **2**
2.1 模块功能介绍 . . . 2
2.2 相关术语介绍 . . . 2
2.3 源码结构介绍 . . . 3

3 模块配置介绍 **4**
3.1 kernel menuconfig 配置 . . . 4
3.2 device tree 源码结构和路径 . . . 6
3.2.1 device tree 对 uart 控制器的通用配置 . . . 6
3.2.2 board.dts 板级配置 . . . 7
3.2.3 uart dma 模式配置 . . . 7
3.2.4 设置其他 uart 为打印 conole . . . 8

4 接口描述 **10**
4.1 打开/关闭串口 . . . 10
4.2 读/写串口 . . . 10
4.3 设置串口属性 . . . 10
4.3.1 tcgetattr . . . 11
4.3.2 tcsetattr . . . 11
4.3.3 cfgetispeed . . . 12
4.3.4 cfgetospeed . . . 12
4.3.5 cfsetispeed . . . 12
4.3.6 cfsetospeed . . . 13
4.3.7 cfsetspeed . . . 13
4.3.8 tcflush . . . 13

5 模块使用范例 **14**

6 FAQ **19**
6.1 UART 调试打印开关 . . . 19
6.1.1 通过 debugfs 使用命令打开调试开关 . . . 19
6.1.2 sysfs 调试接口 . . . 19

插　图

2-1 Linux UART 体系结构图 2
3-1 内核 menuconfig 根菜单 4
3-2 内核 menuconfig device drivers 菜单 5
3-3 内核 menuconfig Character drivers 菜单 5
3-4 内核 menuconfig sunxi uart 配置菜单 6
3-5 内核 menuconfig sunxi uart 配置菜单 8

1 概述

1.1 编写目的

介绍 Linux 内核中 UART 驱动的接口及使用方法，为 UART 设备的使用者提供参考。

1.2 适用范围

表 1-1: 适用产品列表

产品名称	内核版本	驱动文件
D1-H	Linux-5.4	sunxi-uart.c

1.3 相关人员

UART 驱动、及应用层的开发/维护人员。

2 模块介绍

2.1 模块功能介绍

Linux 内核中,UART 驱动的结构图 2-1 所示, 可以分为三个层次:

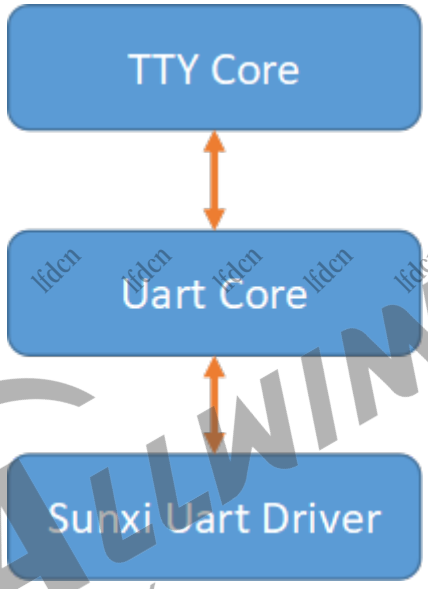

图 2-1: Linux UART 体系结构图

1. Sunxi UART Driver, 负责 SUNXI 平台 UART 控制器的初始化、数据通信等, 也是 Allwinner 实现的部分。
2. UART Core, 为 UART 驱动提供了一套 API, 完成设备和驱动的注册等。
3. TTY core, 实现了内核中所有 TTY 设备的注册和管理。

2.2 相关术语介绍

表 2-1: UART 模块相关术语介绍

术语	解释说明
Sunxi	指 Allwinner 的一系列 SoC 硬件平台
UART	Universal Asynchronous Receiver/Transmitter，通用异步收发传输器
Console	控制台，Linux 内核中用于输出调试信息的 TTY 设备

术语	解释说明
TTY	TeleType/TeleTypewriters 的一个老缩写，原来指的是电传打字机，现在泛指和计算机串行端口连接的终端设备。TTY 设备还包括虚拟控制台，串口以及伪终端设备

2.3 源码结构介绍

```
linux5.4
|-- drivers
|    |-- tty
|    |    |-- serial
|    |          |-- serial_core.c
|    |          |-- sunxi-uart.c
|    |          |-- sunxi-uart.h
```

3 模块配置介绍

3.1 kernel menuconfig 配置

在 longan 顶层目录，执行./build.sh menuconfig(需要先执行./build.sh config) 进入配置主界面，并按以下步骤操作：首先，选择 Device Drivers 选项进入下一级配置，如下图所示：

