# 串口驱动技术

## 一、确定任务

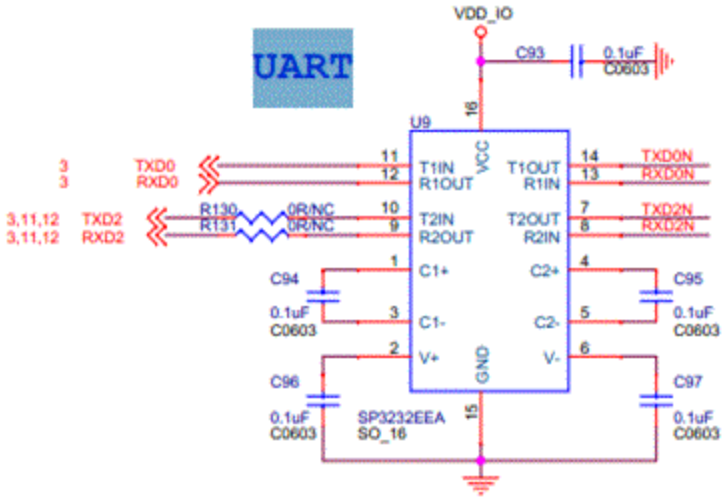
1、 掌握基本的字符设备的驱动程序设计。
2、 掌握基本的文件操作。
3、掌握在操作系统下串口的收发处理过程。

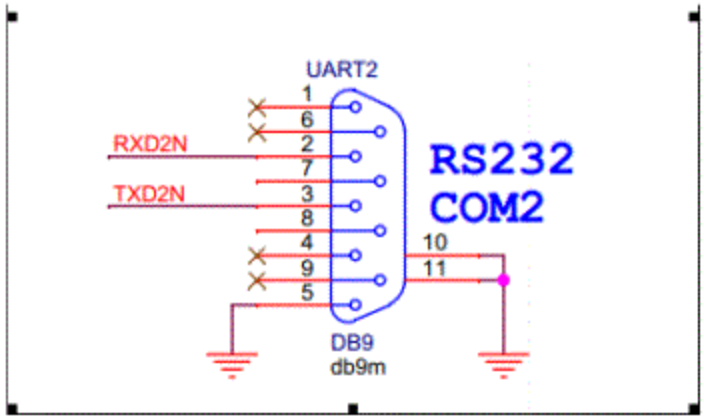
## 二、学习内容

1、 阅读 S5PV210 数据手册，了解串口是原理和使用方法。
2、 编写读写串口的应用程序。
3、 编写 makefile 文件。
4、下载并调试读写串口的应用程序。

