

求投食~ (点图即可)

昵称：silencehuan

园龄：3年6个月

粉丝：53

关注：0

+加关注

< 2021年9月 >						
日	一	二	三	四	五	六
29	30	31	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	1	2
3	4	5	6	7	8	9

搜索

找找我

谷歌搜索

- 常用链接
- 我的随笔

我的评论

我的参与

最新评论

我的标签

- 我的标签
- Linux(16)

RT-Thread(7)

STM32(5)

NRF52832(4)

openwrt(3)

arm(3)

GoAhead(3)

深度学习(2)

vs code(2)

串口(2)

更多

- 随笔分类
- C#(1)

Linux(1)

NRF52832(4)

OpenWRT(3)

RT-Thread(1)

深度学习(2)

- 随笔档案
- 2021年9月(3)

2019年12月(5)

2019年11月(3)

2019年9月(1)

2019年8月(2)

2019年7月(12)

2019年6月(10)

2019年5月(8)

2018年5月(2)

- 阅读排行榜
1. 代码注释规范之Doxygen(29760)

2. Linux串口调试详解(29730)

3. VS Code 搭建stm32开发环境(19217)

4. NRF52832初步使用(19026)

5. freemodbus移植、实例及其测试方法(13164)

Linux串口调试详解

测试平台

宿主机平台：Ubuntu 16.04.6

目标机：iMX6ULL

目标机内核：Linux 4.1.15

目标机添加串口设备

一般嵌入式主板的默认镜像可能只配置了调试串口，并用于 console 控制台打印；

接下来对怎么样通过设备树来分配引脚用于用户串口通信 进行描述；

前提：

目标机以及正常烧录 uboot、内核、文件系统、dtb等；本文仅更新设备树dtb文件；

设备树文件修改

在内核源码中找到相关板子对应的dtb文件；

位置：arch/arm/boot/dts 目录下

本文使用的板子相关文件有：

imx6ull.dtsi // 官方通用板层dtsi
mys-imx6ull-14x14-evk.dts // 基于imx6ull-14x14-evk.dts模板修改
mys-imx6ull-14x14-evk-gpmi-weim.dts // 用户层dts

添加 uart3和uart4 的支持，修改 mys-imx6ull-14x14-evk.dts 文件如下

```
pinctrl_uart2: uart2grp {
    fsl,pins = <
        MX6UL_PAD_UART2_TX_DATA__UART2_DCE_TX      0x1b0b1
        MX6UL_PAD_UART2_RX_DATA__UART2_DCE_RX      0x1b0b1
    >;
};

pinctrl_uart2dte: uart2dtegrp {
    fsl,pins = <
        MX6UL_PAD_UART2_TX_DATA__UART2_DTE_RX      0x1b0b1
        MX6UL_PAD_UART2_RX_DATA__UART2_DTE_TX      0x1b0b1
        MX6UL_PAD_UART3_RX_DATA__UART2_DTE_CTS     0x1b0b1
        MX6UL_PAD_UART3_TX_DATA__UART2_DTE_RTS     0x1b0b1
    >;
};

/* 增加uart3/4/5引脚配置 */
pinctrl_uart3: uart3grp {
    fsl,pins = <
        MX6UL_PAD_UART3_TX_DATA__UART3_DCE_TX      0x1b0b1
        MX6UL_PAD_UART3_RX_DATA__UART3_DCE_RX      0x1b0b1
    >;
};

pinctrl_uart4: uart4grp {
    fsl,pins = <
        MX6UL_PAD_UART4_TX_DATA__UART4_DCE_TX      0x1b0b1
        MX6UL_PAD_UART4_RX_DATA__UART4_DCE_RX      0x1b0b1
    >;
};

pinctrl_uart5: uart5grp {
    fsl,pins = <
        MX6UL_PAD_UART5_TX_DATA__UART5_DCE_TX      0x1b0b1
        MX6UL_PAD_UART5_RX_DATA__UART5_DCE_RX      0x1b0b1
    >;
};

.....

/* 使能串口 */
uart1 {
    pinctrl-names = "default";
    pinctrl-0 = <&pinctrl_uart1>;
    status = "okay";
};

uart2 {
    pinctrl-names = "default";
    pinctrl-0 = <&pinctrl_uart2>;
    /*fsl,uart-has-rtscts*/
    /* for DTE mode, add below change */
    /* fsl,dte-mode; */
    /* pinctrl-0 = <&pinctrl_uart2dte>; */
    status = "disabled";
};

/* 增加使用串口，其中使能3、关闭4/5 */
uart3 {
    pinctrl-names = "default";
    pinctrl-0 = <&pinctrl_uart3>;
    status = "okay";
};

uart4 {
    pinctrl-names = "default";
    pinctrl-0 = <&pinctrl_uart4>;
    status = "okay";
};

/* 这里必须注意一点，由于UART5和I2C2接口的引脚是复用的，I2C2默认是使能的所以必须禁用I2C2，再使能UART5。 */
uart5 {
    pinctrl-names = "default";
    pinctrl-0 = <&pinctrl_uart5>;
    status = "disabled";
};

};
```