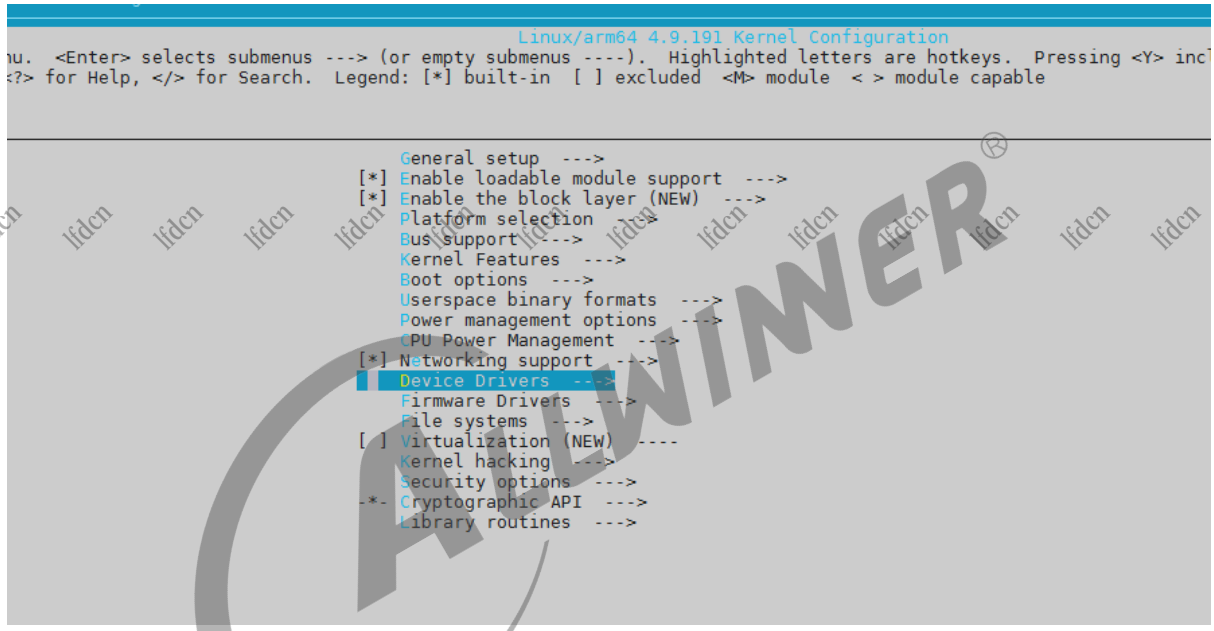

图 3-1: 内核 menuconfig 根菜单

选择 Character devices, 进入下级配置，如下图所示：

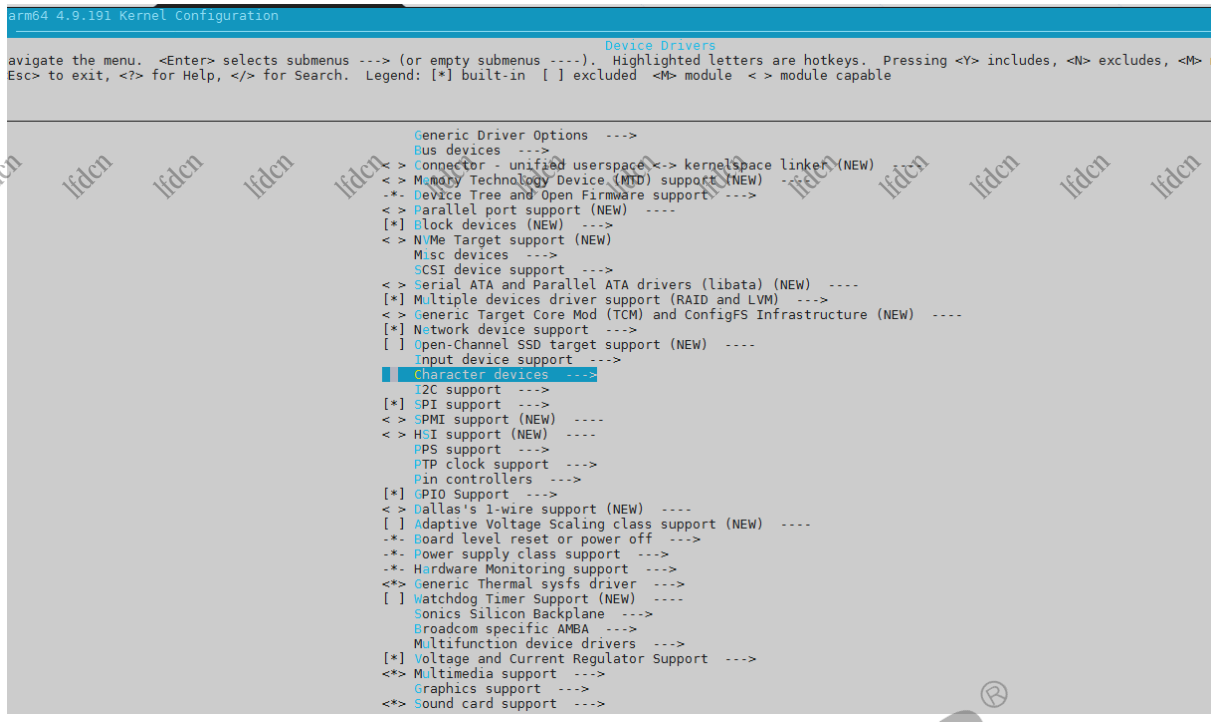

图 3-2: 内核 menuconfig device drivers 菜单

选择 Serial drivers, 进入下级配置，如下图所示：

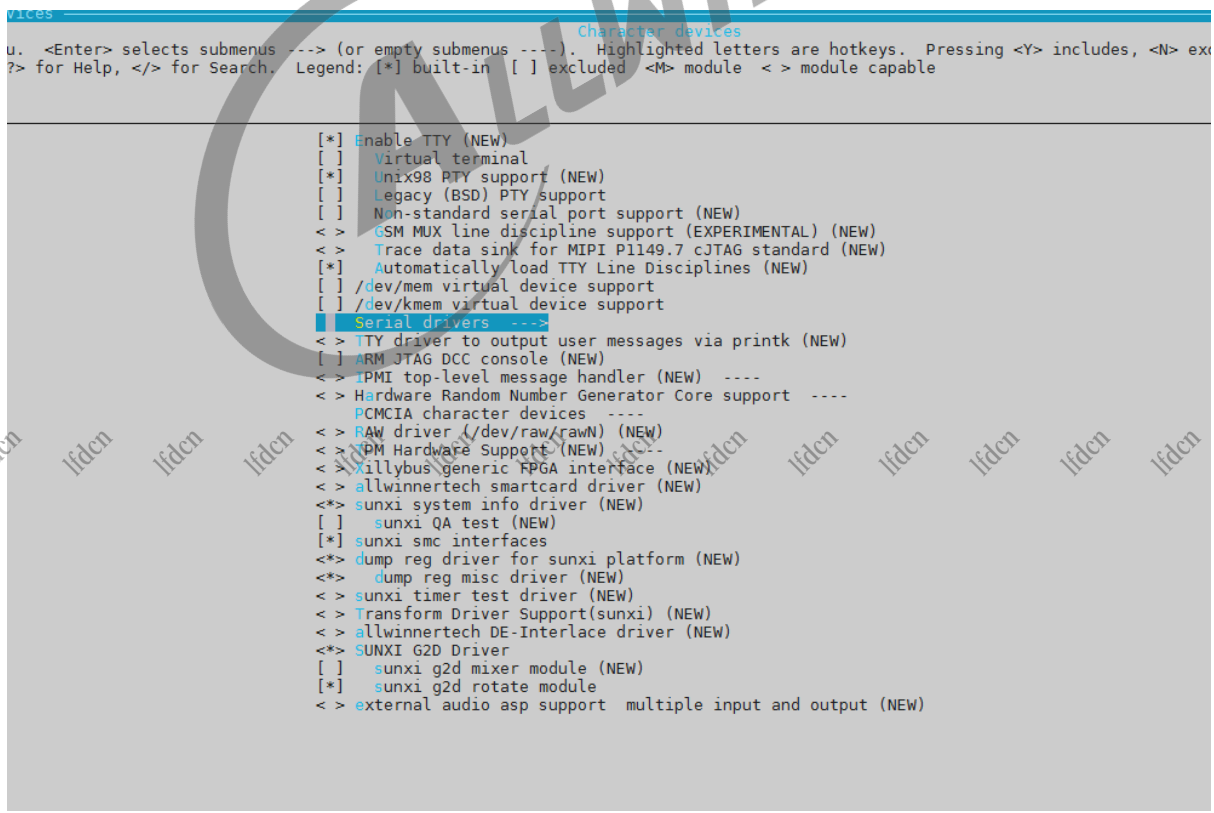

图 3-3: 内核 menuconfig Character drivers 菜单

选择 SUNXI UART Controller 和 Console on SUNXI UART port 选项，如下图所示：

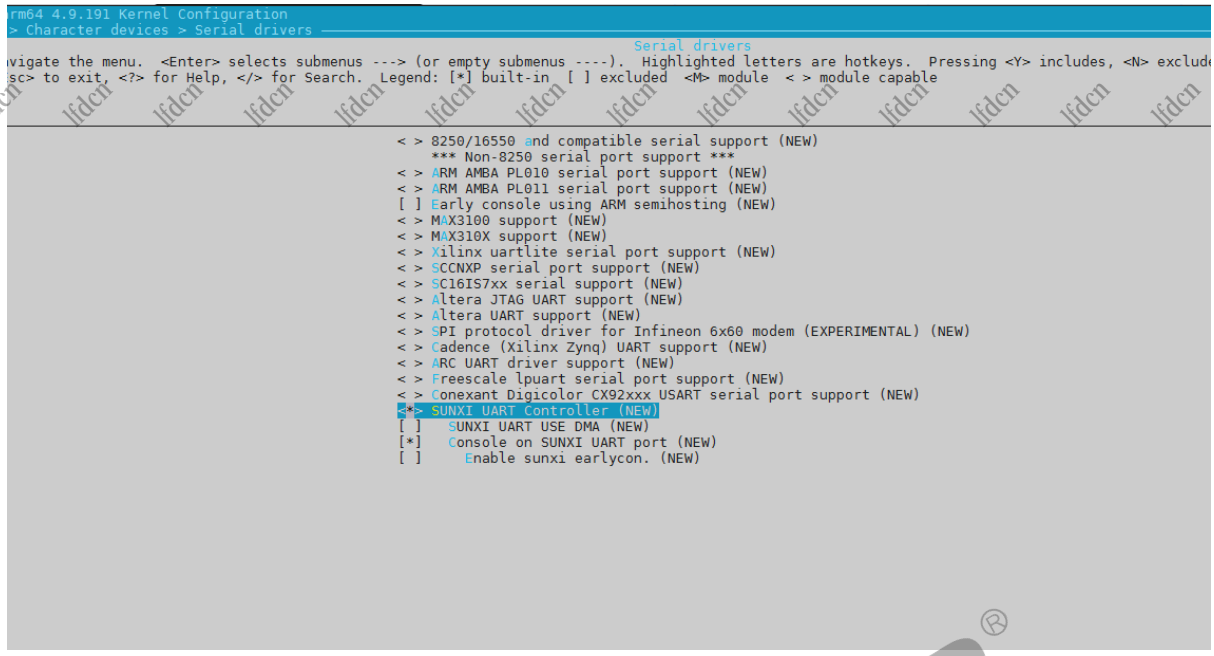

图 3-4: 内核 menuconfig sunxi uart 配置菜单

如果需要 UART 支持 DMA 传输，则可以打开 SUNXI UART USE DMA 选项。

3.2 device tree 源码结构和路径

- 设备树文件的配置是该 SoC 所有方案的通用配置，对于 RISCV 而言，设备树的路径为内核目录下：kernel/linux5.4/arch/riscv/boot/dts/sunxi/sun20iw1p1.dtsi
- 板级设备树 (board.dts) 路径：/device/config/chips/d1-h/configs/nezha/board.dts

device tree 的源码结构关系如下：

```
board.dts