## 三、基础知识

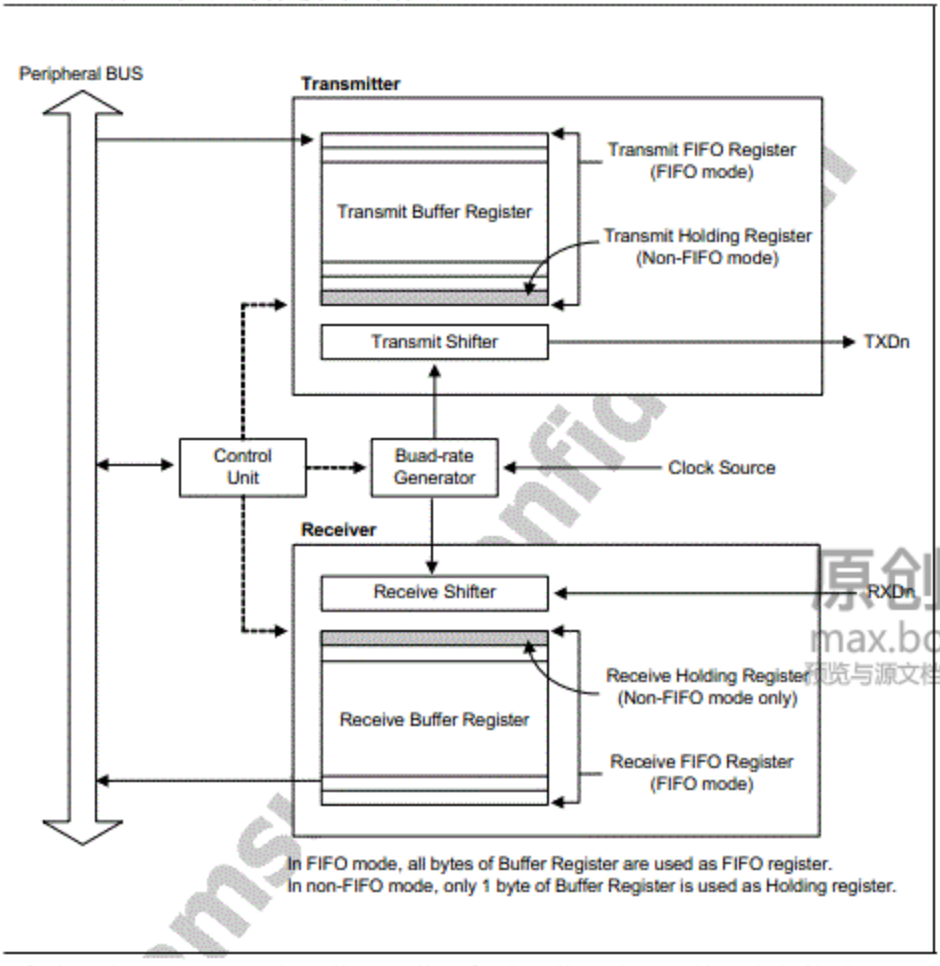
a) 原理图：






(3) 寄存器简介：

a) 系统框架图：UART0 有 256 字节用于 FIFO 模式、UART1 有 64 字节用于 FIFO 模式，UAR2 和 UZRT3 有 16 字节用于 FIFO 模式。在 FIFO 模式下，总共有所有的字节用于 FIFO 寄存器；在非 FIFO 模式下，只有一字节用作缓冲寄存器。






ULCONn 寄存器主要用于设置发送数据位数、起始位数、停止位数、奇偶校验位。

| ULCONn | Bit | Description | Initial State |
|---|---|---|---|
| Reserved | [31:7] | Reserved | 0 |
| Infrared Mode | [6] | Determines whether to use the Infrared mode.<br>0 = Normal mode operation<br>1 = Infrared Tx/Rx mode | 0 |
| Parity Mode | [5:3] | Specifies the type of parity generation to be performed and checking during UART transmit and receive operation.<br>0xx = No parity<br>100 = Odd parity<br>101 = Even parity<br>110 = Parity forced/ checked as 1<br>111 = Parity forced/ checked as 0 | 000 |
| Number of Stop Bit | [2] | Specifies how many stop bits are used to signal end-of-frame signal.<br>0 = One stop bit per frame<br>1 = Two stop bit per frame | 0 |
| Word Length | [1:0] | Indicates the number of data bits to be transmitted or received per frame.<br>00 = 5-bit<br>01 = 6-bit<br>10 = 7-bit<br>11 = 8-bit | 00 |

