SP2538 数据手册

1. 概述

SP2538 系采用低功耗 CMOS 工艺设计生产的专用串行口（RS232）扩展芯片，它可轻松将任意单片机（如：89C51）或 DSP 等现有的 RS232 串行口扩展成 5 个全新的全双工 RS232 串行口（所有串行口可同时独立接收和发送数据）。

SP2538 主要为解决大多数 8 位、16 位、32 位嵌入式单片机及 DSP 等的串行口太少（大多数只有一个 RS232 口）而特别优化设计的低功耗专用串行口扩展芯片。与其他多串口单片机或串口扩展解决方案相比，采用 SP2538 实现的多串口单片机解决方案具有：不需要购买和学习新的开发工具（编译器、仿真器、编程器）；使用极简单上电后不需任何初始化；参考免费提供的应用流程图和经过测试可直接调用的底层驱动函数包，5 分钟即可完全掌握芯片的使用方法和编程技巧；大大缩短开发周期、降低开发成本和生产成本以及大幅提高产品稳定性。

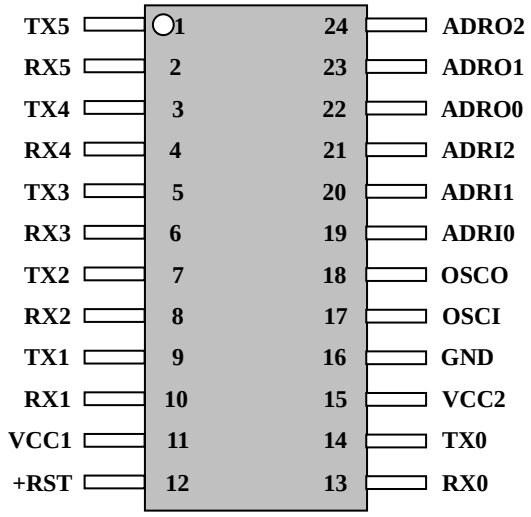
2. 性能特点

- 工作电压宽：3.3V~5.5V。
- 工作速率宽：5 个子串口可产生 2400bps~9600bps 之间的任意波特率。
- 工作电流小：典型电流 4.6mA@子串口速率为 9600bps、VCC=3.3V。
- 全双工工作：母串口和所有子串口都支持全双工 UART 传输模式。
- 资源占用少：除占用上位机原有串行口中断外不占用其他任何中断资源。
- 使用极简单：一般情况下仅需 1~2 条常用指令即可（总共只有 6 条指令）。
- 有节电模式：进入节电模式后典型静态电流约 8μA 。
- 可自动唤醒：上位机发送数据可自动唤醒（注：这与 SP2328/2338 有所不同）。
- 输出误差小：所有子串口的输出波特率误差小于 0.08% 。
- 误码率极低：低于 10^{-9}@所有串行口数据输入波特率误差≤±2%时。
- 接收范围宽：波特率误差小于 2.5%时子串口即可完全正确接收；母串口接收范围更宽，可自适应 56000bps 和 57600bps 两种标准波特率@ F_{osc_in}=20.0 MHz。
- 上位机监控：具有上电复位和看门狗监控输出，适用于没有看门狗或需要多重监控的高可靠高安全上位机程序监控系统；采用专门的喂狗指令可靠性和稳定性远优于常用的“单控制线”喂狗方式；有专门的关闭看门狗指令和启动看门狗指令，用于某些需要禁止看门狗工作的特殊应用。

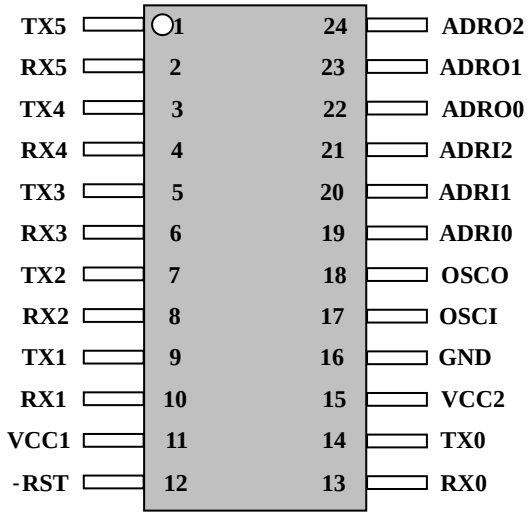
3. 应用领域

- 数据采集 ；
- 工业控制 ；
- 仪器仪表 ；
- 智能家电 ；
- 医疗设备 ；
- 税控加油机 ；
- 商业 POS 机 ；
- 家庭安防控制 ；
- 车辆监控、调度 ；
- 工业 MODEM 阵列 ；
- GSP 卫星定位、导航 ；
- 有线及无线数据传输 ；
- 基于 PC 机的多串口卡 ；
- 水、电、气表抄表系统 ；
- 室外多媒体电子广告牌 ；
- 其他对通信稳定性、成本和开发周期敏感的各种应用 ；
- 需要同时进行多路光电隔离传输的系统、设备（一个 RS232 光电隔离器可以同时实现母串口和 5 路 UART 子串口的电气隔离；

4. 引脚说明



SP2538XX **H** － X



SP2538XX **L** - X

图 1

表 1 （管脚说明）

管脚名称	管脚编号	管脚类型	管　脚　描　述
TX5	1	Output	串口 5 数据发送（连接上位机 RX 口）
RX5	2	Input	串口 5 数据接收（连接上位机 TX 口）
TX4	3	Output	串口 4 数据发送（连接下位机 RX 口）
RX4	4	Input	串口 4 数据接收（连接下位机 TX 口）
TX3	5	Output	串口 3 数据发送（连接下位机 RX 口）
RX3	6	Input	串口 3 数据接收（连接下位机 TX 口）
TX2	7	Output	串口 2 数据发送（连接下位机 RX 口）
RX2	8	Input	串口 2 数据接收（连接下位机 TX 口）
TX1	9	Output	串口 1 数据发送（连接下位机 RX 口）
RX1	10	Input	串口 1 数据接收（连接下位机 TX 口）
VCC1	11	---	电源 1（逻辑电路电源）
+RST	12	Output	复位控制输出（适用于高电平复位的 MCU）
-RST	12	Output	复位控制输出（适用于低电平复位的 MCU）
RX0	13	Input	串口 0 数据接收（连接下位机 TX 口）
TX0	14	Output	串口 0 数据发送（连接下位机 RX 口）
VCC2	15	---	电源 2（时钟电路电源）
GND	16	---	电源（公共）地
OSCI	17	Input	时钟输入（用于波特率发生器等）
OSCO	18	Output	时钟输出
ADRI0	19	Input	母串口（RX5）数据接收地址 0
ADRI1	20	Input	母串口（RX5）数据接收地址 1
ADRI2	21	Input	母串口（RX5）数据接收地址 2
ADRO0	22	Output	母串口（TX5）数据发送地址 0
ADRO1	23	Output	母串口（TX5）数据发送地址 1
ADRO2	24	Output	母串口（TX5）数据发送地址 2