然后重新编译生成 设备树 dtb 文件

```
cp arch/arm/configs/mys_imx6_defconfig .config
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- dtbs
```

生成的 dtb 文件：arch/arm/boot/dts/mys-imx6ull-14x14-evk-gpmi-weim.dtb

设备树文件更新

采用 MFGTool2 进行设备树更新，怎么单独仅更新设备树参见 <https://www.cnblogs.com/silencehuan/p/11010148.html>

mys-imx6ull-14x14-evk-gpmi-weim.dtb 替换 /Profiles/Wh Linux Update/OS Firmware/files/ 下面的 dtb文件

然后执行 mfgtool2-linux-mys-6ulx-nand-dtb.vbs

论坛排行榜
1. C#串口通信及数据表格存储(34)
2. freemodbus移植、实例及其测试方法(5)
3. VS Code 搭建stm32开发环境(4)
4. 代码注释规范之Doxygen(3)
5. GoAhead4.1.0 开发总结—（移植）(2)
推荐排行榜
1. 代码注释规范之Doxygen(6)
2. C#串口通信及数据表格存储(5)
3. freemodbus移植、实例及其测试方法(3)
4. Vs code添加自定义snippet(1)
5. Vs code自动生成Doxygen格式注释(1)
最新评论

1. Re:C#串口通信及数据表格存储
博主能分享下源码吗？ 1473412588@qq.com
--一行代码走天下
2. Re:C#串口通信及数据表格存储
源码能分享一下么，谢谢！ 779673130@qq.com
--洗晓松
3. Re:rt18188eu 驱动移植
老哥，我原本用正点原子的方法移植WiFi到busybox上是可以的，但是现在用Ubuntu根文件系统，移植这个WiFi驱动无法ping通外网了。现在用你的方法，将驱动直接编译进内核，用ifconfig...
--Xia冰
4. Re:Vs code自动生成Doxygen格式注释
博主 我这边设置之后，不起作用呢，是这个插件失效了吗
--jjhhee
5. Re:C#串口通信及数据表格存储
源码能分享一下么，谢谢！ zaitianzhiya1211@126.com
--tianya1122

```
Set wshShell = CreateObject("WScript.Shell")
wshShell.Run "mfgtool2.exe -c ""$H Linux Update"" -i ""$NAME-$th"" -s ""lite1"" -s ""6uluboot-14x14evk"" -s ""nand-nand"" -s ""6uluboot-14x14-evk"" -s ""nanddtb-gpmi-welm"" -s ""part_uboot=0"" -s ""part_kernel=1"" -s ""part_dtb=2"" -s ""part_rootfs=3"" -s ""ddrsize=256"" -s ""rootfs_name=core-image-base""
Set wshShell = Nothing
```

更新成功，设备重启之后，看到添加的串口设备已支持，串口驱动实现框架另外的文章在分析；

串口应用编程

1.串口相关操作

在Linux下，除了网络设备，其余的都是文件的形式，串口设备也一样在/dev下。

打开串口：

示例：`fd = open("/dev/ttyUSB0", O_RDWR|O_NOCTTY|O_NDELAY);`

在打开串口时，除了需要用到 O_RDWR (可读写)选项标志外，

O_NOCTTY:告诉Linux “本程序不作为串口的 ‘控制终端’ ”。如果不使用该选项，会有一些输入字符影响进程运行（如一些产生中断信号的键盘输入字符等）。

O_NDELAY:标志则是告诉Linux,这个程序并不关心DDB信号线的状态—也就是不关心端口另一端是否已经连接。

关闭串口：

`close(fd);`

读写串口：

与普通文件一样，使用read，write函数。

示例：`read(fd, buff, 8);`
`write(fd, buff, 8);`

2.串口属性设置

很多系统都支持POSIX终端(串口)接口,程序可以利用这个接口来改变终端的参数,比如,波特率,字符大小等等.要使用这个端口的话,你必须将<termios.h>头文件包含到你的程序中。这个头文件中定义了终端控制结构体和POSIX控制函数。

termios 结构体：

```
struct termios {
    tcflag_t c_iflag; /* 控制标志 */
    tcflag_t c_lflag; /* 输入标志 */
    tcflag_t c_oflag; /* 输出标志 */
    tcflag_t c_cflag; /* 本地标志 */
    tcflag_t c_cc[NCCS]; /* 控制字符 */
};
```

