全 志 科 技
Allwinner Technology

A20 uart 使用说明

V1.0

2013-08-08

Revision History

Version	Date	Changes compared to previous issue
V1.0	2013-03-15	初建版本

目录

1. 前言

1.1. 编写目的

了解 uart 的相关原理以及 uart 在 A20 上面的开发与使用方法。

1.2. 使用范围

Allwinner A20 平台。

1.3. 相关人员

A20 平台 uart 控制器驱动的负责人，uart 接口设备的驱动开发人员以及应用使用相关人员。

2. 模块介绍

2.1. 功能介绍

UniversalAsynchronousReceiver/Transmitter，UART 是通用异步收发器（异步串行通信口）的英文缩写，它包括了 RS232、RS449、RS423、RS422 和 RS485 等接口标准规范和总线标准规范我、，A20 使用的是 RS232 标准。

UART 也称串行通信接口（通常指 COM 接口），是采用串行通信方式的扩展接口。串口的数据是一位位地顺序传输，通信线路简单，只要一对传输线就可以实现双向通信。

2.2. 体系结构描述

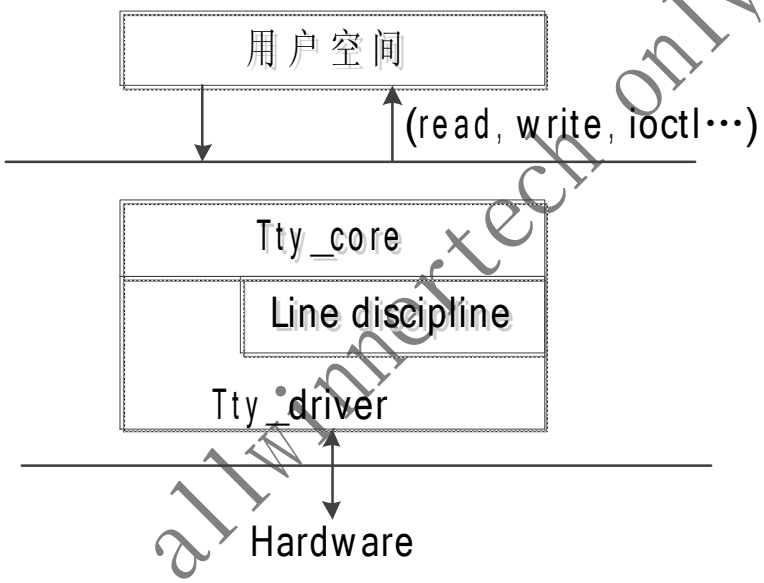

图 1 tty 分层结构

从上图可以看到，uart 设备是继 tty_driver 的又一层封装，实际上 uart_driver 就是对应 tty_driver，在它的操作函数中，将操作转入 uart_port。

tty_io.c 定义了设备通用的 file_operations 结构体并实现了接口函数 tty_register_driver()用于注册 tty 设备，它会利用 fs/char_dev.c 提供的接口函数注册字符设备，与具体设备对应的 tty 驱动将实现 tty_driver 结构体中的成员函数。同时 tty_io.c 也提供了 tty_register_ldisc()接口函数用于注册线路规程。tty 设备驱动的主体工作是填充 tty_driver 结构体中的成员，实现其中的成员函数。

图 2 为 linux uart 的层次图，serial_core.c 串口核心层为 tty 设备驱动的实例，它实现了 UART 设备的 tty 驱动。sw_uart.c 为 A20 平台上具体 uart 端口驱动的实现。