  |-------sun20iw1p1.dtsi
```

3.2.1 device tree 对 uart 控制器的通用配置

linux-5.4 的通用配置如下：

```
1     uart0: uart@5000000 {
2         compatible = "allwinner,sun50i-uart";
3         device_type = "uart0";
4         reg = <0x0 0x05000000 0x0 0x400>;
5         interrupts = <GIC_SPI 0 IRQ_TYPE_LEVEL_HIGH>;
6         sunxi,uart-fifosize = <64>;
```

```
        clocks = <&ccu CLK_BUS_UART0>;  /* 设备使用的时钟 */
        clock-names = "uart0";
        resets = <&ccu RST_BUS_UART0>;  /* 设备reset时钟 */
        pinctrl-names = "default", "sleep";
        pinctrl-0 = <&uart0_pins_a>;
        pinctrl-1 = <&uart0_pins_b>;
        uart0_port = <0>;
        uart0_type = <2>;
        dmas = <&dma 14>;       /* 14表示DRQ */
        dma-names = "tx";
        use_dma = <0>;  /* 是否采用DMA 方式传输，0：不启用，1：只启用TX，2：只启用RX，3：启用TX 与RX */
    };
```

在 Device Tree 中对每一个 UART 控制器进行配置，一个 UART 控制器对应一个 UART 节点，节点属性的含义见注释。为了在 UART 驱动代码中区分每一个 UART 控制器，需要在 Device Tree 中的 aliases 节点中未每一个 UART 节点指定别名，如上 aliases 节点所示。别名形式为字符串 “serial” 加连续编号的数字，在 UART 驱动程序中可以通过 of_alias_get_id() 函数获取对应的 UART 控制器的数字编号，从而区分每一个 UART 控制器。

3.2.2 board.dts 板级配置

board.dts 用于保存每个板级平台的设备信息，board.dts 路径如下：

/device/config/chips/d1-h/configs/nezha/board.dts

在 board.dts 中的配置信息如果在 sun20iw1p1.dtsi 存在，则会存在以下覆盖规则：

1. 相同属性和结点，board.dts 的配置信息会覆盖 sun20iw1p1.dtsi 中的配置信息
2. board.dts 里新增加的属性和结点，会追加到最终生成的 dtb 文件中

uart 在 board.dts 的简单配置如下：

```
soc@02500000 {
    ...
    &uart0 {
        status = "okay";
    };
    &uart1 {
        status = "okay";
    };
    ...
}
```