其中我们更关注的是 c_iflag控制选项。其中包含了波特率、数据位、校验位、停止位的设置。

c_iflag 控制标志常量：

- CBAUD (不属于 POSIX) 波特率掩码 (4+1 位)。
- CBAUDEXT (不属于 POSIX) 扩展的波特率掩码 (1 位)，包含在 CBAUD 中。
- (POSIX 规定波特率存储在 termios 结构中，并未精确指定它的位置，而是提供了函数 cfgetispeed() 和 cfsetispeed() 来存取它。一些系统使用 c_iflag 中 CBAUD 选择的位，其他系统使用单独的变量，例如 sg_ispeed 和 sg_ospeed。)
- CSIZE 字符长度掩码。取值为 CS5, CS6, CS7, 或 CS8。
- CSTOPB 设置两个停止位，而不是一个。
- CREAD 打开接受者。
- PARENB 允许输出产生奇偶信息以及输入的奇偶校验。
- PARODD 输入和输出是奇校验。
- HUPCL 在最后一个进程关闭设备后，降低 modem 控制线 (挂断)。 (?)
- CLOCAL 忽略 modem 控制线。
- LOBLK (不属于 POSIX) 从非当前 shell 层阻塞输出(用于 sh)。 (?)
- CIBAUD (不属于 POSIX) 输入速度的掩码。CIBAUD 各位的值与 CBAUD 各位相同，左移了 IBSHIFT 位。
- CRTSCTS (不属于 POSIX) 启用 RTS/CTS (硬件) 流控制。

c_iflag 输入标志常量：

- IGNBRK 忽略输入中的 BREAK 状态。
- BRKINT 如果设置了 IGNBRK，将忽略 BREAK。如果没有设置，但是设置了 BRKINT，那么 BREAK 将使得输入和输出队列被刷新，如果终端是一个前台进程组的控制终端，这个进程组中所有进程将收到 SIGINT 信号。如果既未设置 IGNBRK 也未设置 BRKINT，BREAK 将视为与 NUL 字符同义，除非设置了 PARMRK。这种情况下它被视为序列 \377 \0 \0。
- IGNPAR 忽略帧错误和奇偶校验错误。
- PARMRK 如果没有设置 IGNPAR，在有奇偶校验错误或帧错误的字符前插入 \377 \0。如果既没有设置 IGNPAR 也没有设置 PARMRK，将有奇偶校验错误或帧错误的字符视为 \0。
- INPCK 启用输入奇偶检测。
- ISTRIP 去掉第八位。
- INLCR 将输入中的 NL 翻译为 CR。
- IGNCR 忽略输入中的回车。
- ICRNL 将输入中的回车翻译为新行 (除非设置了 IGNCR)。
- IUCLC (不属于 POSIX) 将输入中的大写字母映射为小写字母。
- IXON 启用输出的 XON/XOFF 流控制。
- IXANY (不属于 POSIX.1；XSI) 允许任何字符来重新开始输出。 (?)
- IXOFF 启用输入的 XON/XOFF 流控制。
- IMAXBEL (不属于 POSIX) 当输入队列满时响铃。Linux 没有实现这一位，总是将它视为已设置。

c_oflag 输出标志常量：

- OPOST 启用具体实现自行定义的输出处理。
- 其余 c_oflag 标志常量定义在 POSIX 1003.1-2001 中，除非另外说明。
- OLCUC (不属于 POSIX) 将输出中的小写字母映射为大写字母。
- ONLCR (XS) 将输出中的新行符映射为回车-换行。
- OCRNL 将输出中的回车映射为新行符。
- ONOCR 不在第 0 列输出回车。
- ONLRET 不输出回车。
- OFILL 发送填充字符作为延时，而不是使用定时来延时。
- OFDEL (不属于 POSIX) 填充字符是 ASCII DEL (0177)。如果不设置，填充字符则是 ASCII NUL。
- NLDLY 新行延时掩码。取值为 NLD 和 NLL。
- CRDLY 回车延时掩码。取值为 CR0, CR1, CR2, 或 CR3。
- TABDLY 水平跳格延时掩码。取值为 TAB0, TAB1, TAB2, TAB3 (或 XTABS)。取值为 TAB3，即 XTABS，将扩展跳格为空格 (每个跳格符填充 8 个空格)。 (?)
- BSDLY 回退延时掩码。取值为 BS0 或 BS1。(从来没有被实现过)
- VTDLY 垂直跳格延时掩码。取值为 VT0 或 VT1。
- FFDLY 进表延时掩码。取值为 FF0 或 FF1。