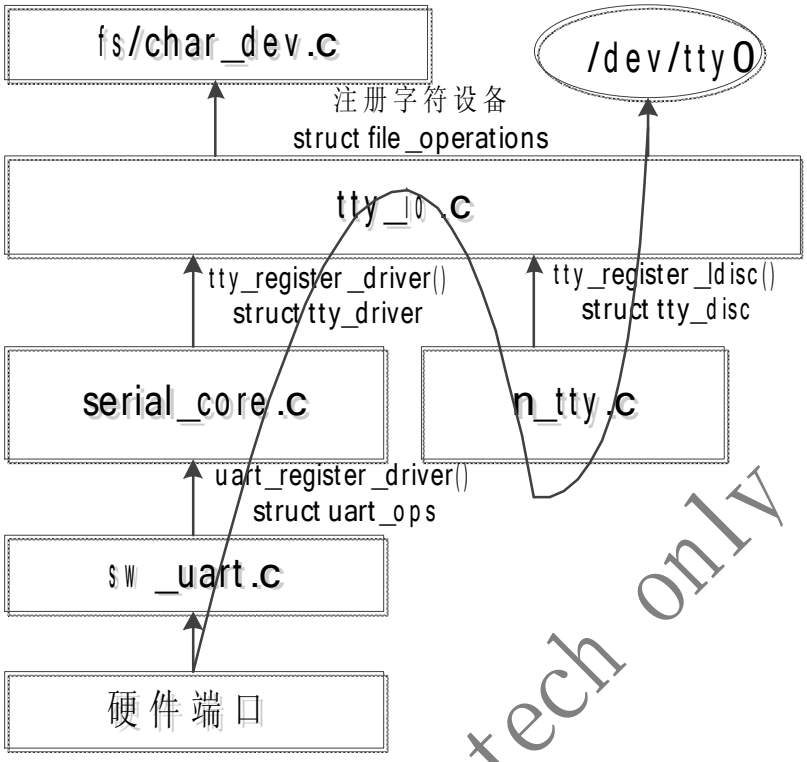

图 2 linux uart 层次图

2.3. 硬件介绍

2.3.1. 硬件结构框图

硬件结构框图如下所示。

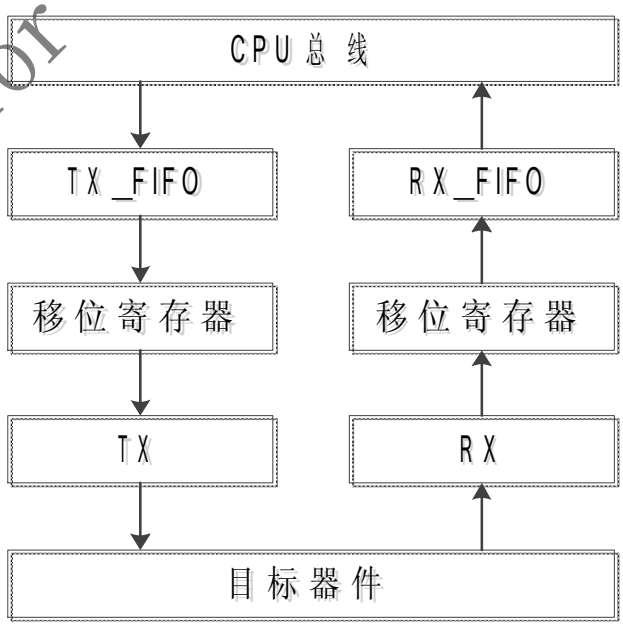

图 3 uart 硬件框图

2.3.2. A20 UART 口 GPIO 说明

A20 上面有 8 组 uart 接口，其中 UART1 支持所有的 modem 控制信号，包括 RTS, CTS, DTR, DSR, DCD 和 RING，UART2/UART3 支持自动流控，其他的串口只支持两个数据信号（DIN 和 DOUT），即 TX/RX。

硬件接口解释

信号	DB-25	DB-27	EIA/TIA 561
公共接地	7	5	4
发送数据（TD、TXD）	2	3	6
接收数据（RD、RXD）	3	2	5
数据终端设备（DTR）	20	4	3
数据准备好（DSR）	6	6	1
请求发送（RTS）	4	7	8
清除发送（CTS）	5	8	7
数据载波检测（DCD）	8	1	2
振铃指示（RI）	22	9	1

A20 中各个串口使用的 GPIO 列如下表。

引脚名称	GPIO 编号	GPIO 功能编号
UART0-TX	PB22	2
	PF2	4
UART0-RX	PB23	2
	PF4	4

引脚名称	GPIO 编号	GPIO 功能编号
UART1-TX	PA10	4
UART1-RX	PA11	4
UART1-RTS	PA12	4
UART1-CTS	PA13	4
UART1-DTR	PA14	4
UART1-DSR	PA15	4
UART1-DCD	PA16	4
UART1-RING	PA17	4