UCONn 控制寄存器主要用于设置传输和接收模式的选择、分频值的配置、信号和中断的使能。

| UCONn | Bit | Description | Initial State |
|---|---|---|---|
| Reserved | [31:21] | Reserved | 000 |
| Tx DMA Burst Size | [20] | Tx DMA Burst Size<br>0 = 1 byte (Single)<br>1 = 4 bytes | 0 |
| Reserved | [19:17] | Reserved | 000 |
| Rx DMA Burst Size | [16] | Rx DMA Burst Size<br>0 = 1 byte (Single)<br>1 = 4 bytes | 0 |
| Reserved | [15:11] | Reserved | 0000 |
| Clock Selection | [10] | Selects PCLK or SCLK_UART (from Clock Controller) clock for the UART baud rate.<br>0 = PCLK: DIV_VAL1) = (PCLK / (bps x 16) ) –1<br>1 = SCLK_UART: DIV_VAL1) = (SCLK_UART / (bps x 16) ) –1 | 00 |
| Tx Interrupt Type | [9] | Interrupt request type.(2)<br>0 = Pulse (Interrupt is requested when the Tx buffer is empty in the Non-FIFO mode or when it reaches Tx FIFO Trigger Level in the FIFO mode.)<br>1 = Level (Interrupt is requested when Tx buffer is empty in the Non-FIFO mode or when it reaches Tx FIFO Trigger Level in the FIFO mode.) | 0 |
| Rx Interrupt Type | [8] | Interrupt request type. (2)<br>0 = Pulse (Interrupt is requested when instant Rx buffer receives data in the Non-FIFO mode or when it reaches Rx FIFO Trigger Level in the FIFO mode.)<br>1 = Level (Interrupt is requested when Rx buffer is receiving data in the Non-FIFO mode or when it reaches Rx FIFO Trigger Level in the FIFO mode.) | 0 |
| Rx Time Out Enable | [7] | Enables/ Disables Rx time-out interrupts if UART FIFO is enabled. The interrupt is a receive interrupt.<br>0 = Disables<br>1 = Enables | 0 |
| Rx Error Status Interrupt Enable | [6] | Enables the UART to generate an interrupt upon an exception, such as a break, frame error, parity error, or overrun error during a receive operation.<br>0 = Does not generate receive error status interrupt.<br>1 = Generates receive error status interrupt. | 0 |

| UFCONn | Bit | Description | Initial State |
|---|---|---|---|
| Reserved | [31:11] | Reserved | 0 |
| Tx FIFO Trigger Level | [10:8] | Determines the trigger level of Tx FIFO. If data count of Tx FIFO is less than or equal to the trigger level, Tx interrupt occurs.<br><br>[Channel 0]<br>000 = 0 byte 001 = 32 bytes<br>010 = 64 bytes 011 = 96 bytes<br>100 = 128 bytes 101 = 160 bytes<br>110 = 192 bytes 111 = 224 bytes<br><br>[Channel 1]<br>000 = 0 byte 001 = 8 bytes<br>010 = 16 bytes 011 = 24 bytes<br>100 = 32 bytes 101 = 40 bytes<br>110 = 48 bytes 111 = 56 bytes<br><br>[Channel 2, 3]<br>000 = 0 byte 001 = 2 bytes<br>010 = 4 bytes 011 = 6 bytes<br>100 = 8 bytes 101 = 10 bytes<br>110 = 12 bytes 111 = 14 bytes | 000 |
| Reserved | [7] | Reserved | 0 |
| Rx FIFO Trigger Level | [6:4] | Determines the trigger level of Rx FIFO. If data count of Rx FIFO is more than or equal to the trigger level, Rx interrupt occurs.<br><br>[Channel 0]<br>000 = 32 byte 001 = 64 bytes<br>010 = 96 bytes 011 = 128 bytes<br>100 = 160 bytes 101 = 192 bytes<br>110 = 224 bytes 111 = 256 bytes<br><br>[Channel 1]<br>000 = 8 byte 001 = 16 bytes<br>010 = 24 bytes 011 = 32 bytes<br>100 = 40 bytes 101 = 48 bytes<br>110 = 56 bytes 111 = 64 bytes | 000 |

