第10章 STC单片机串行异步收发器原理及实现

何宾

2015.02

本章主要内容

- RS-232标准概述
- STC单片机串口模块概述
- 串口1寄存器及工作模式
- 串口2寄存器及工作模式
- 串口3寄存器及工作模式
- 串口4寄存器及工作模式
- 串行通信综合实现

RS-232标准概述

基于通用串行异步收发器的异步串行通信（简称RS-232），是计算机通信中最经典的一个通信方式。

- RS-232是美国电子工业联盟（Electronicl Industries Association，EIA）制定的串行数据通信的接口标准，原始编号全称是EIA-RS-232（简称232，RS232）。
- 它被广泛用于计算机串行接口外设连接。

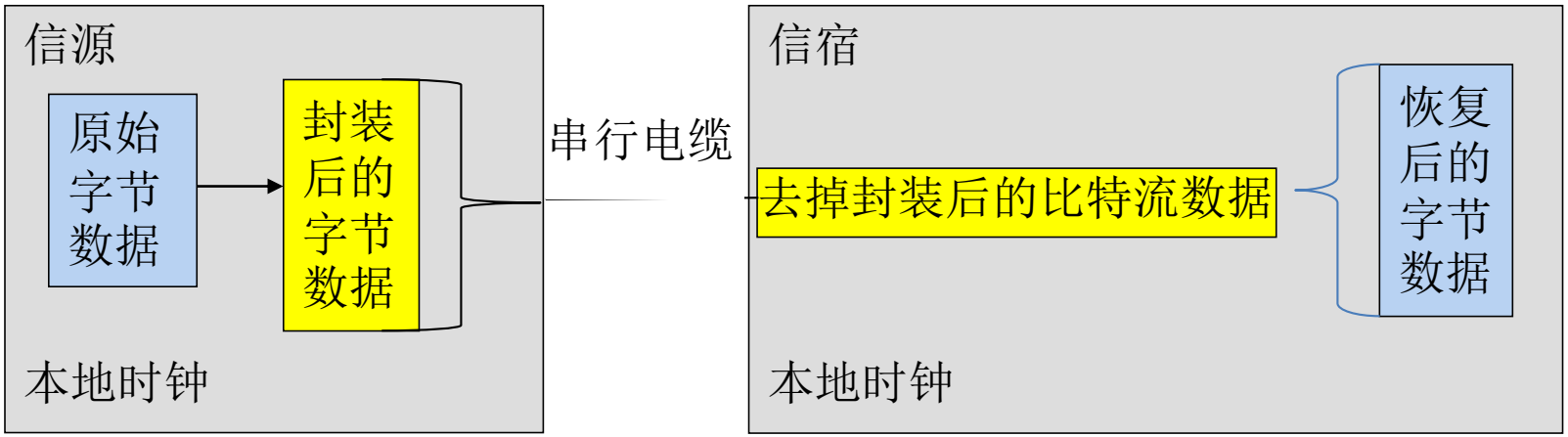
RS-232标准概述
--RS-232传输特点

在RS-232标准中，有下面显著的特点：

- 字符是按一个比特接着另一个比特的方式，使用一根信号线进行传输。这就是我们通常所说的串行方式传输数据.
  - 这种传输方式的优点是传输线少，连线简单，传送距离可以较远。

RS-232标准概述
--RS-232传输特点

- 对于信源（发送方）来说，需将并行的原始数据，进行封装，然后转换成一位一位的串行比特流数据进行发送；对于信宿（目的方）来说，当接收到串行比特流数据后，对接收到的数据进行解析，从数据中找到原始数据的比特流，将其转换成并行数据