5. 设计选型

SP25 X 8 XX X — X

A B C D E

表 2 （命名规则）

代码	内　　容
A	1 个 RS232 串行口扩展为 5 个 RS232 串行口系列
B	3：子串口最高工作速率为：9600bps 4：子串口最高工作速率为：19200bps
C	DP：双列直插封装（DIP） SO：双列贴片封装（SOIC）
D	H：复位时输出高电平（适用于MCS-51等需要高电平复位的MCU） L：复位时输出低电平（适用于其他需要低电平复位的单片机MCU）
E	C：商业级（0℃~70℃） I：工业级（-40℃~85℃）

6. 应用说明

- 母串口和所有子串口内部都有独立的数据发送缓冲存储器（FIFO Buffer）和接收缓冲存储器（FIFO Buffer），并且所有 RS232 串行口都支持全双工异步传输模式即所有串行口都可以同时独立接收和发送数据且不会丢失任何数据。
- 母串口波特率 K_1=2880 * F_{osc_in} （**注：**F_{osc_in} 的计算单位是“MHz”，且 F_{osc_in} ≤20.0MHz， 在 SP2538 输入时钟 F_{osc_in} = 20.0 MHz 时，母串口可自适应上位机的 56000bps 和 57600bps 两种标准波特率输入，即：F_{osc_in} = 20.0 MHz 时，上位机的 RS232 波特率可以设置成 56000bps 或 57600bps）。
- 子串口波特率 K_2= 480 * F_{osc_in} 。
- 母串口和所有子串口都是 TTL 电平接口，可直接匹配其他单片机或 TTL 数字电路，如需连接 PC 机则必须增加电平转换芯片，如：MAX202，MAX232 等。
- SP2538 系列具有内置的上电复位电路和可关闭的看门狗监控电路，用于监控上位机程序是否正常运行，同时简化上位机复位电路设计。
- 上位机写命令字“0x10”可实现喂狗（该喂狗方式可靠性和安全性远优于普通“单控制线”方式喂狗）；上位机也可写命令字“0x15”关闭看门狗（初次上电后看门狗处于激活状态），或写命令字“0x20”激活看门狗监控功能。
- 上位机可通过芯片复位指令（命令字为“0x35”），在任何时候让芯片进行指令

复位（T_{reset}＜50uS，且一般情况下用户不必理会该指令）。

- 上位机也可通过芯片睡眠指令（命令字为“0x55”），在任何时候让芯片进入微功耗睡眠模式以降低系统功耗，初次上电后芯片不会自行进入睡眠模式如不考虑系统功耗则不必理会该指令（**注意：**只能由上位机任意发送一个字节数据将其唤醒（Twake_up＜12mS，参见图 5），其他所有子串口不能将其唤醒，这一特性与 SP2328 和 SP2338 系列串行口扩展芯片的自动唤醒方式有所不同）。
- SP2538XX **H**－X 系列的“+RST”脚必须接上拉电阻 R2（此时 R1 不接）；SP2538XX **L**－X 系列“-RST”必须接下拉电阻 R1（此时 R2 不接，参见图 2）。
- 未使用的输入端口 ，如：RX0、RX1、RX2 **…**等必须连接到 VCC，未使用的输出端口，如 **：**TX0、TX1、TX2 **...** 、ADRO0、ADRO1**…**等必须悬空。
- 母串口收、发数据时序：

①上位机接收来自母串口的数据：上位机从母串口接收到一个字节后，立即读取 SP2538 的输出地址（建议采用中断方式接收来自母串口的数据）：ADRO2~ADRO0（编码方式为：8-4-2-1 码），根据输出地址的编码即可判断接收到的数据来自哪个子串口，即：哪个下位机（参见图 7）。

②上位机向母串口发送数据：A、上位机通过 I/O 口写欲发送数据的子串口号即先由上位机的 I/O 口发送欲发送数据的地址：ADRI2~ADRI0（编码方式：8-4-2-1 码）； B、将欲发送的数据由上位机串口发出（参见图 6）。（注意：母串口的波特率是子串口的 6 倍，即上位机在连续向母串口发送 6 个字节的时间内子串口才能发送完一个字节（参见图 8），建议在产品设计过程中参考或直接调用配套的 SP2538 底层驱动函数包，可使你的产品开发周期大幅缩短）；

表 3 （指令一览表）

命令	地址	内　　容
0x00	101	上位机串行口发送延时指令（类似于程序中的“NOP”指令）
0x10	101	喂看门狗指令（看门狗超时周期≥800mS）
0x15	101	禁止看门狗指令（关闭看门狗，此时上位机可停止喂狗）
0x20	101	使能看门狗指令（开启看门狗，上位机必须在超时周期内喂狗）
0x35	101	芯片复位指令（一般情况用户不必理会该指令）
0x55	101	芯片睡眠指令（初次上电后芯片不会自行进入睡眠模式）