c_lflag 本地标志常量：

- ISIG 当接受到字符 INTR, QUIT, SUSP, 或 DSUSP 时，产生相应的信号。
- ICANON 启用标准模式 (canonical mode)。允许使用特殊字符 EOF, EOL, ERASE, KILL, LNEXT, REPRINT, STATUS, 和 WERASE，以及按行的缓冲。
- XCASE (不属于 POSIX; Linux 下不被支持) 如果同时设置了 ICANON，终端只有大写。输入被转换为小写，除了以 \ 前缀的字符。输出时，大写字母被前缀 \，小写字母被转换成大写。
- ECHO 回显输入字符。
- ECHOE 如果同时设置了 ICANON，字符 ERASE 擦除前一个输入字符，WERASE 擦除前一个词。
- ECHOK 如果同时设置了 ICANON，字符 KILL 删除前一行。
- ECHONL 如果同时设置了 ICANON，回显字符 NL，即使没有设置 ECHO。
- ECHOCTL (不属于 POSIX) 如果同时设置了 ECHO，除了 TAB, NL, START, 和 STOP 之外的 ASCII 控制信号被回显为 ^X。这里 X 是比控制信号大 0x04 的 ASCII 码。例如，字符 0x08 (BS) 被回显为 ^H。
- ECHOPRT (不属于 POSIX) 如果同时设置了 ICANON 和 IECHO，字符在删除的同时被打印。
- ECHOKE (不属于 POSIX) 如果同时设置了 ICANON，回显 KILL 时将删除一行中的每个字符，如同指定了 ECHOE 和 ECHOPRT 一样。
- DEFECHO (不属于 POSIX) 只在一个进程读的时候回显。
- FLUSHO (不属于 POSIX; Linux 下不被支持) 输出被刷新。这个标志可以通过键入字符 DISCARD 来开关。
- NOFLSH 禁止在产生 SIGINT, SIGQUIT 和 SIGSUSP 信号时刷新输入和输出队列。
- TOSTOP 向试图写控制终端的后台进程组发送 SIGTTOU 信号。
- TEXTEN 启用实现自定义的输入处理。这个标志必须与 ICANON 同时使用，才能解释特殊字符 EOL2, LNEXT, REPRINT 和 WERASE，IUCLC 标志才有效。

c_cc 特殊的控制字符

- VINTR (003, ETX, Ctrl-C, or also 0177, DEL, rubout) 中断字符。发出 SIGINT 信号。当设置 ISIG 时可被识别，不再作为输入传递。
- VQUIT (034, FS, Ctrl-\) 退出字符。发出 SIGQUIT 信号。当设置 ISIG 时可被识别，不再作为输入传递。
- VERASE (0177, DEL, rubout, or 010, BS, Ctrl-H, or also #) 删除字符。删除上一个还没有删掉的字符，但不删除上一个 EOF 或行首。当设置 ICANON 时可被识别，不再作为输入传递。
- VKILL (025, NAK, Ctrl-U, or Ctrl-X, or also @) 终止字符。删除自上一个 EOF 或行首以来的输入。当设置 ICANON 时可被识别，不再作为输入传递。
- VEOF (004, EOT, Ctrl-D) 文件尾字符。更精确地说，这个字符使得 tty 缓冲中的内容被送到等待输入的用户程序中，而不必等到 EOL。如果它是一行的第一个字符，那么用户程序的 read() 将返回 0，指示读到了 EOF。当设置 ICANON 时可被识别，不再作为输入传递。
- VMIN 非 canonical 模式下的最小字符数。

VEOL (0, NUL) 附加的行尾字符。当设置 ICANON 时可被识别。
VTIME 非 canonical 模式读时的延时，以十分之一秒为单位。
VEOL2 (not in POSIX; 0, NUL) 另一个行尾字符。当设置 ICANON 时可被识别。
VSWTCH (not in POSIX; not supported under Linux; 0, NUL) 开关字符。(只为 sh1 所用。)
VSTART (021, DC1, Ctrl-Q) 开始字符。重新开始被 Stop 字符中止的输出。当设置 IXON 时可被识别，不再作为输入传递。
VSTOP (023, DC3, Ctrl-S) 停止字符。停止输出，直到输入 Start 字符。当设置 IXON 时可被识别，不再作为输入传递。
VSUSP (032, SUB, Ctrl-Z) 挂起字符。发送 SIGTSTP 信号。当设置 ISIG 时可被识别，不再作为输入传递。
VDSUSP (not in POSIX; not supported under Linux; 031, EM, Ctrl-Y) 延时挂起信号。当用户程序读到这个字符时，发送 SIGTSTP 信号。当设置 IEXTEN 和 ISIG，并且系统支持作业管理时可被识别，不再作为输入传递。
VLNEXT (not in POSIX; 026, SYN, Ctrl-V) 字面上的下一个。引用下一个输入字符，取消它的任何特殊含义。当设置 IEXTEN 时可被识别，不再作为输入传递。
VWERASE (not in POSIX; 027, ETB, Ctrl-W) 删除词。当设置 ICANON 和 IEXTEN 时可被识别，不再作为输入传递。
VREPRINT (not in POSIX; 022, DC2, Ctrl-R) 重新输出未读的字符。当设置 ICANON 和 IEXTEN 时可被识别，不再作为输入传递。
VDISCARD (not in POSIX; not supported under Linux; 017, SI, Ctrl-O) 开关：开始/结束丢弃未完成的输出。当设置 IEXTEN 时可被识别，不再作为输入传递。