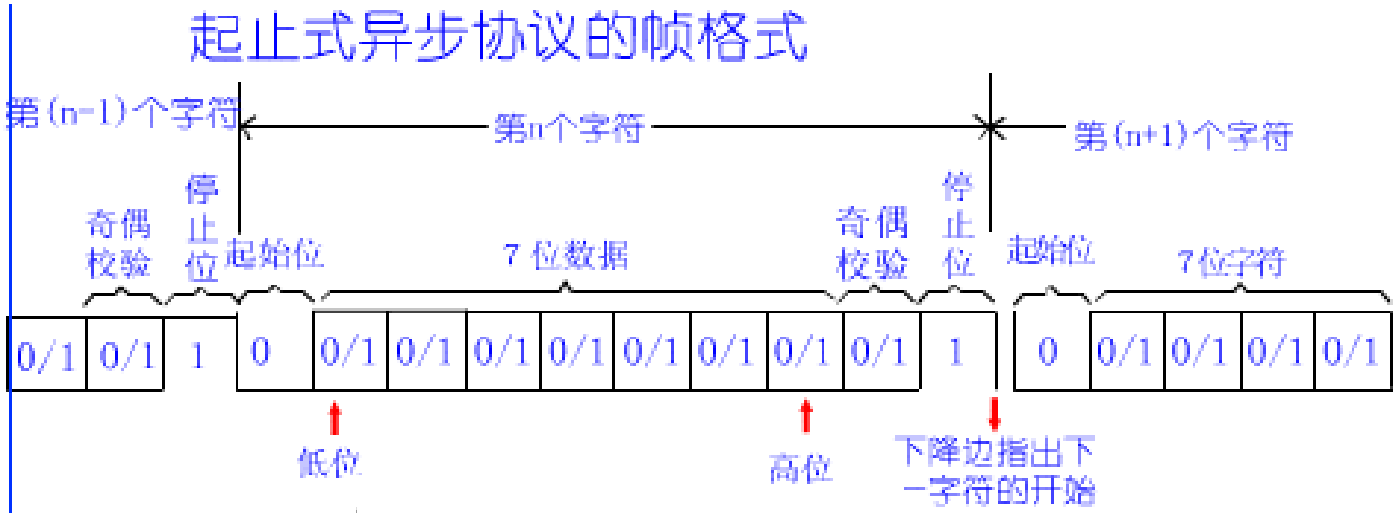
RS-232标准概述
--RS-232传输特点

- 在从信源（发送方）发送数据给信宿（目的方）的时候，并不需要传输时钟信号。
  - 当信宿接收到串行数据的时候，会使用信宿本地的时钟对接收到的数据进行采样和解码，然后将数据恢复出来。
- 通过RS-232在传送数据时，并不需要额外使用一个信号来传送同步信息。
  - 通过在数据前部和尾部加上识别标志，就能正确的将数据顺利传送到对方。

注：在计算机中，将实现RS-232通信功能的专用芯片，典型的8251芯片，称为通用异步接收发送器 (Universal Asynchronous Receiver Transmitter，UART)。

RS-232标准概述
--RS-232数据传输格式

在RS-232中，使用的编码格式是异步起停数据格式



RS-232标准概述
--RS-232数据传输格式

在该数据格式中：

- 首先有一个逻辑0标识的起始位，该位标识新的一帧数据的开始。
- 在起始位后面紧跟7或者8 个比特数据
  - 数据比特的开始位对应于原始字节数据的最低位，数据比特的结束位对应于原始字节数据的最高位。
- 数据比特后面跟随可选的奇偶校验比特。可以在发送数据的时候进行设置
- 最后是以逻辑1标识的1~2个停止比特位。

RS-232标准概述
--RS-232数据传输格式

- 在一个异步起停数据格式中，发送一个8位的字符数据至少需要10个比特位。
- 在协议中，每一个比特位持续的时间和发送时钟有关。
  - 一个时钟周期发送一个比特位。将这个时钟称为波特率时钟，用波特率表示，即：每秒中发送比特位的个数。

注：在采用RS-232通信协议的信源和信宿，必须采用相同的数据格式，以及波特率时钟。

RS-232标准概述
--RS-232电气标准

RS-232标准

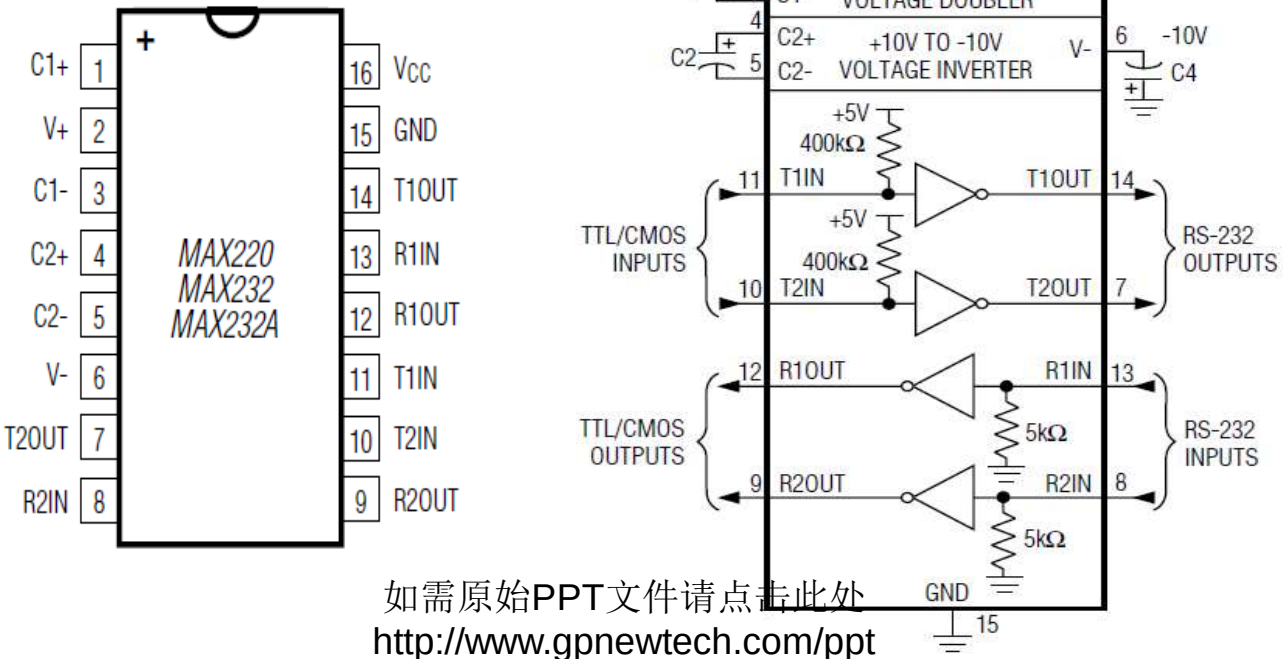
- 分别定义了逻辑1和逻辑0的电压范围。
  - 逻辑1的电压范围为-15~-3V；
  - 逻辑0的电压范围为+3~+15V。