引脚名称	GPIO 编号	GPIO 功能编号
UART2-TX	PA2	4
	PI18	3
UART2-RX	PA3	4

	PI19	3
UART2-RTS	PA0	4
	PI16	3
UART2-CTS	PA1	4
	PI17	3
引脚名称	GPIO 编号	GPIO 功能编号
UART3-TX	PG6	4
	PH0	4
UART3-RX	PG7	4
	PH1	4
UART3-RTS	PG8	4
	PH2	4
UART3-CTS	PG9	4
	PH3	4

引脚名称	GPIO 编号	GPIO 功能编号
UART4-TX	PG10	4
	PH4	4
UART4-RX	PG11	4
	PH5	4

引脚名称	GPIO 编号	GPIO 功能编号
UART5-TX	PI10	3
	PH6	4
UART5-RX	PI11	3
	PH7	4

引脚名称	GPIO 编号	GPIO 功能编号
UART6-TX	PA12	3
	PI12	3
UART6-RX	PA13	3
	PI13	3

引脚名称	GPIO 编号	GPIO 功能编号
UART7-TX	PA14	3
	PI20	3
UART7-RX	PA15	3
	PI21	3

2.3.3. UART 标准时序图

uart 标准时序图如下所示：

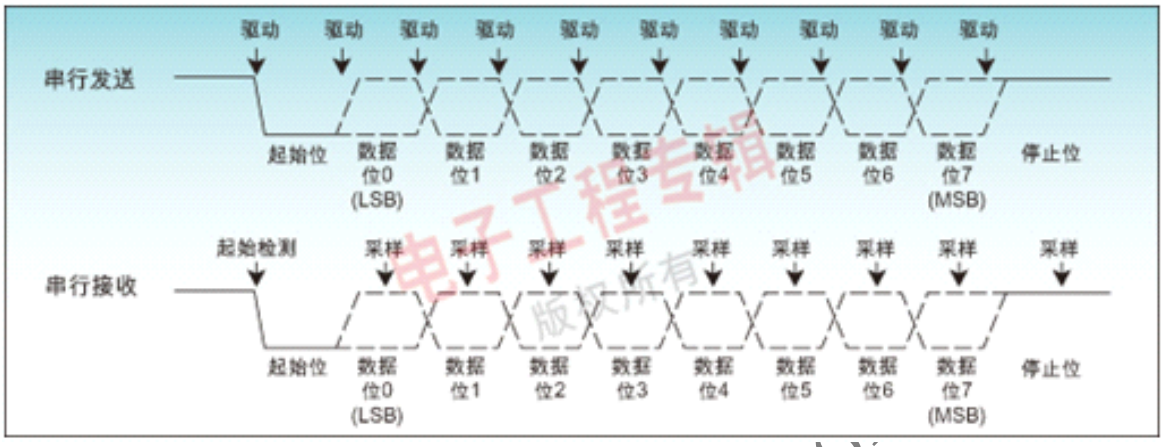

图 4 uart 标准时序图

2.4. 源码结构介绍

A20 uart 接口的驱动源码位置：lichee\linux-3.x\drivers\tty\serial\sw_uart.c

2.5. 模块配置介绍

2.5.1. sys_config.fex 配置

配置文件的位置：

\lichee\tools\pack\chips\sun7i\configs\android\wing-XXX 目录下。

截取文件中部分配置项，如下所示：

```
;------------------------------------------------------------------------
;uart configuration
;uart_type ---  2 (2 wire), 4 (4 wire), 8 (8 wire, full function)
;------------------------------------------------------------------------
[uart_para0]
uart_used           = 1
uart_port           = 0
uart_type           = 2
uart_tx             = port:PB22<2><1><default><default>
uart_rx             = port:PB23<2><1><default><default>
```

```
[uart_para1]
uart_used           = 0
uart_port           = 1
uart_type           = 8
uart_tx             = port:PA10<4><1><default><default>
uart_rx             = port:PA11<4><1><default><default>
uart_rts            = port:PA12<4><1><default><default>
uart_cts            = port:PA13<4><1><default><default>
uart_dtr            = port:PA14<4><1><default><default>
uart_dsr            = port:PA15<4><1><default><default>
uart_dcd            = port:PA16<4><1><default><default>
uart_ring           = port:PA17<4><1><default><default>

[uart_para2]
uart_used           = 1
uart_port           = 2
uart_type           = 4
uart_tx             = port:PI18<3><1><default><default>
uart_rx             = port:PI19<3><1><default><default>
uart_rts            = port:PI16<3><1><default><default>
uart_cts            = port:PI17<3><1><default><default>
```