调用read函数读取串口数据时，返回读取数据的数量需要考虑两个变量：MIN和TIME。

MIN和TIME在termios结构的c_cc成员的数组下标名为VMIN和VTIME。MIN是指一次read调用期望返回的最小字节数。VTIME说明等待数据到达的分秒数（秒的1/10为分秒）。TIME与MIN组合使用的具体含义分为以下四种情形：

当MIN>0TIME>0时 计时器在收到第一个字节后启动，在计时器超时之前TIME的时间到），若已收到MIN个字节，则read返回MIN个字节，否则，在计时器超时后返回实际接收到的字节。

注意：因为只有在接收到第一个字节时才开始计时，所以至少可以返回1个字节。这种情形中，在接到第一个字节之前，调用者阻塞。如果在调用read时数据已经可用，则如同在read后数据立即被接到一样。

当MIN>0TIME=0时 MIN个字节完整接收后，read才返回，这可能会造成read无限期地阻塞。

当MIN=0,TIME>0时 TIME为允许等待的最大时间，计时器在调用read时立即启动，在串口接到1字节数据或者计时器超时后即返回，如果是计时器超时，则返回0。

当MIN=0TIME=0时 如果有数据可用，则read最多返回所要求的字节数，如果无数据可用，则read立即返回0。

终端api函数接口：

tcgetattr() 得到与fd指向的对象相关的参数，将它们保存于termios_p引用的termios结构中。函数可以从后台进程中调用；但是，终端属性可能被后来的前台进程所改变。

tcsetattr() 设置与终端相关的参数（除非需要底层支持却无法满足），使用termios_p引用的termios结构。optional_actions指定了什么时候改变会起作用：

TCSANOW	改变立即发生
TCSADRAIN	改变在所有写入fd的输出都被传输后生效。这个函数应当用于修改影响输出的参数时使用。
TCSAFLUSH	改变在所有写入fd引用的对象的输出都被传输后生效，所有已接受但未读的输入都在改变发生前丢弃。

tcsendbreak() 发送连续的0值比流，持续一段时间，如果终端使用异步串行数据传输的话。如果duration是0，它至少传输0.25秒，不会超过0.5秒。如果duration非零，它发送的时间长度由实现定义。如果终端并非使用异步串行数据传输，tcsendbreak()什么都不做。

tcdrain() 等待直到所有写入fd引用的对象的输出都被传输。

tcflush() 丢弃要写入引用的对象，但是尚未传输的数据，或者收到但是尚未读取的数据，取决于queue_selector的值：

TCIFLUSH	刷新收到的数据但是不读
TCOFLUSH	刷新写入的数据但是不传送
TCIOFLUSH	同时刷新收到的数据但是不读，并且刷新写入的数据但是不传送

tcflow() 挂起fd引用的对象上的数据传输或接收，取决于action的值：

TCOOFF	挂起输出
TCOON	重新开始被挂起的输出
TCIOFF	发送一个STOP字符，停止终端设备向系统传送数据
TCION	发送一个START字符，使终端设备向系统传输数据

打开一个终端设备时的默认设置是输入和输出都没有挂起。

2.1 串口属性设置示例

设置串口属性主要是配置termios结构体中的各个变量，大致流程如下：

1.使用函数tcgetattr保存原串口属性

```
struct termios newtio,oldtio;  
tcgetattr(fd,&oldtio);
```

2.通过位掩码的方式激活本地连接和接受使能选项：CLOCAL和CREAD

```
newtio.c_cflag |= CLOCAL | CREAD;
```

3.使用函数cfsetispeed和cfsetospeed设置数据传输率

```
cfsetispeed(&newtio,B115200);  
cfsetospeed(&newtio,B115200);
```

4.通过位掩码设置字符大小。

```
newtio.c_cflag &= ~CSIZE;  
newtio.c_cflag |= CS8;
```

5.设置奇偶校验位需要用到两个termios中的成员：c_cflag和c_iflag。首先要激活c_cflag中的校验位使能标志PARENB和是否进行奇偶校验，同时还要激活c_iflag中的奇偶校验使能。

设置奇校验：

```
newtio.c_cflag |= PARENB;  
newtio.c_cflag |= PARODD;  
newtio.c_iflag |= (INPCK | ISTRIP);
```

设置偶校验：

```
newtio.c_iflag |= (INPCK | ISTRIP);  
newtio.c_cflag |= PARENB;  
newtio.c_cflag |= ~PARODD;
```

6.激活c_cflag中的CSTOPB设置停止位。若停止位为1，则清除CSTOPB；若停止位为0，则激活CSTOPB。

```
newtio.c_cflag &= ~CSTOPB;
```

7.设置最少字符和等待时间。在对接收字符和等待时间没有特别要求的情况下，可以将其设置为0。

```
newtio.c_cc[VTIME] = 0;  
newtio.c_cc[VMIN] = 0;
```