注：在RS-232中，接近零的电平是无效的。

- 这与传统数字逻辑中，对逻辑1和逻辑0的定义是不同的。
  - 需要进行电气标准的转换，将TTL/CMOS电平转换为RS-232电平，以及将RS-232电平转换为TTL/CMOS电平。

RS-232标准概述
--RS-232电气标准

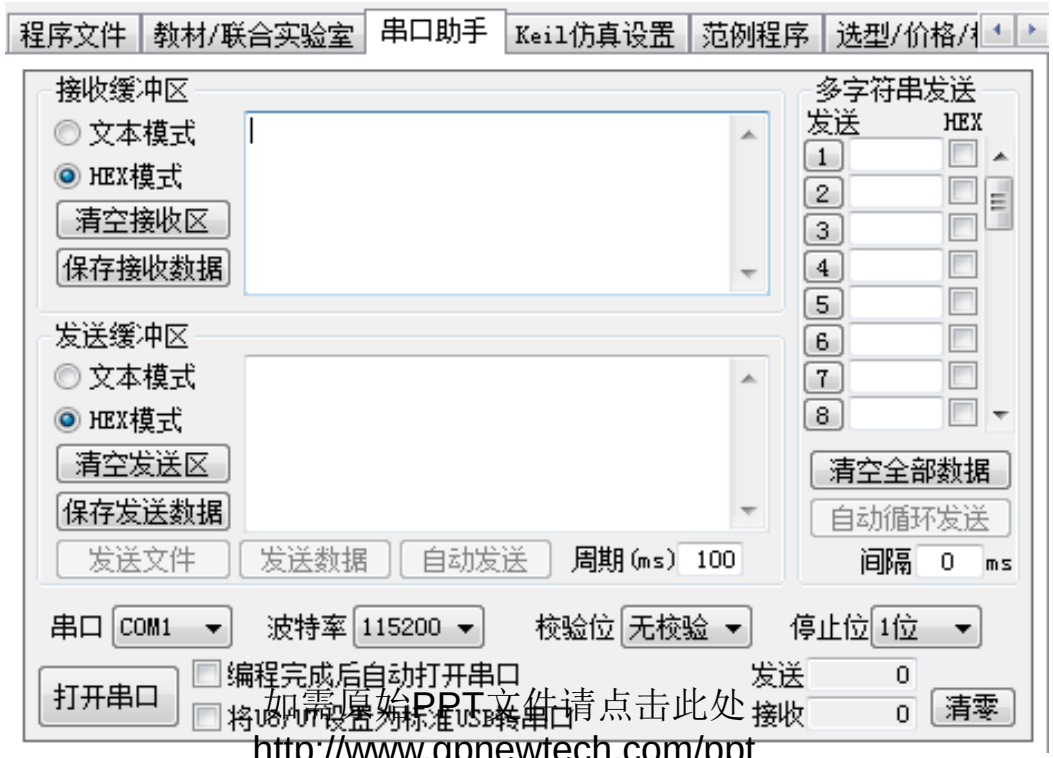
美信公司的MAX232芯片，可以实现TTL/ CMOS电平与RS-232电平之间的相互转换。



RS-232标准概述
--RS-232参数设置

打开STC-ISP软件，在串口助手标签下，可以看到串口参数设置界面。

- **在该界面中，需要设置以下：波特率、奇偶校验和停止位。**



RS-232标准概述
--RS-232参数设置

波特率

- 是指从一设备发到另一设备的波特率，即每秒钟发送的比特位的个数，单位为波特率（bits per second，bit/s）。
  - 典型地，可选择的波特率有300、1200、2400、9600、19200、115200 等。