7. 参考电路

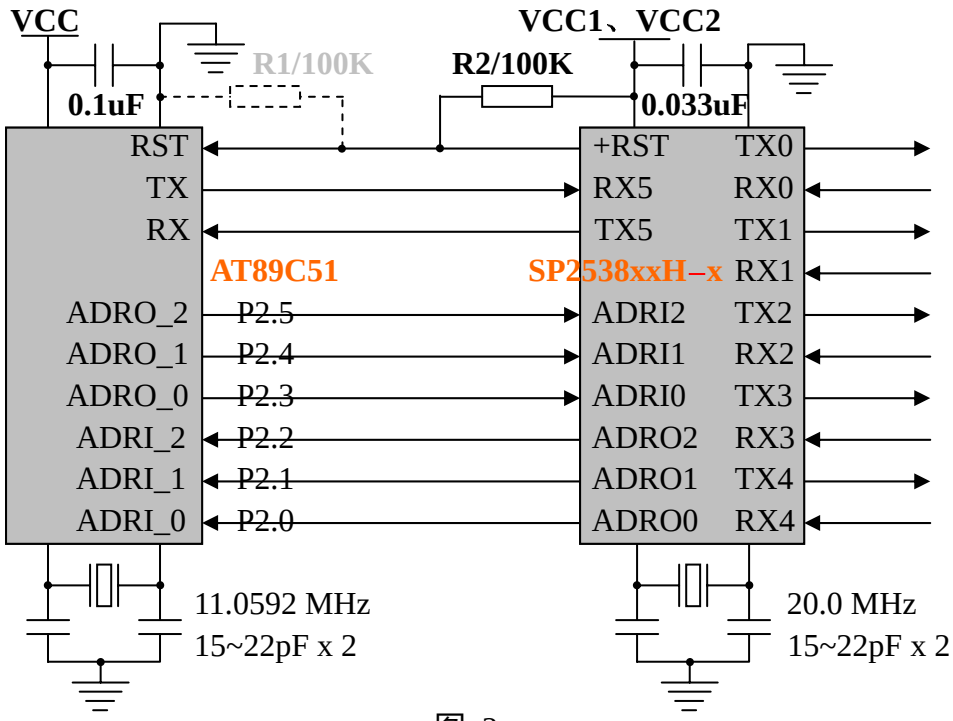
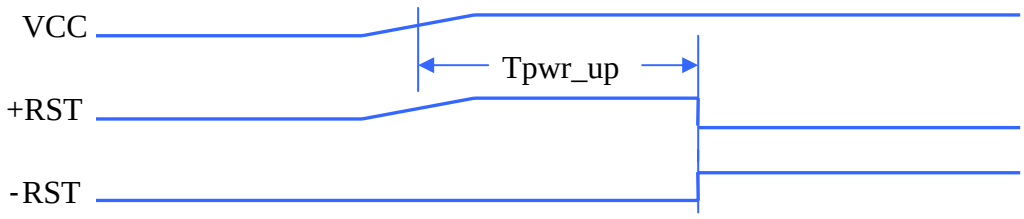