8.调用函数“tcflush(fd,queue_selector)”来处理要写入引用的对象，queue_selector可能的取值有以下几种。

TCIFLUSH:刷新收到的数据但是不读

TCOFLUSH:刷新写入的数据但是不传送

TCIOFLUSH:同时刷新收到的数据但是不读，并且刷新写入的数据但是不传送。

9.激活配置。在完成配置后，需要激活配置使其生效。使用tcsetattr()函数。

```
int tcsetattr(int fildes,int opt,const struct termios *termtp);
```

3.串口编程实例

备注：

1.串口设备操作要进行设备检查，设置之前最好进行termios参数清零；

2.串口读取要善于使用select函数；

```


/**@file      main.c
 * @brief      串口应用编程测试
 * @details
 * @author     wanghuan  any question please send mail to 371463817@qq.com
 * @date       2019-06-17
 * @version    V1.0
 * @copyright  Copyright (c) 2019-2022
 *
 * *****
 * @attention
 * 硬件平台:MMIO6ULL \n
 * 内核版本:4.1.15
 * @par 修改日志:
 * <table>
 * <tr><th>Date          <th>Version  <th>Author   <th>Description
 * <tr><td>2019/06/17  <td>1.0      <td>          <td>创建初始版本
 * </table>
 * *****
 */

/* 包含的头文件 */
#include <stdio.h>          //标准输入输出,如printf,scanf以及文件操作
#include <stdlib.h>         //标准库头文件,定义了五种类型,一些宏和通用工具函数
#include <unistd.h>         //定义 read write close lseek 等Unix标准函数
#include <sys/types.h>      //定义数据类型,如 ssize_t,off_t 等
#include <sys/stat.h>       //文件状态
#include <fcntl.h>          //文件控制定义
#include <termios.h>        //终端I/O
#include <errno.h>          //与全局变量 errno 相关的定义
#include <getopt.h>         //处理命令行参数
#include <string.h>         //字符串操作
#include <time.h>           //时间
```

```

#include <sys/select.h> //select函数

#define DEV_NAME    "/dev/ttymx3"    ///< 串口设备

/**@brief   设置串口参数:波特率, 数据位, 停止位和校验位
 * @param[in]  fd          类型   int      打开的串口文件句柄
 * @param[in]  nSpeed      类型   int      波特率
 * @param[in]  nBits       类型   int      数据位   取值为 7 或者8
 * @param[in]  nParity     类型   int      停止位   取值为 1 或者2
 * @param[in]  nStop       类型   int      校验类型   取值为N,E,O,,S
 * @return     返回设置结果
 * - 0         设置成功
 * - -1        设置失败
 */
int setOpt(int fd, int nSpeed, int nBits, int nParity, int nStop)
{
    struct termios newtio, oldtio;

    // 保存测试现有串口参数设置, 在这里如果串口号出错, 会有相关的出错信息
    if (tcgetattr(fd, &oldtio) != 0)
    {
        perror("SetupSerial 1");
        return -1;
    }

    bzero(&newtio, sizeof(newtio)); //新termios参数清零
    newtio.c_cflag |= CLOCAL | CREAD; //CLOCAL--忽略 modem 控制线, 本地连线, 不具数据机控制功能, CREAD--使能接收标志
    // 设置数据位数
    newtio.c_cflag &= ~CSIZE; //清数据位标志
    switch (nBits)
    {
        case 7:
            newtio.c_cflag |= CS7;
            break;
        case 8:
            newtio.c_cflag |= CS8;
            break;
        default:
            fprintf(stderr, "Unsupported data size\n");
            return -1;
    }

    // 设置校验位
    switch (nParity)
    {
        case 'o':
        case 'O':
            //奇校验
            newtio.c_cflag |= PARENB;
            newtio.c_cflag |= PARODD;
            newtio.c_cflag |= (INPCK | ISTRIP);
            break;
        case 'e':
        case 'E':
            //偶校验
            newtio.c_cflag |= (INPCK | ISTRIP);
            newtio.c_cflag |= PARENB;
            newtio.c_cflag &= ~PARODD;
            break;
        case 'n':
        case 'N':
            //无校验
            newtio.c_cflag &= ~PARENB;
            break;
        default:
            fprintf(stderr, "Unsupported parity\n");
            return -1;
    }

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

    // 设置波特率 2400/4800/9600/19200/38400/57600/115200/230400
    switch (nSpeed)
    {
        case 2400:
            cfsetispeed(&newtio, B2400);
            cfsetospeed(&newtio, B2400);
            break;
        case 4800:
            cfsetispeed(&newtio, B4800);
            cfsetospeed(&newtio, B4800);
            break;
        case 9600:
            cfsetispeed(&newtio, B9600);
            cfsetospeed(&newtio, B9600);
            break;
        case 19200:
            cfsetispeed(&newtio, B19200);
            cfsetospeed(&newtio, B19200);
            break;
        case 38400:
            cfsetispeed(&newtio, B38400);
            cfsetospeed(&newtio, B38400);
            break;
        case 57600:
            cfsetispeed(&newtio, B57600);
            cfsetospeed(&newtio, B57600);
            break;
        case 115200:
            cfsetispeed(&newtio, B115200);
            cfsetospeed(&newtio, B115200);
            break;
        case 230400:
            cfsetispeed(&newtio, B230400);
            cfsetospeed(&newtio, B230400);
            break;
        default:
            printf("\tSorry, Unsupported baud rate, set default 9600!\n\n");
            cfsetispeed(&newtio, B9600);
            cfsetospeed(&newtio, B9600);
            break;
    }

    // 设置read读取最小字节数和超时时间
    newtio.c_cc[VTIME] = 1; // 读取一个字符等待1*(1/10)s
    newtio.c_cc[VMIN] = 1; // 读取字符的最少数为1

    tcflush(fd, TCIFLUSH); //清空缓冲区
    if (tcsetattr(fd, TCSANOW, &newtio) != 0) //激活新设置
    {
        perror("SetupSerial 3");
        return -1;
    }

    printf("Serial set done!\n");
    return 0;
}

/**@brief 串口读取函数
 * @param[in]  fd          打开的串口文件句柄
 * @param[in]  *rcv_buf    接收缓存指针
 * @param[in]  data_len     要读取数据长度
 * @param[in]  timeout      接收等待超时时间, 单位ms
 * @return     返回设置结果
 * - >0       设置成功
 * - 其他      读取超时或错误
 */
int UART_Recv(int fd, char *rcv_buf, int data_len, int timeout)
{
    int len, fs_sel;
    fd_set fs_read;
    struct timeval time;

    time.tv_sec = timeout / 1000; //set the rcv wait time
    time.tv_usec = timeout % 1000 * 1000; //1000000us = 0.1s