注：一般通信两端设备都要设为相同的波特率，有些设备也可以设置为自动检测波特率。

RS-232标准概述
--RS-232参数设置

奇偶校验

- 是用来验证接收数据的正确性。一般不使用奇偶校验，若使用，那么既可以选择设置为奇校验或选择设置为偶校验。
  - 在偶校验中，要求所有发送数据的比特（包括校验位在内）1的个数是偶数。根据这个校验标准，在校验位置1或者置0；
  - 在奇校验中，要求所有发送数据的比特（包括校验位在内）1的个数是奇数。根据这个校验标准，在校验位置1或者置0；

RS-232标准概述
--RS-232参数设置

停止位

- 是在每个字节传输之后发送的，它用来帮助接受信号方硬件重同步。
  - 比如：在传输8位原始数据11001010时，数据的前后就需加入起始位（逻辑低）以及停止位（逻辑高）。

注：起始位固定为一个比特，而停止位则可以是1、1.5或者2个比特位。这由使用RS-232的信源与信宿共同确定，并且通过软件进行设置。

RS-232标准概述
--RS-232参数设置

流量控制

- 当需要发送握手信号或数据完整性检测时需要制定其他设置。公用的组合有RTS/CTS, DTR/DSR或者XON/XOFF。这种方式称为硬件流量控制。

注：通常为了简化连接和控制，不使用硬件流量控制方式。

RS-232标准概述
--RS-232连接器

RS-232指定了20个不同的信号连接，由25个D-sub（微型D类）管脚构成的DB-25连接器。

- DB-25和DB-9型的连接器在大部分设备上是雌型（母头，即插孔），但并不一定都是这样，有些设备上就是雄型（公头，即插针）。

RS-232串口连接器-母头

RS-232串口连接器-母头和公头引脚顺序

RS-232标准概述
--RS-232连接器

DB-9连接器信号定义

引脚名字	序号	功能
公共接地	5	地线
发送数据（/TXD）	3	发送数据
接受数据（/RXD）	2	接收数据
数据终端准备（DTR）	4	终端设备通知调制解调器可以进行数据传输
数据准备好（DSR）	6	调制解调器通知终端设备准备就绪
请求发送（RTS）	7	终端设备要求调制解调器将数据提交
清除发送（CTS）	8	调制解调器通知终端设备可以传数据过来
数据载波检测（CD）	1	调制解调器通知终端设备侦听到载波信号
振铃指示（RI）	9	调制解调器通知终端设备有电话进来

STC单片机串口模块概述
--串口模块结构

两个数据缓冲区

- 每个串行接口的数据缓冲区由两个独立的接收缓冲区和发送缓冲区构成。
  - 这两个缓冲区可以同时收发数据。用户只能向发送缓冲区写入数据；而从接收缓冲区读取数据。
- 两个缓冲区共用一个地址。
  - 串口1的两个缓冲区SBUF地址为0x99；
  - 串口2的两个缓冲区S2BUF地址为0x9B；
  - 串口3的两个缓冲区S3BUF地址为0xAD；
  - 串口4的两个缓冲区S4BUF地址为0x85。

STC单片机串口模块概述
--串口模块结构

- 一个移位寄存器；
- 一个串行控制寄存器；
- 一个波特率发生器。

注：对于串口1来说有四种工作方式，其中两种工作方式的波特率可变，另外两种是固定的；串口2/串口3/串口4都只有两种工作模式，这两种方式的波特率都是可变的。

STC单片机串口模块概述
--串口引脚

串口1可用的引脚

- STC15W4K32S4系列单片机串口1对应的引脚是TxD和RxD。
- 串口1可以在3组引脚之间进行切换。
- 通过设置AUXR1（P_SW1）寄存器中的S1_S1比特位和S1_S0比特位
  - 可以将串口1从[RxD/P3.0，TxD/P3.1]切换到
  - [RxD_2/P3.6，TxD_2/P3.7]
  - 还可以切换到[RxD_3/P1.6/XTAL2，TxD_3/P1.7/XTAL1]。

STC单片机串口模块概述
--串口引脚

串口2可用的引脚

- STC15W4K32S4系列单片机串口2对应的引脚是TxD2和RxD2。
- 串口2可以在2组引脚之间进行切换。
- 通过设置P_SW2寄存器中的S2_S比特位，可以
  - 将串口2从[RxD2/P1.0，TxD2/P1.1]
  - 切换到[RxD2_2/P4.6，TxD2_2/P4.7]