3.2.3 uart dma 模式配置

1. 在内核配置菜单打开 CONFIG_SERIAL_SUNXI_DMA 配置，如下图所示:

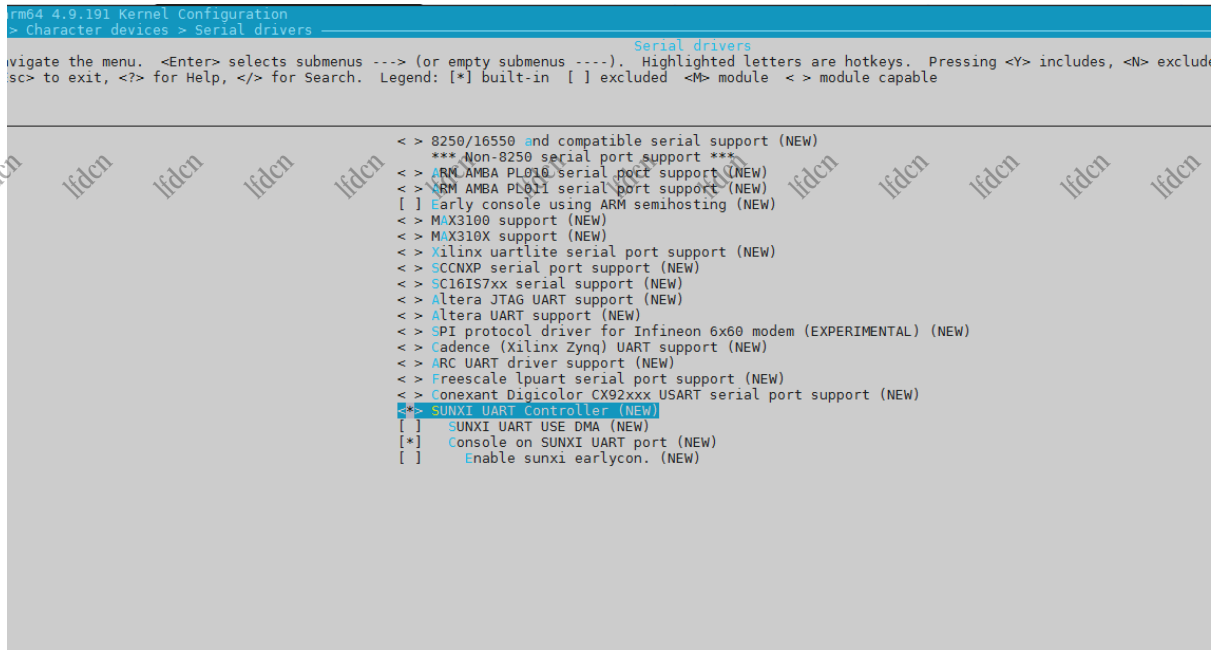

图 3-5: 内核 menuconfig sunxi uart 配置菜单

2. 在对应 dts 配置使用 dma，如下所示:

linux-5.4 配置如下:

```
    uart0: uart@2500000 {
        ...
        dmas = <&dma 14>;    /* 14表示DRQ, 请参考DMA相关文档 */
        dma-names = "tx";
        use_dma = <0>;  /* 是否采用DMA 方式传输，0：不启用，1：只启用TX，2：只启用RX，3：启用TX 与RX */
    };
```

3.2.4 设置其他 uart 为打印 conole

按照以下两个步骤进行设置:

步骤一. 从 board.dts 中查看对应的 uart 口（想要作为新 console 的 uart 口）的配置，确认status配置为okay

```
&uart1 {
    pinctrl-names = "default", "sleep";
    pinctrl-0 = <&uart1_pins_a>;
    pinctrl-1 = <&uart1_pins_b>;
    status = "okay"; /* 确保该uart已经使能 */
};
```

步骤二. 修改方案使用的 env.cfg 文件，如下所示:

```
console=ttyS1,115200

说明:
ttyS0 <===> uart0
ttyS1 <===> uart1
...
```

4 接口描述

UART 驱动会注册生成串口设备/dev/ttySx，应用层的使用只需遵循 Linux 系统中的标准串口编程方法即可。

4.1 打开/关闭串口

需要引用头文件：

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
```

使用标准的文件打开函数：

```
int open(const char *pathname, int flags);
int close(int fd);
```

4.2 读/写串口

需要引用头文件：

```
#include <unistd.h>
```

同样使用标准的文件读写函数：

```
ssize_t read(int fd, void *buf, size_t count);
ssize_t write(int fd, const void *buf, size_t count);
```

4.3 设置串口属性

需要引用头文件：