UFCONn 寄存器主要用于发送和接收缓冲区大小的设置、FIFO 的使能。

| UCONn | Bit | Description | Initial State |
|---|---|---|---|
| Loop-back Mode | [5] | Setting loop-back bit to 1 trigger the UART to enter the loop-back mode. This mode is provided for test purposes only.<br>0 = Normal operation<br>1 = Loop-back mode | 0 |
| Send Break Signal | [4] | Setting this bit trigger the UART to send a break during 1 frame time. This bit is automatically cleared after sending the break signal.<br>0 = Normal transmit<br>1 = Sends the break signal | 0 |
| Transmit Mode | [3:2] | Determines which function is able to write Tx data to the UART transmit buffer register.<br>00 = Disables<br>01 = Interrupt request or polling mode<br>10 = DMA mode<br>11 = Reserved | 00 |
| Receive Mode | [1:0] | Determines which function is able to read data from UART receive buffer register.<br>00 = Disables<br>01 = Interrupt request or polling mode<br>10 = DMA mode<br>11 = Reserved | 00 |

| UFCONn | Bit | Description | Initial State |
|---|---|---|---|
| | | [Channel 2, 3]<br>000 = 2 byte 001 = 4 bytes<br>010 = 6 bytes 011 = 8 bytes<br>100 = 10 bytes 101 = 12 bytes<br>110 = 14 bytes 111 = 16 bytes | |
| Reserved | [3] | - | 0 |
| Tx FIFO Reset | [2] | Auto-clears after resetting FIFO<br>0 = Normal<br>1 = Tx FIFO reset | 0 |
| Rx FIFO Reset | [1] | Auto-clears after resetting FIFO<br>0 = Normal<br>1 = Rx FIFO reset | 0 |
| FIFO Enable | [0] | 0 = Disables<br>1 = Enables | 0 |

UTRSTSTn 寄存器主用用于保存串口的状态信息。

| UTRSTATn | Bit | Description | Initial State |
|---|---|---|---|
| Reserved | [31:3] | Reserved | 0 |
| Transmitter empty | [2] | This bit is automatically set to 1 if the transmit buffer register has no valid data to transmit, and the transmit shift register is empty.<br>0 = Not empty<br>1 = Transmitter (which includes transmit buffer and shifter register) empty | 1 |
| Transmit buffer empty | [1] | This bit is automatically set to 1 if transmit buffer register is empty.<br>0 = Buffer register is not empty<br>1 = Buffer register is empty (In Non-FIFO mode, Interrupt or DMA is requested.<br>In FIFO mode, Interrupt or DMA is requested, if Tx FIFO Trigger Level is set to 00 (Empty))<br>If UART uses FIFO, check Tx FIFO Count bits and Tx FIFO Full bit in UFSTAT register instead of this bit. | 1 |
| Receive buffer data ready | [0] | This bit is automatically set to 1 if receive buffer register contains valid data, received over the RXDn port.<br>0 = Buffer register is empty<br>1 = Buffer register has a received data<br>(In Non-FIFO mode, Interrupt or DMA is requested)<br>If UART uses the FIFO, check Rx FIFO Count bits and Rx FIFO Full bit in UFSTAT register instead of this bit. | 0 |

UERSTATn 寄存器主要用于保存串口的错误信息。

| UERSTATn | Bit | Description | Initial State |
|---|---|---|---|
| Reserved | [31:4] | Reserved | 0 |
| Break Detect | [3] | This bit is automatically set to 1 to indicate that a break signal has been received.<br>0 = No break signal is received<br>1 = Break signal is received (Interrupt is requested.) | 0 |
| Frame Error | [2] | This bit is automatically set to 1 if a frame error occurs during the receive operation.<br>0 = No frame error occurs during the receive operation<br>1 = Frame error occurs (Interrupt is requested.) during the receive operation | 0 |
| Parity Error | [1] | This bit is automatically set to 1 if a parity error occurs during the receive operation.<br>0 = No parity error occurs during receive the receive operation<br>1 = Parity error occurs (Interrupt is requested.) the receive operation | 0 |
| Overrun Error | [0] | This bit is automatically set to 1 automatically if an overrun error occurs during the receive operation.<br>0 = No overrun error occurs during the receive operation<br>1 = Overrun error occurs (Interrupt is requested.) during the receive operation | 0 |