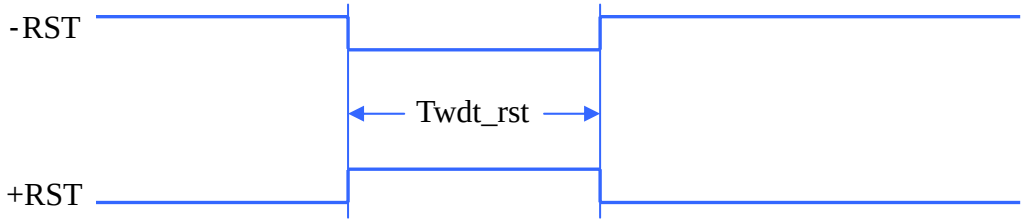
图 2

8. 工作时序



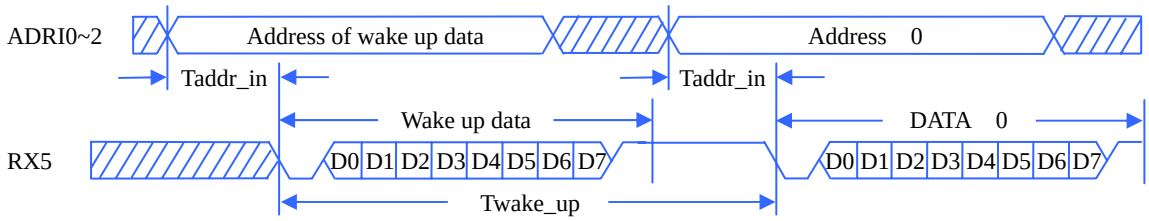
上电复位延时

图 3



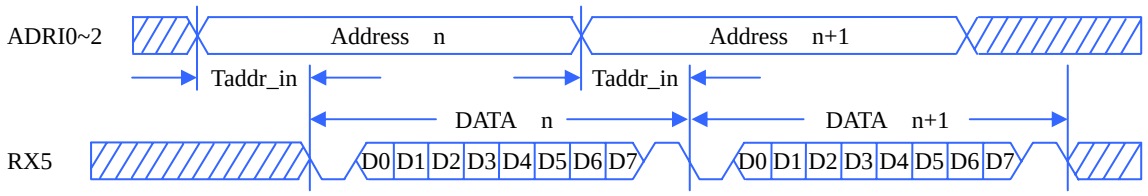
看门狗溢出复位脉冲宽度

图 4



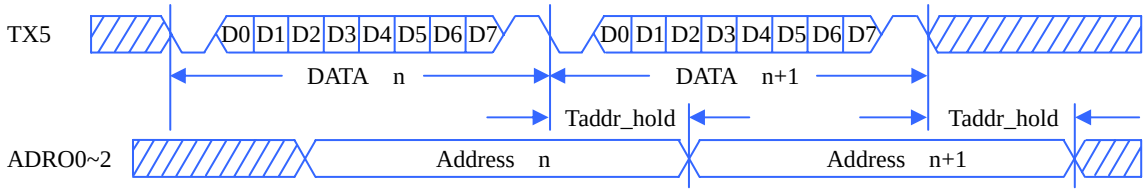
芯片自动唤醒

图 5



母串口（RX5）接收数据

图 6

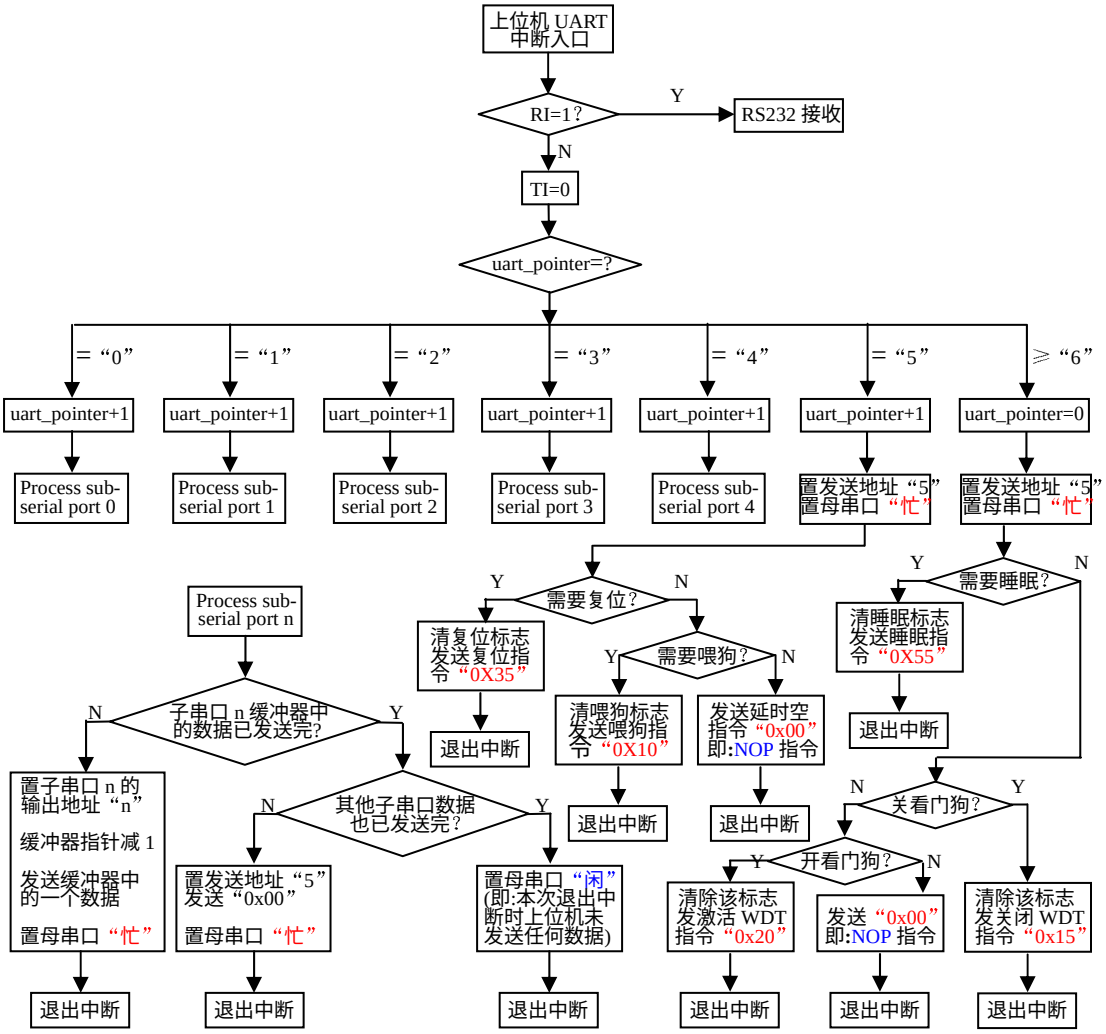


母串口（TX5）发送数据

图 7

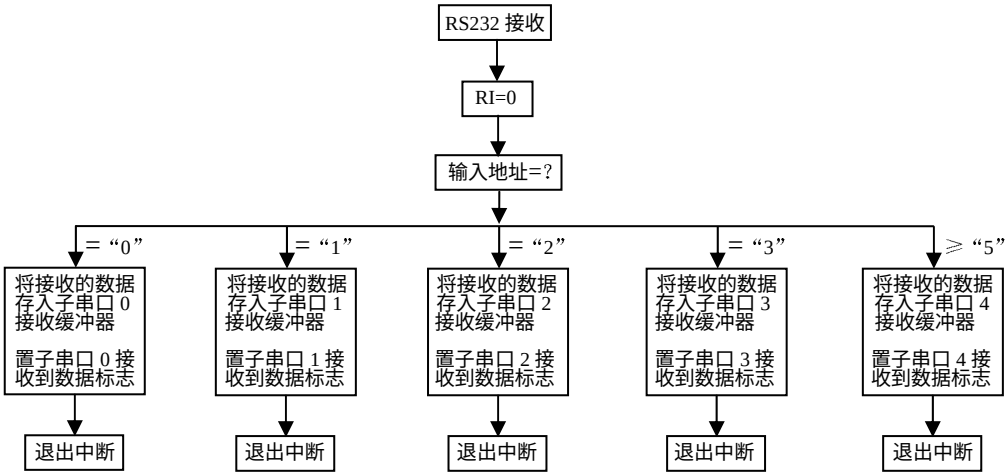
表 4 （操作时限）

内容	说明	最小值	典型值	最大值
Tpwr_up	上电复位延时	150 mS	---	---
Treset	芯片指令复位	---	---	50 uS
Twdt_rst	看门狗溢出复位脉冲宽度	80 mS	---	---
Taddr_in	数据接收地址保持	10 nS	---	---
Twake_up	芯片唤醒延时	---	---	9 mS
Taddr_hold	数据发送地址保持	$(2/F_{osc\_in})$ mS	---	---
Twdt_over	看门狗溢出周期	800 mS	---	---



上位机数据发送流程图

图 8



上位机数据接收流程图

图 9

9. 极限参数

- 环境温度：-55℃ ~125℃。
- 储藏温度：-65℃ ~150℃。
- 最高工作电压：6.0 V。
- 最高时钟频率：30.0 MHz @ 5.5 V。
- 最大功率消耗：0.8 W