STC单片机串口模块概述
--串口引脚

串口3可用的引脚

- STC15W4K32S4系列单片机串口3对应的引脚是TxD3和RxD3。
- 串口3可以在2组引脚之间进行切换。通过设置P_SW2寄存器中的S3_S比特位
  - 可以将串口3从[RxD3/P0.0，TxD3/P0.1]
  - 切换到[RxD3_2/P5.0，TxD3_2/P5.1]

STC单片机串口模块概述
--串口引脚

串口4可用的引脚

- STC15W4K32S4系列单片机串口4对应的引脚是TxD4和RxD4。
- 串口4可以在2组引脚之间进行切换。通过设置P_SW2寄存器中的S4_S比特位，可以将
  - 串口4从[RxD4/P0.2，TxD4/P0.3]
  - 切换到[RxD4_2/P5.2，TxD4_2/P5.3]

串口1寄存器及工作模式
--串口1寄存器组

串口1控制寄存器SCON

- 该寄存器位于STC单片机特殊功能寄存器地址为0x98的位置。
- 当复位后，该寄存器的值为“00000000”。

串口1控制寄存器SCON各位的含义

比特	B7	B6	B5	B4	B3	B2	B1	B0
名字	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI

- SM0/FE
  - 当PCON寄存器中SMOD0比特位为1时，该位用于检测侦错误。当检测到一个无效的停止位时，通过UART接收器将该位置1。

注：该位由软件清零。

  - 当PCON寄存器中SMOD0比特位为0时，该位和SM1位一起指定串口1的通信方式。

串口1寄存器及工作模式
--串口1寄存器组

■ SM1

- 该位和SM0位一起确定串口1的通信方式。

SM1和SM0各位的含义

SM0	SM1	工作模式	功能说明	波特率
0	0	模式0	同步移位串行方式：移位寄存器	当UART_M0x6=0时，波特率为SYSclk/12 当UART_M0x6=1时，波特率为SYSclk/2
0	1	模式1	8位UART，波特率可变	当串口1用定时器1作为其波特率发生器且定时器工作于模式0或串行口用定时器2作为其波特率发生器时，波特率=（定时器1的溢出率或者定时器T2的溢出率）/4 当串口1用定时器1作为其波特率发生器且定时器1工作于模式2（8位自动重加载模式）时，波特率=（$2^{SMOD}/32$）×（定时器1的溢出率）
1	0	模式2	9位UART	波特率=（$2^{SMOD}/64$）×SYSclk系统工作时钟频率
1	1	模式3	9位UART，波特率可变	当串口1用定时器1作为其波特率发生器且定时器工作于模式0或串行口用定时器2作为其波特率发生器时，波特率=（定时器1的溢出率或者定时器T2的溢出率）/4 当串口1用定时器1作为其波特率发生器且定时器1工作于模式2时，波特率=（$2^{SMOD}/32$）×（定时器1的溢出率）

串口1寄存器及工作模式
--串口1寄存器组

- SM2

允许方式2或者方式3多机通信控制位。在方式2或者方式3时，如果SM2位为1，则接收机处于地址帧选状态。此时可以利用接收到的第9位（即RB8）来筛选地址帧：

- 当RB8=1时，说明该帧为地址帧，地址信息可以进入SBUF，并使得RI置1，进而在中断服务程序中再进行地址号比较；
- 当RB8=0时，说明该帧不是地址帧，应丢掉并保持RI=0。

注：在方式2或者方式3中，如果SM2位为0且REN位为1，接收机处于禁止筛选地址帧状态。不论收到的RB8是否为1，均可使接收到的信息进入SBUF，并使得RI置1，此时RB8通常为校验位。

串口1寄存器及工作模式
--串口1寄存器组

- **REN**
  - 允许/禁止串行接收控制位。当该位为1时，允许串行接收状态，可以启动串行接收器RxD，开始接收信息；当该位为0时，禁止串行接收状态，禁止串行接收器RxD。
- **TB8**
  - 当选择方式2或者方式3时，该位为要发送的第9位数据，按需要由软件置1或者清0。例如：可用作数据的校验位或者多机通信中表示地址帧/数据帧的标志位。
- **RB8**
  - 当选择方式2或者方式3时，该位为接收到的第9位数据，作为奇偶校验位或者地址帧/数据帧的标志位。

串口1寄存器及工作模式
--串口1寄存器组

- TI
  - 发送中断请求标志位。在方式0时，当串行发送数据第8位结束时，由硬件自动将该位置1，向CPU发出中断请求。当CPU响应中断后，必须由软件将该位清0。在其它方式中，则在停止位开始发送时由硬件置1，向CPU发出中断请求。同样的，当CPU响应中断后，必须由软件将该位清0。
- RI
  - 接收中断请求标志位。在方式0时，当串行接收数据第8位结束时，由硬件自动将该位置1，向CPU发出中断请求。当CPU响应中断后，必须由软件将该位清0。在其它方式中，则在接收到停止位的中间时刻由内部硬件置1，向CPU发出中断请求。同样的，当CPU响应中断后，必须由软件将该位清0。