```
terminos.h
```

串口属性包括波特率、数据位、停止位、校验位、流控等，这部分是串口设备特有的接口。串口属性的数据结构 termios 定义如下：

```
#define NCCS 19
struct termios {
    tcflag_t c_iflag;       /* input mode flags */
    tcflag_t c_oflag;       /* output mode flags */
    tcflag_t c_cflag;       /* control mode flags */
    tcflag_t c_lflag;       /* local mode flags */
    cc_t c_line;            /* line discipline */
    cc_t c_cc[NCCS];        /* control characters */
};
```

其中，termios 结构体中各个标志位的使用说明见 linux 官方帮助文档（用户可以通过man指令查询）：

```
man tcsetattr
```

串口属性相关的控制接口如下所示：

4.3.1 tcgetattr

- 函数原型：`int tcgetattr(int fd, struct termios *termios_p);`
- 作用：获取串口设备的属性。
- 参数：
  - fd，串口设备的文件描述符。
  - termios_p，用于保存串口属性。
- 返回：
  - 成功，返回 0。
  - 失败，返回-1，给出具体错误码。

4.3.2 tcsetattr

- 函数原型：`int tcsetattr(int fd, int optional_actions, const struct termios *termios_p);`
- 作用：设置串口设备的属性。
- 参数：
  - fd，串口设备的文件描述符。
  - optional_actions，本次设置什么时候生效。
  - termios_p，指向要设置的属性结构。
- 返回：
  - 成功，返回 0。
  - 失败，返回-1，errno 给出具体错误码

说明

其中，*optional_actions* 的取值有：
***TCSANOW*：会立即生效。**
***TCSADRAIN*：当前的输出数据完成传输后生效，适用于修改了输出相关的参数。**
***TCSAFLUSH*：当前的输出数据完成传输，如果输入有数据可读但没有读就会被丢弃。**

4.3.3 cfgetispeed

- 作用：返回串口属性中的输入波特率。
- 参数：
  - termios_p，指向保存有串口属性的结构。
- 返回：
  - 成功，返回波特率，取值是一组宏，定义在 terminos.h。
  - 失败，返回-1，errno 给出具体错误码。

4.3.4 cfgetospeed

- 作用：返回串口属性中的输出波特率。
- 参数：
  - termios_p，指向保存有串口属性的结构。
- 返回：
  - 成功，返回波特率，取值是一组宏，定义在 terminos.h，见 4.3.3
  - 失败，返回-1，errno 给出具体错误码。

4.3.5 cfsetispeed

- 作用：设置输入波特率到属性结构中。
- 参数：
  - termios_p，指向保存有串口属性的结构。
  - speed，波特率，取值同 4.3.3。
- 返回：
  - 成功，返回 0。
  - 失败，返回-1，errno 给出具体错误码。

4.3.6 cfsetospeed

- 作用：设置输出波特率到属性结构中。
- 参数：
  - termios_p，指向保存有串口属性的结构。
  - speed，波特率，取值同 4.3.3。
- 返回：
  - 成功，返回 0。
  - 失败，返回-1，errno 给出具体错误码

4.3.7 cfsetspeed

- 作用：同时设置输入和输出波特率到属性结构中。
- 参数：
  - termios_p，指向保存有串口属性的结构。
  - speed，波特率，取值同 4.3.3。
- 返回：
  - 成功，返回 0。
  - 失败，返回-1，errno 给出具体错误码

4.3.8 tcflush

- 作用：清空输出缓冲区、或输入缓冲区的数据，具体取决于参数 queue_selector。
- 参数：
  - fd，串口设备的文件描述符。
  - queue_selector，清空数据的操作。
- 返回：
  - 成功，返回 0。
  - 失败，返回-1，errno 给出具体错误码。

说明

参数 *queue_selector* 的取值有三个：
***TCIFLUSH*：清空输入缓冲区的数据。**
***TCOFLUSH*：清空输出缓冲区的数据。**
***TCIOFLUSH*：同时清空输入/输出缓冲区的数据。**

5 模块使用范例

此 demo 程序是打开一个串口设备，然后侦听这个设备，如果有数据可读就读出来并打印。设备名称、侦听的循环次数都可以由参数指定

```
#include <stdio.h>      /*标准输入输出定义*/
#include <stdlib.h>     /*标准函数库定义*/
#include <unistd.h>     /*Unix 标准函数定义*/
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>      /*文件控制定义*/
#include <termios.h>    /*PPSIX 终端控制定义*/
#include <errno.h>      /*错误号定义*/
#include <string.h>

enum parameter_type {
    PT_PROGRAM_NAME = 0,
    PT_DEV_NAME,
    PT_CYCLE,

    PT_NUM
};