uart_used

uart_used 置 1 时表示启动该串口，置 0 时表示禁用该串口。如果发现某个串口完全不能通信，请先确保对应的 uart_used 置 1。

uart_port

uart_port 指串口硬件的编号，与段名 uart_partx 的 x 一致。

uart_type

uart_type 表示几线串口通信。2 表示两线通信，即只有 TX 和 RX；4 表示支持自动流控，属于 4 线通信，除了 TX 和 RX 之后，还需要 RTS 和 CTS；8 表示支持所有的 modem 控制信号，一般不建议用，驱动也没有针对这种模式完全测试。

uart_tx、uart_rx、uart_rts 和 uart_cts 就是对应的 GPIO 配置，需要确保串口需要使用的 GPIO 没有和其他模块的发生冲突。

GPIO 配置的格式是“port:”后面跟着 GPIO 的名字（端口分组+组内序号，组内序号从 0 开始）；第一个尖括号是 GPIO 的功能选择；第二个尖括号是内置电阻（内置电阻是 IC 提供给引脚的电路属性，即上拉电阻或者下拉电阻，通常使用上拉电阻，即是默认状态），0 表示内部电阻高阻态，1 表示内部电阻上拉，2 表示内部电阻下拉，default 表示默认状态，电阻上拉；第三个尖括号是驱动能力（驱

动能力表示提供给当前 GPIO 的能力水平，这个数值越大，在 IO 口上的电平变化将越陡峭），共有 4 个等级（0、1、2、3），通常设为 default；第四个尖括号是输出电平，只在 GPIO 功能选择配成输出（即第一个尖括号配成 1）才会生效，0 表示低电平，1 表示高电平。

2.5.2. 打印串口的配置

Bootloader 打印串口配置

Bootloader 打印串口需要配置 sys_config1.fex 的[uart_para]段。如下：

```
[uart_para]
uart_debug_port      = 0
uart_debug_tx        = port:PB22<2><1><default><default>
uart_debug_rx        = port:PB23<2><1><default><default>
```

uart_debug_port 表示使用哪个串口作为 Bootloader（Boot0/Boot1）的打印输出。uart_debug_tx 和 uart_debug_rx 是打印串口 TX 和 RX 的 GPIO 配置。

U-Boot 打印串口配置

目前 U-Boot 的打印串口无法通过配置文件修改，A20 默认使用串口 0 作为打印串口。

内核/Android 打印串口配置

内核/Android 打印串口配置涉及三个文件，分别是

lichee\tools\pack\chips\sun7i\configs\android\default\env.cfg，

android4.x\device\softwinner\wing-common\init.rc,

android4.x\device\softwinner\wing-xxx\BoardConfig.mk。(wing-xxx 为具体的方案目录)

第一个文件 env.cfg 中 console=ttyS0,115200 指定了内核的 console，但是如果第三个文件 BoardConfig.mk 指定了 BOARD_KERNEL_CMDLINE，就会覆盖第一个文件 env.cfg 中的配置。例如：

```
BOARD_KERNEL_CMDLINE := console=ttyS0,115200 rw init=/init loglevel=8
```

第二个文件启动了一个 console 服务，如下：

```
service console /system/bin/sh /dev/ttyS0
    class core
    console
```

/dev/ttyS0 表示使用哪个串口作为/system/bin/sh 的输入输出。

因此，如果需要修改内核/Android 打印串口，必须修改上述的 3 个文件。

注意：这里的 ttyS0 是逻辑的编号，而非 sys_config1.fex 里 uart_parax 的 uart_port，即启用的第几个 uart。

2.5.3. menuconfig 的配置

在编译服务器上，目录为\lichee\linux-3.x 上，输入命令：

```
make ARCH=arm menuconfig
```

进入目录 Device Drivers\Character devices\Serial drivers 目录下可以看到 winner serial ports 为编译进内