```

```
FD_ZERO(&fs_read); //每次循环都要清空集合, 否则不能检测描述符变化
FD_SET(fd, &fs_read); //添加描述符

// 超时等待读变化, >0:就绪描述字的正数目, -1:出错, 0 :超时
fs_sel = select(fd + 1, &fs_read, NULL, NULL, &time);
// printf("fs_sel = %d\n", fs_sel);
if (fs_sel)
{
    len = read(fd, rcv_buf, data_len);
    return len;
}
else
{
    printf("Sorry,I am wrong!");
    return -1;
}
}

/**@brief 串口发送函数
 * @param[in] fd 打开的串口文件句柄
 * @param[in] *send_buf 发送数据指针
 * @param[in] data_len 发送数据长度
 * @return 返回结果
 * ~ data_len 成功
 * ~ -1 失败
 */
int UART_Send(int fd, char *send_buf, int data_len)
{
    ssize_t ret = 0;

    ret = write(fd, send_buf, data_len);
    if (ret == data_len)
    {
        printf("send data is %s\n", send_buf);
        return ret;
    }
    else
    {
        printf("write device error\n");
        fflush(fd, TCIOFLUSH);
        return -1;
    }
}

/**@fn main
 * @brief main入口函数
 */
int main (int argc, char *argv[])
{
    int fdSerial;

    // 打开串口设备
    fdSerial = open(DEV_NAME, O_RDWR | O_NOCTTY | O_NDELAY);
    if (fdSerial < 0)
    {
        perror(DEV_NAME);
        return -1;
    }
    // 设置串口阻塞, 0:阻塞, FNDELAY:非阻塞
    if (fcntl(fdSerial, F_SETFL, 0) < 0) //阻塞, 即使前面在open串口设备时设置的是非阻塞的
    {
        printf("fcntl failed!\n");
    }
    else
    {
        printf("fcntl=%d\n", fcntl(fdSerial, F_SETFL, 0));
    }
    if (isatty(fdSerial) == 0)
    {
        printf("standard input is not a terminal device\n");
        close(fdSerial);
        return -1;
    }
    else
    {
        printf("is a tty success!\n");
    }
    printf("fd-open=%d\n", fdSerial);

    // 设置串口参数
    if (setOpt(fdSerial, 115200, 8, 'N', 1) == -1) //设置8位数据位, 1位停止位, 无校验
    {
        fprintf(stderr, "Set opt Error\n");
        close(fdSerial);
        exit(1);
    }

    fflush(fdSerial, TCIOFLUSH); //清除串口缓存
    fcntl(fdSerial, F_SETFL, 0); //串口阻塞

    char rcv_buf[100];
    int len;

    while(1) //循环读取数据
    {
        len = UART_Recv(fdSerial, rcv_buf, 99, 10000);
        if (len > 0)
        {
            rcv_buf[len] = '\0';
            printf("receive data is %s\n", rcv_buf);
            printf("len = %d\n", len);
            UART_Send(fdSerial, rcv_buf, len);
        }
        else
        {
            printf("cannot receive data\n");
        }
        usleep(100000); //休眠100ms
    }
}
```