串口1寄存器及工作模式
--电源控制寄存器

电源控制寄存器PCON

- 该寄存器位于STC单片机特殊功能寄存器地址为0x87的位置。
- 当复位后，该寄存器的值为“00110000”。

电源控制寄存器PCON各位的含义

比特	B7	B6	B5	B4	B3	B2	B1	B0
名字	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL

串口1寄存器及工作模式
--电源控制寄存器

- **SMOD**
  - 波特率选择位。当该位为1时，则使串行通信方式1、2和3的波特率加倍；当该位为0时，则使各工作方式的波特率不加倍。
- **SMOD0**
  - 帧错误检测有效控制位。当该位为1时，SCON寄存器中的SM0/FE比特位用于FE（帧错误检测）功能；当该位为0时，SCON寄存器中的SM0/FE比特用于SM0功能，该位和SM1比特位一起用来确定串口的工作方式。

串口1寄存器及工作模式
--串口数据缓冲寄存器

串口数据缓冲寄存器

- STC15系列单片机的串口1缓冲寄存器SBUF地址为0x99。在该地址实际是两个缓冲寄存器。
  - 一个缓冲寄存器用于保存要发送的数据；而另一个缓冲寄存器用于读取已经接收到的数据。
- 在串口的串行通道内，设置数据寄存器。
  - 在该串口所有工作模式中，在写入信号SBUF的控制下，把数据加载到相同的9位移位寄存器中，前面8位为数据字节，最低位为移位寄存器的输出位。根据所设置的工作模式，自动将1或者TB8的值加载到移位寄存器的第9位，并进行发送。

串口1寄存器及工作模式
--串口数据缓冲寄存器

- 在串口的接收寄存器是一个输入移位寄存器。
  - 当设置为方式0时，它的字长为8位；
  - 当设置为其它工作模式时，它的字长为9位。
- 当接收完一帧数据后，将移位寄存器中的串行字节数据加载到数据缓冲寄存器SBUF中，将其第9位加载到SCON寄存器的RB8位。如果由于SM2使得已经接收到的数据无效时，RB8和SBUF中的内容不变。
  - 由于在串行通道内设置了输入移位寄存器和SBUF缓冲寄存器，从而在接收完一帧串行数据将其从移位寄存器加载到并行SBUF缓冲寄存器后，可以立即开始接收下一帧数据。

串口1寄存器及工作模式
--辅助寄存器

辅助寄存器AUXR

- 该寄存器位于STC单片机特殊功能寄存器地址为0x8E的位置。
- 当复位后，该寄存器的值为“00000000”。

辅助寄存器AUXR各位的含义

比特	B7	B6	B5	B4	B3	B2	B1	B0
名字	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2

- T1x12
  - 定时器1速度控制位。当该位为0时，定时器1是传统8051的速度，12分频；当该位为1时，定时器1的速度是传统8051的12倍，不分频。

串口1寄存器及工作模式
--辅助寄存器

- **UART_M0x6**
  - 串口模式0的通信速率设置位。当该位为0时，串口1模式0的速度是传统8051单片机串口的速度，12分频；当该位为1时，串口1模式0的速度是传统8051单片机速度的6倍2分频。
- **S1ST2**
  - 串口1选择定时器2作波特率发生器的控制位。当该位为0时，选择定时器1作为串口1的波特率发生器；当该位为1时，选择定时器2作为串口1的波特率发生器。

串口1寄存器及工作模式
--从机地址控制寄存器

在STC单片机中，设置了从机地址寄存器SADEN和SADDR。

- SADEN寄存器为从机地址掩模寄存器
  - 该寄存器位于特殊功能寄存器地址为0xB9的位置。
  - 当复位后，该寄存器的值为“00000000”；
- SADDR寄存器为从机地址寄存器
  - 该寄存器位于特殊功能寄存器地址为0xA9的位置。
  - 当复位后，该寄存器的值为“00000000”。

串口1寄存器及工作模式
--中断允许寄存器

中断允许寄存器IE。

- **该寄存器位于STC单片机特殊功能寄存器地址为0xA8的位置。**
- **当复位后，该寄存器的值为“00000000”。**

中断允许寄存器IE各位的含义

比特	B7	B6	B5	B4	B3	B2	B1	B0
名字	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0