10. 直流电气特性

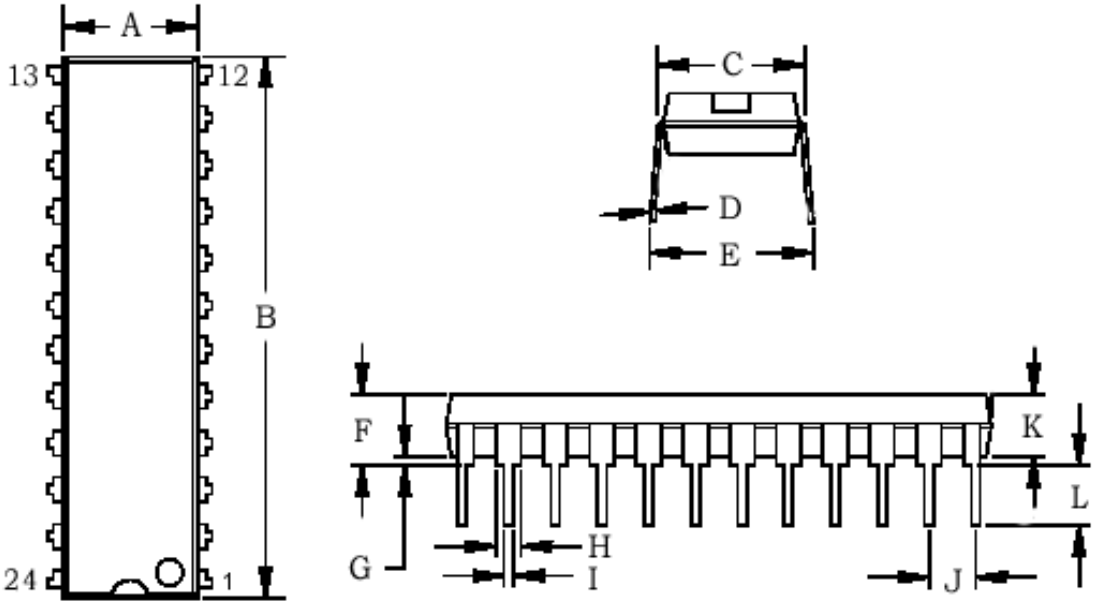
（环境温度：25℃）

表 5 （直流电气特性）

特性	最小值	典型值	最大值	单位	条　　件
工作电压	3.3	—	5.5	V	
RX 口/ADRI 口输入低电平	GND	—	0.8	V	VCC=5.0V
RX 口/ADRI 口输入高电平	2.0	—	VCC	V	VCC=5.0V
输入口漏电流	-1	—	+1	uA	Input pin at VCC or GND
RX 口/ADRI 口输入钳位电流	—	—	12.0	mA	VCC=5.0V
TX 口/ADRO 口输出低电平	GND	—	0.7	V	VCC=5.0V IOL （拉电流）=5.5mA
TX 口/ADRO 口输出高电平	VCC-0.7	—	VCC	V	VCC=5.0V IOH（灌电流）=2.7mA
TX 口/ADRO 口输出拉电流	—	9.0	—	mA	VCC=5.0V
TX 口/ADRO 口输出灌电流	—	7.5	—	mA	VCC=5.0V
OSCI 口输入低电平	GND	—	1.55	V	VCC=5.0V
OSCI 口输入高电平	3.45	—	VCC	V	VCC=5.0V
OSCO 口输出低电平	GND	—	0.7	V	VCC=5.0V IOL （拉电流）=1.0mA
OSCO 口输出高电平	VCC-0.7	—	VCC	V	VCC=5.0V IOH（灌电流）=1.0mA
时钟频率范围	5.0	—	20.0	MHz	3.3V≤VCC≤5.5V
工作电流（正常模式）	—	1.3	3.2	mA	VCC=3.3V　F_{osc_in}=5.0MHz
	—	4.6	9.4	mA	VCC=3.3V　F_{osc_in}=20.0MHz
睡眠电流	—	8.0	—	uA	VCC=3.3V　F_{osc_in}=0MHz

11. 产品封装

- DIP 封装

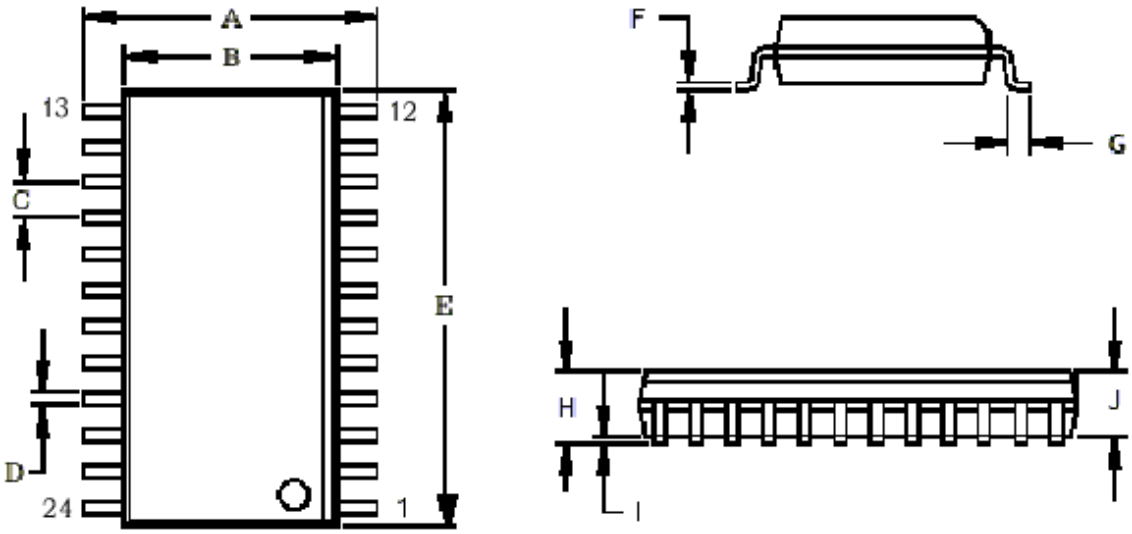


DIP 封装尺寸图
图 10

表 6 （For DIP Package）

Units		MILLIMETERS			INCHES		
Dimension Limits		MIN	TYP	MAX	MIN	TYP	MAX
Molded Package Width	A	6.15	6.40	6.65	0.242	0.252	0.262
Overall Length	B	29.42	29.72	29.92	1.158	1. 170	1. 178
Shoulder to Shoulder Width	C	7.62	7.94	8.26	0.300	0.313	0.325
Lead Thickness	D	0.20	0.25	0.30	0.008	0.010	0.012
Overall Row Spacing	E	7.87	9.40	10.92	0.310	0.370	0.430
Top to Seating Plane	F	3.56	3.94	4.36	0.140	0.155	0.172
Base to Seating Plane	G	0.50			0.020		
Upper Lead Width	H	1.27	1.46	1.57	0.050	0.057	0.062
Lower Lead Width	I	0.38	0.46	0.54	0.015	0.018	0.021
Pitch	J		2.54			0.100	
Molded Package Thickness	K	2.92	3.30	3.68	0.115	0.130	0.145
Tip to Seating Plane	L	3.00	3.25	3.43	0.118	0.128	0.135