编译之后，测试结果如下：

4. RS485编程实例

首先RS485和RS232在应用层面的编程是没有区别的。

RS232用两根线实现全双工，两根线各做各自的，互不影响，可以同时进行；RS485虽然可以用四根线实现全双工，但是实际应用中比较少见，更常见的是只用两根线实现半双工，这样一来，就涉及到“收状态”和“发状态”的切换，这一切又涉及两种情况：

- 1、驱动程序中已经含有对半双工情况下的接受切换，驱动程序会根据你读或写的动作，自动进行切换。这种情况下，RS485的编程就与RS232完全没有区别。
 - 2、驱动程序不带自动切换，此时，为了完成切换，必须使用额外的GPIO连接RS485收发模块的接受使能端，在接受、发送数据之前，首先对使能端置位，使之处于正确的“接收”或“发送”状态。
- 其中第二种操作较为简单，可以基于上面例程增加一个IO控制实现，在此不做赘述（对于不了解接触底层的可以使用这种方式）。

实现方式：

设备树添加 RS485设备

```
uart3 {
    pinctrl-names = "default";
    pinctrl-0 = <&pinctrl_uart3>;
    fsl,rs485-gpio-txen = <&gpio1 18 GPIO_ACTIVE_HIGH>;
    linux,rs485-enabled-at-boot-time;
    status = "okay";
};
```

应用代码如下：

```
/* 包含的头文件 */
#include <stdio.h> //标准输入输出,如printf,scanf以及文件操作
#include <stdlib.h> //标准库头文件,定义了五种类型,一些宏和通用工具函数
#include <unistd.h> //定义 read write close lseek 等Unix标准函数
#include <sys/types.h> //定义数据类型,如 ssize_t off_t 等
#include <sys/stat.h> //文件状态
#include <fcntl.h> //文件控制定义
#include <termios.h> //终端I/O
#include <linux/ioctl.h>
#include <asm-generic/ioctls.h> //tty ioctl numbers,TIOCGRS485 + TIOCSR485 ioctl 定义
#include <linux/serial.h> //serial驱动相关
#include <errno.h> //与全局变量 errno 相关的定义
#include <getopt.h> //处理命令行参数
#include <string.h> //字符串操作
#include <time.h> //时间
#include <sys/select.h> //select函数

.....

int rs485_enable(const int fd, const RS485_ENABLE_t enable)
{
    struct serial_rs485 rs485conf;
    int res;

    // 获取设备的485配置
    res = ioctl(fd, TIOCGRS485, &rs485conf);
    if (res < 0)
    {
        perror("Ioctl error on getting 485 configure:");
        close(fd);
        return res;
    }

    // 设置485模式的使能/禁止
    if (enable)
    {
        // 使能485模式
        rs485conf.flags |= SER_RS485_ENABLED;
    }
    else
    {
        // 关闭485模式
        rs485conf.flags &= ~(SER_RS485_ENABLED);
    }

    rs485conf.delay_rts_before_send = 0x00000004;

    // 将485配置设置到设备中
    res = ioctl(fd, TIOCSR485, &rs485conf);
    if (res < 0) {
        perror("Ioctl error on setting 485 configure:");
        close(fd);
    }

    return res;
}

int main (int argc, char *argv[])
{
    // 打开串口设备
    .....

    // 使能485功能
    rs485_enable(fd, ENABLE);
    .....
}
```

标签: [Linux](#) [设备树](#) [串口](#)

好文置顶

关注我

收藏该文

6

0



silencehuan

关注 - 0

粉丝 - 53

加关注

« 上一篇: [ubuntu下使用minicom](#)
» 下一篇: [rtl8188eu 驱动移植](#)

0


推荐

0

反对

posted @ 2019-06-28 15:46 silencehuan 阅读(29732) 评论(1) 编辑 收藏 举报

[刷新评论](#) [刷新页面](#) [返回顶部](#)

 登录后才能查看或发表评论，立即 [登录](#) 或者 [逛逛](#) [博客园首页](#)

编辑推荐：

- 记一次 .NET 某上市工业智造 CPU+内存+挂死 三高分析
- 深入 xLua 实现原理之 C# 如何调用 Lua
- 记一次 k8s pod 频繁重启的优化之旅
- .Net Core with 微服务：分布式事务 - 可靠消息最终一致性
- 通过 Wireshark 抓包分析谈谈 DNS 域名解析的那些事儿

最新新闻：

- 微软面向Beta和Release Preview频道推出全新Photos应用程序 (2021-09-27 08:22)
- 谷歌云平台下调抽成比例，传从20%降至3% (2021-09-27 08:17)
- 900年前中国古人看到的超新星原来是怎么形成的 (2021-09-27 08:11)
- iPhone 13换屏更难！用户自行维修或致Face ID失效 (2021-09-27 08:07)
- 天舟三号也做了新冠核酸检测，确保不引入任何污染 (2021-09-27 08:00)
- » 更多新闻...