- **ES**
  - **串口1中断允许位。当该位为1时，允许串口中断；当该位为0时，禁止串口中断。**

串口1寄存器及工作模式
--中断优先级控制寄存器

中断优先级控制寄存器IP

- 该寄存器位于STC单片机特殊功能寄存器地址为0xB8的位置。
- 当复位后，该寄存器的值为“00000000”。

中断优先级控制寄存器IP各位的含义

比特	B7	B6	B5	B4	B3	B2	B1	B0
名字	PPCA	PLVD	PADC	PS	PT1	PX1	PT0	PX0

- PS
  - 串口1中断优先级控制位。当该位为0时，串口1中断为最低优先级中断（优先级为0）；当该位为1时，串口1中断为最高优先级中断（优先级1）。

串口1寄存器及工作模式
--串口1中继广播方式设置

串口1中继广播方式设置是在CLK_DIV寄存器中实现

- 该寄存器位于STC单片机特殊功能寄存器地址为0x97的位置。
- 当复位后，该寄存器的值为“00000000”。

CLKDIV寄存器各位的含义

比特	B7	B6	B5	B4	B3	B2	B1	B0
名字	MCKO_S1	MCKO_S0	ADRJ	Tx_Rx	MCLKO_2	CLKS2	CLKS1	CLKS0

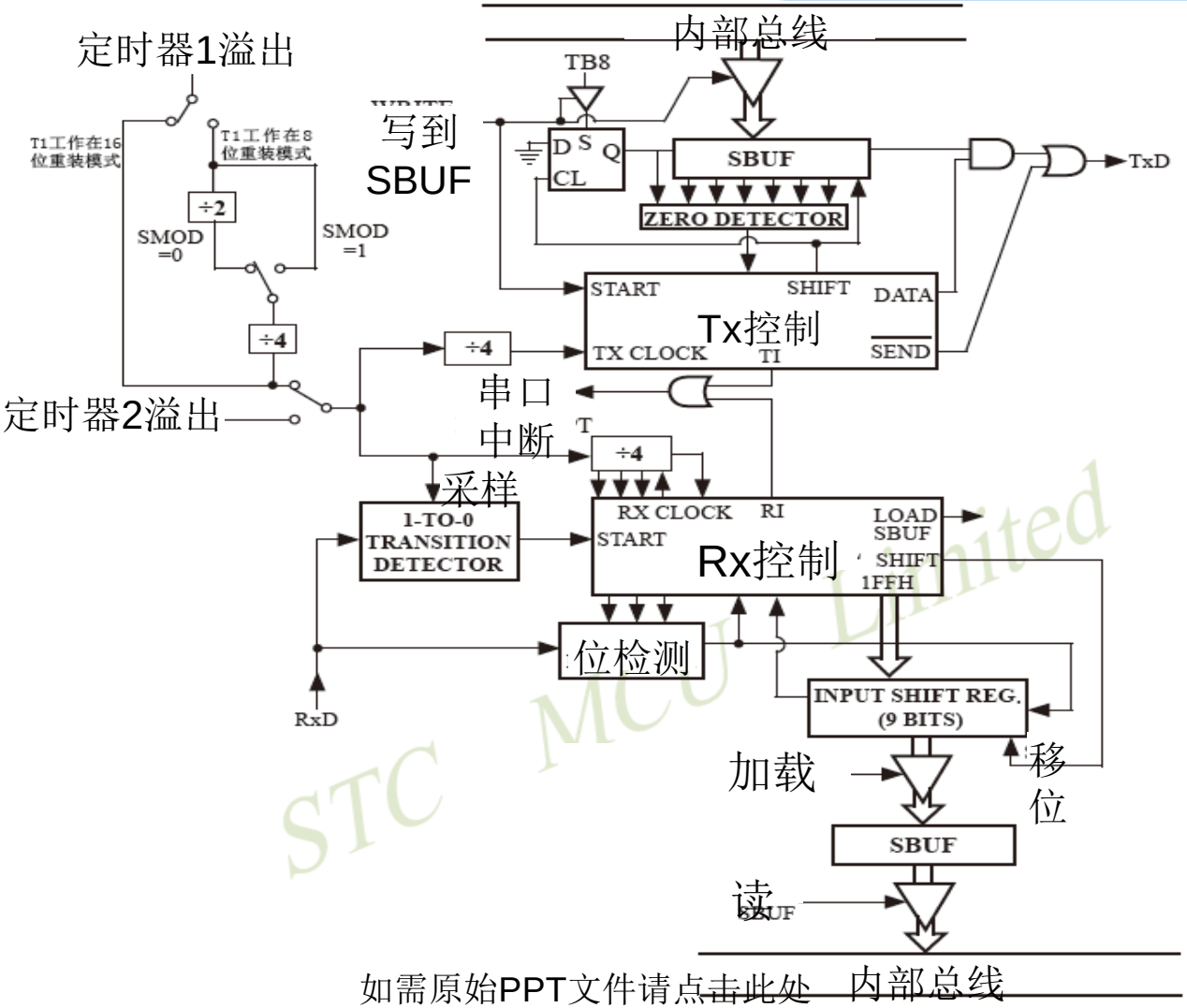
- Tx_Rx
  - 串口1中继广播方式设置位。当该位为0时，串口1为正常工作模式；当该位为1时，串口1为中继广播方式，将RxD端口输入的电平状态实时输出到TxD外部引脚上，TxD引脚可以对RxD引脚的输入信号进行实时整形放大输出，TxD引脚对外输出实时反映RxD端口输入的电平状态。