核，A20 中 uart 默认编译进内核。

2.6. UART 设备节点说明

串行端口终端在 Linux 内核中对应的设备文件是/dev/ttySn，n = 0, 1, 2, …, 7。这个编号 n 和 sys_config1.fex 里 uart_parax 的 uart_port 没有一一对应的关系，一般理解为第几个启用（uart_used 置 1）的串口。例如：sys_config1.fex 里启用了 0 和 2（即[uart_para0] 和[uart_para2]的 uart_used 置 1），那么，它们对应的设备文件就是/dev/ttyS0 和/dev/ttyS1，而非/dev/ttyS0 和/dev/ttyS2。

3. UART 通信协议

UART 作为异步串口通信协议的一种，工作原理是将传输数据的每个字符一位接一位的传输。如下图所示：

图 5

起始位：先发出一个逻辑“0”的信号，表示传输字符的开始。

数据位：紧接着起始位之后，数据位的个数可以是 4，5，6，7，8 等，构成一个字符。通常采用 ASCII 码。从最低位开始传送，靠时钟定位。

奇偶校验位：数据位加上这一位后，使得“1”的位数应为偶数（偶校验）或奇数（奇校验），以此来校验资料传送的正确性。

停止位：它是一个字符数据结束的标志。可以是 1，1.5，2 位高电平。由于数据是在传输线上定时的，并且每一个设备有其自己的时钟，很可能在通信中两台设备间出现了小小的不同步。因此停止位不仅仅是表示传输的结束，并且提供计算机校正时钟同步的机会。适用于停止位的位数越多，不同时钟同步的容忍程度越大，但是数据传输率同时也越慢。

空闲位：处于逻辑“1”状态，表示当前线路上没有资料传送。

波特率：是衡量数据传输速率的指标。表示每秒钟传送的二进制位数。

4. UART 驱动中重要数据结构及关联

uart 中重要数据结构体关联图如下所示：

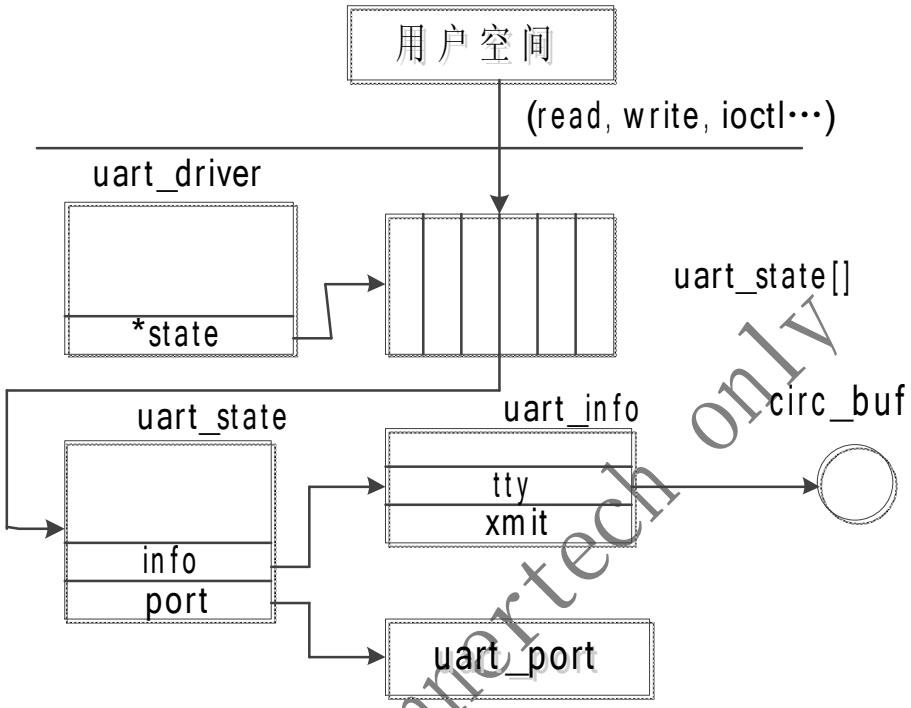

图 6 uart 结构体关联图

一个 uart_driver 通常会注册一段设备号.即在用户空间会看到 uart_driver 对应有多个设备节点。

/dev/ttyS0 /dev/ttyS1 每个设备节点是对应一个具体硬件的,这样就可做到对多个硬件设备的统一管理，而每个设备文件应该对应一个 uart_port，也就是说:uart_device 要和多个 uart_port 关系起来。并且每个 uart_port 对应一个 circ_buf（用来接收数据）,所以 uart_port 必须要和这个缓存区关系起来。