- SOIC **封装**



SOIC 封装尺寸图
图 11

表 7 （For SOIC Package）

Units		MILLIMETRES			INCHES		
Dimension Limits		MIN	TYP	MAX	MIN	TYP	MAX
Overall Width	A	9.80	10.20	10.60	0.386	0.402	0.417
Molded Package Width	B	7.30	7.60	7.90	0.287	0.299	0.311
Pitch	C		1.27			0.050	
Lead Width	D	0.30	0.40	0.50	0.012	0.016	0.020
Overall Length	E	15.14	15.34	15.59	0.596	0.604	0.614
Lead Thickness	F	0.23	0.25	0.30	0.009	0.010	0.012
Foot Length	G	0.30	0.50	0.70	0.012	0.020	0.028
Overall Height	H	2.36	2.54	3.10	0.093	0.100	0.122
Stand off	I	0.05	0.20	0.30	0.002	0.008	0.012
Molded Package Thickness	J	2.31	2.34	2.80	0.091	0.092	0.110

```
//*********************************************************************************************//
//        Microsystem_for_SP2538.c source file SP2538 serial communication example for MCS-51 MCU      //
//                                        Author: Chen Wei, 2003-01-16                                 //
//*********************************************************************************************//
#define _DEBUG_SUB_MODULE              // Be used to test sub-serial port 0~4

#include <reg52.h>
#define uchar unsigned char
#define uint    unsigned int

uchar idata uart0_t_buf[8];                    // you can modify the buffer size of sub-serial port 0
uchar idata uart1_t_buf[8];                    // you can modify the buffer size of sub-serial port 1
uchar idata uart2_t_buf[8];                    // you can modify the buffer size of sub-serial port 2
uchar idata uart3_t_buf[8];                    // you can modify the buffer size of sub-serial port 3
uchar idata uart4_t_buf[8];                    // you can modify the buffer size of sub-serial port 4
uchar idata send_buf      [8];                 // you can modify the buffer size of temporary data package

uchar idata uart0_r_buf,uart1_r_buf,uart2_r_buf,uart3_r_buf,uart4_r_buf;
uchar idata uart0_send_num,uart1_send_num,uart2_send_num,uart3_send_num,uart4_send_num;
uchar idata uart_port_num,send_byte_num,uart_pointer;

bit bdata write_success,uart_busy;
bit bdata uart0_receive,uart1_receive,uart2_receive,uart3_receive,uart4_receive;
bit bdata f_clear_wdt,f_disable_wdt,f_enable_wdt,f_reset_chip,f_sleep_chip;

sbit   ADRI_0=P2^0;                    // connect to pin ADRO0 of SP2538
sbit   ADRI_1=P2^1;                    // connect to pin ADRO1 of SP2538
sbit   ADRI_2=P2^2;                    // connect to pin ADRO2 of SP2538
sbit   ADRO_0=P2^3;                    // connect to pin ADRI0 of SP2538
sbit   ADRO_1=P2^4;                    // connect to pin ADRI1 of SP2538
sbit   ADRO_2=P2^5;                    // connect to pin ADRI2 of SP2538

/*********************************************************************************************
  serial port ISR.  (上位机 RS232 中断服务函数,该函数可自动完成 5 个子串口数据的发送和接收)
*********************************************************************************************/
serial () interrupt 4 using 3{                 //  上位机 RS232 接收、发送中断入口
      if(RI){                                  //  判断上位机是发送中断还是接收中断?
            RI=0;                              //  本次上位机 RS232 中断是接收中断
            switch(P2&0x07){                   //  判断接收到的数据来自哪个子串口?
                  case 0:{                     //  接收到的数据来自串口 0
                        uart0_r_buf=SBUF;      //  清空"SBUF"以便接收其他子串口的数据
                        uart0_receive=1;       //  置串口 0 收到数据标志,主程序采用查询该
                        break;                 //  标志的方式处理收到的数据,主程序处理
                  }                            //  完刚收到的数据后软件必须将该标志"清零"
```

```
            case 1:{                    // 接收到的数据来自串口 1
                uart1_r_buf=SBUF;
                uart1_receive=1;
                break;
            }

            case 2:{                    // 接收到的数据来自串口 2
                uart2_r_buf=SBUF;
                uart2_receive=1;
                break;
            }

            case 3:{                    // 接收到的数据来自串口 3
                uart3_r_buf=SBUF;
                uart3_receive=1;
                break;
            }

            case 4:{                    // 接收到的数据来自串口 4
                uart4_r_buf=SBUF;
                uart4_receive=1;
                break;
            }

            default:{                   // 防止程序跑飞
                break;
            }
        }
    }

    else{                               // 上位机发送中断
        TI=0;                           // 必须由软件清除发送中断标志
        switch(uart_pointer){           // 判断轮到该哪个子串口发送数据
            case 0:{                    // 轮到串口 0 发送数据
                uart_pointer++;         // 串口数据发送指针加"1",下一次中断判断串口 1 有无数据发送
                if(uart0_send_num){     // 判断串口 0 缓冲区的数据是否已经完全发送
                    ADRO_0=0;           // 串口 0 缓冲区的数据没有发送完
                    ADRO_1=0;           // 注意：写数据到"SBUF"前必须先置欲发送子串口的地址!!!
                    ADRO_2=0;
                    uart0_send_num--;                       // 下一次发送数据包中的下一个字节数据
                    SBUF=uart0_t_buf[uart0_send_num];       // 从缓冲区发送一个字节到指定的子串口
                    uart_busy=1;                            // 置母串口忙（正在发送数据）标志
                    break;
                }
                else if(uart1_send_num|uart2_send_num|uart3_send_num|uart4_send_num){
```

```
            ADRO_0=1;       // 串口 0 的数据已经发送完,继续判断其他子串口的数据