#define DBG(string, args...) \
    do { \
        printf("%s, %s()%u---", __FILE__, __FUNCTION__, __LINE__); \
        printf(string, ##args); \
        printf("\n"); \
    } while (0)

void usage(void)
{
    printf("You should input as: \n");
    printf("\t select_test [/dev/name] [Cycle Cnt]\n");
}

int OpenDev(char *name)
{
    int fd = open(name, O_RDWR );         //| O_NOCTTY | O_NDELAY
    if (-1 == fd)
        DBG("Can't Open(%s)!", name);

    return fd;
}

/**
*@brief  设置串口通信速率
*@param  fd     类型 int  打开串口的文件句柄
*@param  speed  类型 int  串口速度
*@return  void
*/
void set_speed(int fd, int speed){
```

```
    int   i;
    int   status;
    struct termios  Opt = {0};
    int speed_arr[] = { B38400, B19200, B9600, B4800, B2400, B1200, B300,
              B38400, B19200, B9600, B4800, B2400, B1200, B300, };
    int name_arr[] = {38400,  19200,  9600,  4800,  2400,  1200,  300, 38400,
              19200,  9600, 4800, 2400, 1200,  300, };

    tcgetattr(fd, &Opt);

    for ( i= 0;  i < sizeof(speed_arr) / sizeof(int);  i++) {
        if  (speed == name_arr[i])
            break;
    }

    tcflush(fd, TCIOFLUSH);
    cfsetispeed(&Opt, speed_arr[i]);
    cfsetospeed(&Opt, speed_arr[i]);

    Opt.c_lflag  &= ~(ICANON | ECHO | ECHOE | ISIG);  /*Input*/
    Opt.c_oflag  &= ~OPOST;   /*Output*/

    status = tcsetattr(fd, TCSANOW, &Opt);
    if  (status != 0) {
        DBG("tcsetattr fd");
        return;
    }
    tcflush(fd, TCIOFLUSH);
}

/**
*@brief   设置串口数据位，停止位和效验位
*@param  fd     类型  int  打开的串口文件句柄
*@param  databits 类型  int 数据位   取值 为 7 或者8
*@param  stopbits 类型  int 停止位   取值为 1 或者2
*@param  parity  类型  int  效验类型 取值为N,E,O,,S
*/
int set_Parity(int fd,int databits,int stopbits,int parity)
{
    struct termios options;

    if  ( tcgetattr( fd,&options)  !=  0) {
        perror("SetupSerial 1");
        return -1;
    }
    options.c_cflag &= ~CSIZE;

    switch (databits) /*设置数据位数*/
    {
    case 7:
        options.c_cflag |= CS7;
        break;
    case 8:
        options.c_cflag |= CS8;
        break;
    default:
        fprintf(stderr,"Unsupported data size\n");
        return -1;
    }

```

```
    switch (parity)
    {
    case 'n':
    case 'N':
        options.c_cflag &= ~PARENB;   /* Clear parity enable */
        options.c_iflag &= ~INPCK;     /* Enable parity checking */
        break;
    case 'o':
    case 'O':
        options.c_cflag |= (PARODD | PARENB); /* 设置为奇效验*/
        options.c_iflag |= INPCK;             /* Disnable parity checking */
        break;
    case 'e':
    case 'E':
        options.c_cflag |= PARENB;     /* Enable parity */
        options.c_cflag &= ~PARODD;   /* 转换为偶效验*/
        options.c_iflag |= INPCK;       /* Disnable parity checking */
        break;
    case 'S':
    case 's':  /*as no parity*/
        options.c_cflag &= ~PARENB;
        options.c_cflag &= ~CSTOPB;break;
    default:
        fprintf(stderr,"Unsupported parity\n");
        return -1;
    }

    /* 设置停止位*/
    switch (stopbits)
    {
        case 1:
            options.c_cflag &= ~CSTOPB;
            break;
        case 2:
            options.c_cflag |= CSTOPB;
           break;
        default:
             fprintf(stderr,"Unsupported stop bits\n");
             return -1;
    }

    /* Set input parity option */
    if (parity != 'n')
        options.c_iflag |= INPCK;
    tcflush(fd,TCIFLUSH);
    options.c_cc[VTIME] = 150; /* 设置超时15 seconds*/
    options.c_cc[VMIN] = 0; /* Update the options and do it NOW */
    if (tcsetattr(fd,TCSANOW,&options) != 0)
    {
        perror("SetupSerial 3");
        return -1;
    }
    return 0;
}

void str_print(char *buf, int len)
{
    int i;