4.1. uart_driver

uart_driver 包含串口设备的驱动名，设备名，设备号等信息，它封装了 tty_driver，使得底层的 UART 无需关心 tty_driver。该结构体定义的位置如下：

lichee\linux-3.x\include\linux\serial_core.h

其定义如下所示：

```
struct uart_driver {
        struct module        *owner;
        const char           *driver_name;
        const char           *dev_name;
        int                  major;
        int                  minor;
        int                  nr;
```

```
        struct console      *cons;

        /*
         * these are private; the low level driver should not
         * touch these; they should be initialised to NULL
         */
        struct uart_state   *state;
        struct tty_driver   *tty_driver;
};
```

4.2. uart_port

uart_port 用于描述一个 UART 端口（直接对应一个串口）的 I/O 端口或 I/O 内存地址、FIFO 大小，端口类型等信息，该结构体定义的位置如下：

lichee\linux-3.x\include\linux\serial_core.h

4.3. uart_ops

uart_ops 定义了针对 UART 的一些列操作，包括发送、接收及线路设置等，如果说 tty_driver 中的 tty_operation 对于串口还较为抽象，那么 uart_ops 则直接面向了串口的 UART。该结构体定义的位置如下：

lichee\linux-3.x\include\linux\serial_core.h

4.4. uart_state

每一个 uart 端口对应这一个 uart_state,该结构体将 uart_port 与相对应的 circ_buf 联系起来。该结构体定义的位置如下：

lichee\linux-3.x\include\linux\serial_core.h

5. UART 驱动中关键接口

5.1. uart_register_driver

int uart_register_driver(struct uart_driver *uart);
该函数用于将 uart driver 注册到 tty 系统中。

5.2. uart_register_driver

void uart_unregister_driver(struct uart_driver *drv)
该函数用于注销一个 uart driver

5.3. uart_add_one_port

int uart_add_one_port(struct uart_driver *drv, struct uart_port *uport)
该函数将 uart_port 与 uart_driver 关联起来。

5.4. uart_remove_one_port

int uart_remove_one_port(struct uart_driver *drv, struct uart_port *uport)
该函数用于删除 uart_port 与 uart_driver 的关联。

5.5. uart_open

static int uart_open(struct tty_struct *tty, struct file *filp)
在用户空间执行 open 操作的时候，就会执行到 uart_open 函数。函数中继续完成操作的设备文件所对应 state 的初始化，在用户空间 open 这个设备之后，将要对文件进行操作，uart_port 也开始工作，调用 uart_startup()使其进入工作状态，初始化对应的环形缓冲区 circ_buf.

6. UART 应用层接口

用户空间的相关接口存在与 termios.h 头文件中，因此应用程序中应该添加该头文件。

6.1. tcgetattr

```
int tcgetattr(int fd, struct termios *s)
```

该函数用于获取终端设备的操作模式。

6.2. tcsetattr

```
tcsetattr(int fd, int __opt, const struct termios *s)
```

该函数用于设置终端设备的操作模式。

6.3. cfgetospeed

```
speed_t cfgetospeed(const struct termios *s)
```

该函数用于获取输出波特率

6.4. cfsetospeed

```
int cfsetospeed(struct termios *s, speed_t    speed)
```

该函数用于设置输出波特率

6.5. cfgetispeed

```
speed_t cfgetispeed(const struct termios *s)
```

该函数用于获取输入波特率。

6.6. cfsetispeed

```
int cfsetispeed(struct termios *s, speed_t    speed)
```

该函数用于设置输入波特率。

6.7. tcflush

```
int tcflush(int fd, int __queue)
```

该函数用于 flush 输入/输出缓冲区。

7. UART 应用 demo

应用的使用步骤：

（1）根据 uart 口的设备节点号，打开设备。

（2）设置 uart 口数据传输格式、波特率等参数。

（3）对 uart 进行相关的操作，写入或者读取。

Demo 如下所示：

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <termios.h>//必须包含此头文件
#include <errno.h>