            ADRO_1=0;       // 是否也已经发送完,如果还没有发送完,则需要发送
            ADRO_2=1;       // "NOP"指令(即发送"0x00")用于延时一个字节时间长度
            SBUF=0x00;      // 注意：发送"NOP"前必须先设置串口地址!!!
            uart_busy=1;    // 置母串口忙（正在发送数据）标志
            break;
        }                   // 如果串口 0~串口 2 的数据包都已经发送完,置母串口"空闲"标志
        else {uart_busy=0;break;} // 主程序通过该标志判断是否要以"TI=1"启动子串口发送数据
}                           // 注：如所有子串口的数据发送都已经完,则不会再产生发送中断

case 1:{                    // 轮到串口 1 发送数据
        uart_pointer++;
        if(uart1_send_num){
            ADRO_0=1;
            ADRO_1=0;
            ADRO_2=0;
            uart1_send_num--;
            SBUF=uart1_t_buf[uart1_send_num];
            uart_busy=1;
            break;
        }
        else if(uart0_send_num|uart2_send_num|uart3_send_num|uart4_send_num){
            ADRO_0=1;
            ADRO_1=0;
            ADRO_2=1;
            SBUF=0x00;
            uart_busy=1;
            break;
        }
        else {uart_busy=0;break;}
}

case 2:{                    // 轮到串口 2 发送数据
        uart_pointer++;
        if(uart2_send_num){
            ADRO_0=0;
            ADRO_1=1;
            ADRO_2=0;
            uart2_send_num--;
            SBUF=uart2_t_buf[uart2_send_num];
            uart_busy=1;
            break;
        }
        else if(uart0_send_num|uart1_send_num|uart3_send_num|uart4_send_num){
            ADRO_0=1;
```

```
            ADRO_1=0;
            ADRO_2=1;
            SBUF=0x00;
            uart_busy=1;
            break;
        }
        else {uart_busy=0;break;}
}

case 3:{                    // 轮到串口 3 发送数据
        uart_pointer++;
        if(uart3_send_num){
            ADRO_0=1;
            ADRO_1=1;
            ADRO_2=0;
            uart3_send_num--;
            SBUF=uart3_t_buf[uart3_send_num];
            uart_busy=1;
            break;
        }
        else if(uart0_send_num|uart1_send_num|uart2_send_num|uart4_send_num){
            ADRO_0=1;
            ADRO_1=0;
            ADRO_2=1;
            SBUF=0x00;
            uart_busy=1;
            break;
        }
        else {uart_busy=0;break;}
}

case 4:{                    // 轮到串口 4 发送数据
        uart_pointer++;
        if(uart4_send_num){
            ADRO_0=0;
            ADRO_1=0;
            ADRO_2=1;
            uart4_send_num--;
            SBUF=uart4_t_buf[uart4_send_num];
            uart_busy=1;
            break;
        }
        else if(uart0_send_num|uart1_send_num|uart2_send_num|uart3_send_num){
            ADRO_0=1;
            ADRO_1=0;
```

```
            ADRO_2=1;
            SBUF=0x00;
            uart_busy=1;
            break;
        }
        else {uart_busy=0;break;}
}

case 5:{
        uart_pointer++;
        ADRO_0=1;
        ADRO_1=0;
        ADRO_2=1;
        uart_busy=1;
        if(f_reset_chip){              // 让 SP2538 进行内部复位
            f_reset_chip=0;
            SBUF=0x35;
            break;
        }
        else if(f_clear_wdt){          // 上位机对 SP2538 进行喂狗
            f_clear_wdt=0;
            SBUF=0x10;
            break;
        }
        else{SBUF=0x00;break;}    // 发送"NOP"用于延时一个字节时间长度
}

default:{
        uart_pointer=0;
        ADRO_0=1;
        ADRO_1=0;
        ADRO_2=1;
        uart_busy=1;
        if(f_sleep_chip){              // 让 SP2538 进入睡眠模式
            f_sleep_chip=0;
            SBUF=0x55;
            break;
        }
        else if(f_disable_wdt){        // 禁止 SP2538 中的看门狗监控电路
            f_disable_wdt=0;
            SBUF=0x15;
            break;
        }
        else if(f_enable_wdt){         // 使能 SP2538 中的看门狗监控电路
            f_enable_wdt=0;
```

```
                    SBUF=0x20;
                    break;
                }
                else{SBUF=0x00;break;}    // 发送"NOP"用于延时一个字节时间长度
            }
        }
    }
}

/*********************************************************************************************
 function:          void uart_send(uchar uart_port_num,uchar send_byte_num)
 input:             uchar uart_port_num - index of sub-serial port
                    uchar send_byte_num - data number will be sent from “1” to “8”
                    write_success=0 (Call the function must before "write_success=0" first )
 output:            write_success (If data is writed success,then "write_success=1" else "write_success=0")
*********************************************************************************************/
void uart_send (uchar uart_port_num,uchar send_byte_num){
    uchar i;
    switch(uart_port_num){
        case 0:{                            // 上位机需要向串口 0 发送数据
            for(i=0;i<send_byte_num;i++){   // 将临时缓冲区的数据转移到串口 0 的发送缓冲区中
            uart0_t_buf[i]=send_buf[i];
            }
            uart0_send_num=send_byte_num;
            write_success=1;                // 置串口 0 数据"发送成功"标志
            if(uart_busy==0){               // 判断母串口发送中断是否已经启动
                TI=1;                       // 母串口一直没有发送数据,需要启动发送中断
                uart_pointer=0;             // 其他串口无发送数据，直接将指针指向串口 0 发送数据
                break;                      // 如果串口 0 的上一包数据没有发送完,则该次数据包
            }                               // 发送不成功(数据标志为："write_success==0")
            else {break;}                   // 母串口正发送其他子串口数据,不需再次启动发送中断
        }
        case 1:{                            // 上位机需要向串口 1 发送数据