    for (i=0; i<len; i++) {
```

```
168         if (i%10 == 0)
169             printf("\n");
170
171         printf("0x%02x ", buf[i]);
172     }
173     printf("\n");
174 }
175
176 int main(int argc, char **argv)
177 {
178     int i = 0;
179     int fd = 0;
180     int cnt = 0;
181     char buf[256];
182
183     int ret;
184     fd_set rd_fdset;
185     struct timeval dly_tm;      // delay time in select()
186
187     if (argc != PT_NUM) {
188         usage();
189         return -1;
190     }
191
192     sscanf(argv[PT_CYCLE], "%d", &cnt);
193     if (cnt == 0)
194         cnt = 0xFFFF;
195
196     fd = OpenDev(argv[PT_DEV_NAME]);
197     if (fd < 0)
198         return -1;
199
200     set_speed(fd,19200);
201     if (set_Parity(fd,8,1,'N') == -1)  {
202         printf("Set Parity Error\n");
203         exit (0);
204     }
205
206     printf("Select(%s), Cnt %d. \n", argv[PT_DEV_NAME], cnt);
207     while (i<cnt) {
208         FD_ZERO(&rd_fdset);
209         FD_SET(fd, &rd_fdset);
210
211         dly_tm.tv_sec  = 5;
212         dly_tm.tv_usec = 0;
213         memset(buf, 0, 256);
214
215         ret = select(fd+1, &rd_fdset, NULL, NULL, &dly_tm);
216 //      DBG("select() return %d, fd = %d", ret, fd);
217         if (ret == 0)
218             continue;
219
220         if (ret < 0) {
221             printf("select(%s) return %d. [%d]: %s \n", argv[PT_DEV_NAME], ret, errno,
    strerror(errno));
222             continue;
223         }
224
225         i++;
226         ret = read(fd, buf, 256);
```

```
227         printf("Cnt%d: read(%s) return %d.\n", i, argv[PT_DEV_NAME], ret);
228         str_print(buf, ret);
229     }
230
231     close(fd);
232     return 0;
233 }
```

6 FAQ

6.1 UART 调试打印开关

6.1.1 通过 debugfs 使用命令打开调试开关

注：内核需打开 CONFIG_DYNAMIC_DEBUG 宏定义

```
1.挂载debugfs
mount -t debugfs none /sys/kernel/debug
2.打开uart模块所有打印
echo "module sunxi_uart +p" > /mnt/dynamic_debug/control
3.打开指定文件的所有打印
echo "file sunxi-uart.c +p" > /mnt/dynamic_debug/control
4.打开指定文件指定行的打印
echo "file sunxi-uart.c line 615 +p" > /mnt/dynamic_debug/control
5.打开指定函数名的打印
echo "func sw_uart_set_termios +p" > /mnt/dynamic_debug/control
6.关闭打印
把上面相应命令中的+p 修改为-p 即可
更多信息可参考linux 内核文档：linux-3.10/Documentation/dynamic-debug-howto.txt
```

6.1.2 sysfs 调试接口

UART 驱动通过 sysfs 节点提供了几个在线调试的接口。

```
1./sys/devices/platform/soc/uart0/dev_info

cupid-p2:/ # cat /sys/devices/platform/soc/uart0/dev_info
id     = 0
name   = uart0
irq    = 247
io_num = 2
port->mapbase = 0x0000000005000000
port->membase = 0xffffff800b005000
port->iobase  = 0x00000000
pdata->regulator    = 0x          (null)
pdata->regulator_id =

从该节点可以看到uart端口的一些硬件资源信息

2./sys/devices/platform/soc/uart0/ctrl_info
cupid-p2:/ # cat /sys/devices/platform/soc/uart0/ctrl_info
 ier  : 0x05
 lcr  : 0x13
 mcr  : 0x03
```

```
 fcr  : 0xb1
 dll  : 0x0d
 dlh  : 0x00
 last baud : 115384 (dl = 13)

TxRx Statistics:
 tx     : 61123
 rx     : 351
 parity : 0
 frame  : 0
 overrun: 0
```

此节点可以打印出软件中保存的一些控制信息，如当前UART 端口的寄存器值、收发数据的统计等

```
3./sys/devices/platform/soc/uart0/status
cupid-p2:/ # cat /sys/devices/platform/soc/uart0/status
uartclk = 24000000
The Uart controller register[Base: 0xffffff800b005000]:
[RTX] 0x00 = 0x0000000d, [IER] 0x04 = 0x00000005, [FCR] 0x08 = 0x000000c1
[LCR] 0x0c = 0x00000013, [MCR] 0x10 = 0x00000003, [LSR] 0x14 = 0x00000060
[MSR] 0x18 = 0x00000000, [SCH] 0x1c = 0x00000000, [USR] 0x7c = 0x00000006
[TFL] 0x80 = 0x00000000, [RFL] 0x84 = 0x00000000, [HALT] 0xa4 = 0x00000002
```

此节点可以打印出当前UART 端口的一些运行状态信息，包括控制器的各寄存器值

著作权声明

商标声明

免责声明