#define FALSE  0
#define TRUE   1

int speed_arr[] = {B115200, B57600, B38400, B19200, B9600, B4800, B2400, B1200, B300, B38400, B19200, B9600, B4800, B2400, B1200, B300,}; //波特率
int name_arr[] = {115200, 57600, 38400,  19200,  9600,  4800,  2400,  1200,  300,  38400, 19200,  9600, 4800, 2400, 1200,  300, };

void set_speed(int fd, int speed)
{
        int     i;
        int     status;
        struct  termios opt;

        tcgetattr(fd, &opt);
        for( i = 0; i < sizeof(speed_arr)/sizeof(int); i++) {
                if(speed == name_arr[i]) {
```

```
                        printf("set speed!\n");
                        tcflush(fd, TCIOFLUSH);
                        cfsetispeed(&opt, speed_arr[i]);
                        cfsetospeed(&opt, speed_arr[i]);
                        status = tcsetattr(fd, TCSANOW, &opt);
                        if(status != 0)
                                perror("tcsetattr fd1");
                        return;
                }
                printf("no set speed!\n");
                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(FALSE);
        }

        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 (FALSE);
        }

        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 (FALSE);
        }
        /*  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 (FALSE);
        }

        /* Set input parity option */
        if (parity != 'n')
                options.c_iflag |= INPCK;

        options.c_cc[VTIME] = 150; // 15 seconds
        options.c_cc[VMIN] = 0;

        tcflush(fd,TCIFLUSH); /* Update the options and do it NOW */
        if (tcsetattr(fd,TCSANOW,&options) != 0)    {
                perror("SetupSerial 3");
                return (FALSE);
        }
        return (TRUE);
    }
/**
```

```
*@breif 打开串口
*/
int OpenDev(char *Dev)
{
int fd = open( Dev, O_RDWR );                //| O_NOCTTY | O_NDELAY
        if (-1 == fd)    { /*设置数据位数*/
                perror("Can't Open Serial Port");
                return -1;
        }    else
                return fd;

}

static void changestr(char *str, char * out_str)
{
        int i = 0;
        int j = 0;

        while((str[i++]) && (i <= 1024)) {

                if(str[i-1] == ',') {
                        out_str[j] = ' ';
                }else
                        out_str[j] = str[i-1];
                j++;
        }
}
/**
*@breif        main()
*/
int main(int argc, char **argv)
{
```

```
        int fd;
        int nread;
        char buff[512];
        char *dev ="/dev/ttyS0";
        char write_info[1024];
        char write_ch[256];

        fd = OpenDev(dev);   //根据设备节点，打开相应的 uart 口
        if (fd>0)
                set_speed(fd,115200);   //设备波特率
        else {
                printf("Can't Open Serial Port!\n");
                exit(0);
        }

        if (set_Parity(fd,8,1,'N')== FALSE)   {   //设置数据格式
                printf("Set Parity Error\n");
                exit(1);
        }
        scanf("%s", write_info);
        changestr(write_info, write_ch);
        sprintf(write_ch,"%s\n",write_ch);

        printf("write_info:%s\n", write_info);

        nread = write(fd, write_ch, sizeof(write_ch)/sizeof(char));//写测试

        close(fd);

//  while(1)
//      {
//          while((nread = read(fd,buff,512))>0)   //读测试
```

```
//              {
//                      printf("\nLen %d\n",nread);
//                      buff[nread+1]='\0';
//                      printf("\n%s",buff);
//              }
//      }
//      //close(fd);
//      //exit(0);
}
```

8. Declaration

This document is the original work and copyrighted property of Allwinner Technology ("Allwinner"). Reproduction in whole or in part must obtain the written approval of Allwinner and give clear acknowledgement to the copyright owner.

The information furnished by Allwinner is believed to be accurate and reliable. Allwinner reserves the right to make changes in circuit design and/or specifications at any time without notice. Allwinner does not assume any responsibility and liability for its use. Nor for any infringements of patents or other rights of the third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of Allwinner. This datasheet neither states nor implies warranty of any kind, including fitness for any particular application.