            for(i=0;i<send_byte_num;i++){
            uart1_t_buf[i]=send_buf[i];
            }
            uart1_send_num=send_byte_num;
            write_success=1;
            if(uart_busy==0){
                uart_pointer=1;
                TI=1;
                break;
            }
            else {break;}
        }
```

```
            case 2:{                                    // 上位机需要向串口 2 发送数据
                for(i=0;i<send_byte_num;i++){
                uart2_t_buf[i]=send_buf[i];
                }
                uart2_send_num=send_byte_num;
                write_success=1;
                if(uart_busy==0){
                    uart_pointer=2;
                    TI=1;
                    break;
                }
                else {break;}
            }

            case 3:{                                    // 上位机需要向串口 3 发送数据
                for(i=0;i<send_byte_num;i++){
                uart3_t_buf[i]=send_buf[i];
                }
                uart3_send_num=send_byte_num;
                write_success=1;
                if(uart_busy==0){
                    uart_pointer=3;
                    TI=1;
                    break;
                }
                else {break;}
            }

            default:{                                   // 上位机需要向串口 4 发送数据
                for(i=0;i<send_byte_num;i++){
                uart4_t_buf[i]=send_buf[i];
                }
                uart4_send_num=send_byte_num;
                write_success=1;
                if(uart_busy==0){
                    uart_pointer=4;
                    TI=1;
                    break;
                }
                else {break;}
            }
        }
}

uart_receive (void){ ; }                                // 用户自己编写的母串口接收处理函数
```

```
reset_chip (void){                          // 该函数将让 SP2538 复位
    f_reset_chip=1;
    if(!uart_busy){
        uart_pointer=5;
        TI=1;
    }
    else;
}

sleep_chip (void){                          // 该函数将让 SP2538 进入睡眠模式
    f_sleep_chip=1;
    if(!uart_busy){
        uart_pointer=6;
        TI=1;
    }
    else;
}

clear_wdt (void){                           // 该函数将对 SP2538 进行喂狗
    f_clear_wdt=1;
    if(!uart_busy){
        uart_pointer=5;
        TI=1;
    }
    else;
}

disable_wdt (void){                         // 该函数将禁止 SP2538 中的看门狗
    f_disable_wdt=1;
    if(!uart_busy){
        uart_pointer=6;
        TI=1;
    }
    else;
}

enable_wdt (void){                          // 该函数将使能 SP2538 中的看门狗
    f_enable_wdt=1;
    if(!uart_busy){
        uart_pointer=6;
        TI=1;
    }
    else;
}
```

```
#ifdef _DEBUG_SUB_MODULE
main(){
    TMOD=0x20;
    TH1=0xff;                   // 57600bps@11.0592MHz
    TCON=0x40;
    SCON=0x50;
    PCON=0x80;                  // 波特率加倍
    IE=0x90;
    P1=0;
    while(1){
        send_buf[0]=0x31;
        send_buf[1]=0x32;
        send_buf[2]=0x33;
        send_buf[3]=0x34;
        send_buf[4]=0x35;
        send_buf[5]=0x36;
        write_success=0;        // 调用函数"void uart_send (uchar uart_port_num,uchar send_byte_num)"
        if(!uart0_send_num){    // 前必须先将"write_success"标志清"0"
            uart_send(0,4);     // 向串口 0 发送 3 个字节数据(按照先后顺序分别为:
        }                       // "0x34"、"0x33"、"0x32"、"0x31")

        send_buf[0]=0x41;
        send_buf[1]=0x42;
        send_buf[2]=0x43;
        send_buf[3]=0x44;
        send_buf[4]=0x45;
        send_buf[5]=0x46;
        send_buf[6]=0x47;
        write_success=0;
        if(!uart1_send_num){    // 向串口 1 发送 6 个字节数据(按照先后顺序分别为:
            uart_send(1,6);     // "0x46"、"0x45"、"0x44"、"0x43"、"0x42"、"0x41")
        }

        send_buf[0]=0x51;
        send_buf[1]=0x52;
        send_buf[2]=0x53;
        send_buf[3]=0x54;
        send_buf[4]=0x55;
        send_buf[5]=0x56;
        send_buf[6]=0x57;
        send_buf[7]=0x58;
        write_success=0;
        if(!uart2_send_num){    // 向串口 2 发送 8 个字节数据(按照先后顺序分别为:
            uart_send(2,8);     // "0x58"、"0x57"、"0x56"、"0x55"、"0x54"、"0x53"... ...）
        }
```

```
            send_buf[0]=0x61;
            send_buf[1]=0x62;
            send_buf[2]=0x63;
            send_buf[3]=0x64;
            send_buf[4]=0x65;
            send_buf[5]=0x66;
            send_buf[6]=0x67;
            send_buf[7]=0x68;
            write_success=0;
            if(!uart3_send_num){        // 向串口 3 发送 8 个字节数据(按照先后顺序分别为:
                uart_send(3,8);         // "0x68"、"0x67"、"0x66"、"0x65"、"0x64"、"0x63"... ...）
            }

            send_buf[0]=0x71;
            send_buf[1]=0x72;
            send_buf[2]=0x73;
            send_buf[3]=0x74;
            send_buf[4]=0x75;
            send_buf[5]=0x76;
            send_buf[6]=0x77;
            send_buf[7]=0x78;
            write_success=0;
            if(!uart4_send_num){        // 向串口 4 发送 7 个字节数据(按照先后顺序分别为:
                uart_send(4,7);         // ""0x77"、"0x76"、"0x75"、"0x74"、"0x73"... ...）
            }
        }
}
#endif                                  // _DEBUG_SUB_MODULE
```

注：本源代码请到公司网站 www.sepertech.com 下载

成都视普科技有限公司 Chengdu Seper Technology Co., Ltd.

地址：成都市高新区高朋大道 5 号（中国·成都留学人员创业园）A 座 4 楼
邮编：610041
电话：028-85138086 85188046
传真：028-85188046
网站：Http://www.sepertech.com
电邮：seper@sepertech.com（销售）
help@sina.com （技术支持）