UTXHn 和 URXHn 寄存器分别为发送寄存器和接收寄存器。

| UTXHn | Bit | Description | Initial State |
|---|---|---|---|
| Reserved | [31:8] | Reserved | - |
| UTXHn | [7:0] | Transmit data for UARTn | - |

| URXHn | Bit | Description | Initial State |
|---|---|---|---|
| Reserved | [31:8] | Reserved | 0 |
| URXHn | [7:0] | Receive data for UARTn | 0x00 |

## 四、代码设计

1、 RS-232 原理，请读者自行查找相关资料。

2、 测试程序（详细程序请查看附件）：

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <errno.h>
#include <string.h>
#include <termios.h>

#define max(x, y) (((int)(x) > (int)(y)) ? x : y)
#define TRUE  1
#define FALSE 0
```

```
static char *serial_name[]={"/dev/s3c2410_serial3"};
static struct timeval timeout={3,0};
static struct termios oldtio_r,oldtio_w;

int speed_arr[] = {B115200,B57600, B38400, B19200,
                B9600, B4800, B2400, B1200, B300,
                B38400, B19200, B9600, B4800,
                B2400, B1200, B300, };
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]){
            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;
            }
        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){
        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 */
            options.c_iflag &= ~(ICRNL|IGNCR);
            options.c_lflag &= ~(ICANON );
            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  超时时间
        options.c_cc[VMIN] = 0;   //最小读取位

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

int OpenDev(char *Dev)
{
    int fd = open( Dev, O_RDWR | O_NOCTTY | O_NDELAY);         //| O_NOCTTY
| O_NDELAY
    if (fd <0){
            perror("Can't Open Serial Port");
            exit(EXIT_FAILURE);
    }

    return fd;
}


int serial_write(int fd)
{
    char buf[]={"this is uart test\n"};
```

```
    int ret ;

//  memset(buf, 0 , sizeof(buf));
//  printf("write serial:\n");
//  fgets(buf , 256, stdin);
//  if(!strcmp(buf,"quit")){  //输入 quit 退出测试程序
//      printf("good bye\n");
//      return FALSE;
//  }
    if(tcflush(fd,TCIOFLUSH)<0){   //清空串口缓冲区
        fprintf(stderr, "tcflush error\n");
    }
    ret = write(fd , buf , strlen(buf));  //写串口
    if(ret <0){
        perror("write serial");
        exit(EXIT_FAILURE);
    }
    //write(fd , "\n\0",2);
    //printf("write uart\n");

    return  TRUE;
}

void serial_read(int fd )
{
    int ret ;
    int len= 0;
    char buf[256];
    memset(buf , 0 , 256);

    ret = read(fd , buf , 256); //读取串口
    if(ret<0){
        perror("read");
        exit(EXIT_FAILURE);
    }
#if 0
    while(0){
        ret = read(fd , buf , 256);

        if(ret ==-1 && errno == EINTR){
            continue;
        }
        else if(ret <0){
            perror("read serial");
```

```
            exit(EXIT_FAILURE);
        }
        else if(ret ==0){
            break;
        }
        else{
            printf("receive %s \n",buf);
        }
    }
#endif
    //printf("%d\n",ret);

    printf("read serial: %s \n",buf);
    if(tcflush(fd,TCIOFLUSH)<0){  //清空串口缓冲区
        fprintf(stderr,"flush error\n");
    }
    return ;
}

void serial_config(int fd)
{
    set_speed(fd,115200);  //设置波特率 baud rate
    if (!set_Parity(fd,8,1,'N'))     {
            printf("Set Parity Error\n");
            exit(EXIT_FAILURE);
    }
    return ;
}

int main(void)
{
    int fd_r , fd_w;
    int i ;
    int ret ;
    char ch ;
    fd_set readfds;

    fd_w = OpenDev(serial_name[0]);  //打开 串口
    tcgetattr(fd_w,&oldtio_w);
    serial_config(fd_w);


//  fcntl(fd_r, F_SETFL, F_UNLCK); // Non-blocking read
```