串口1寄存器及工作模式
--串口1工作模式

串口1有四种工作模式，可以通过对SCON寄存器的SM0和SM1的设置进行选择。

- 模式1、模式2和模式3为异步通信方式，每个发送和接收的字符都带有1个起始位、1个停止位。
- 在模式0中，串口1作为1个简单的移位寄存器使用。

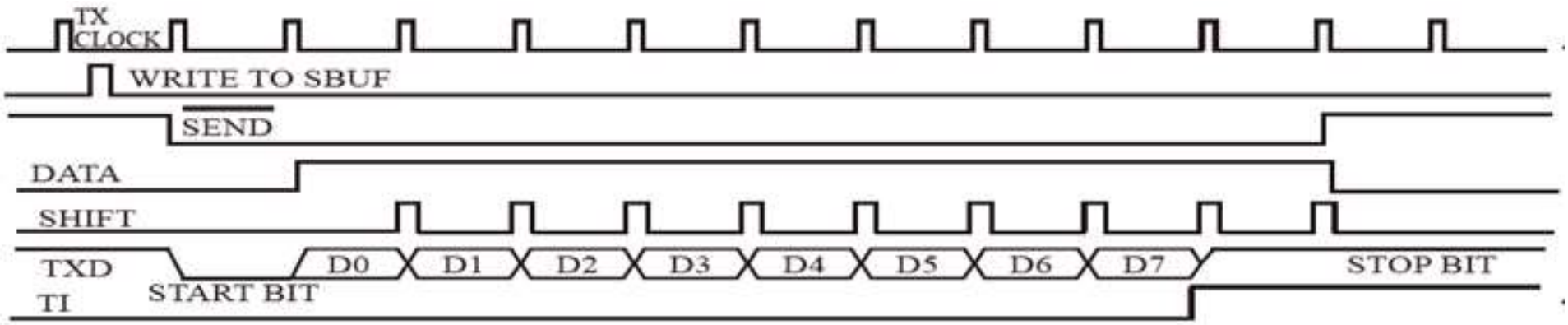
串口1寄存器及工作模式
--串口1工作模式（8位波特率可变）



串口1工作模式（8位波特率可变）
--串口1发送过程

当串口1发送数据时，数据从单片机的串行发送引脚TxD发送出去。

- 当主机执行一条写SBUF的指令时，就启动串口1的数据发送过程，写SBUF信号将1加载到发送移位寄存器的第9位，并通知Tx控制单元开始发送。
- 通过16分频计数器，同步发送串行比特流。

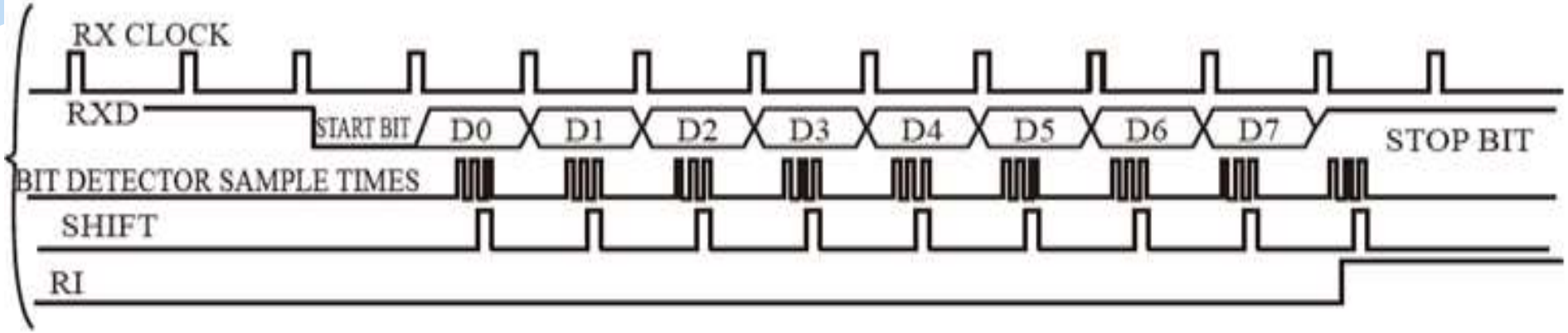


串口1工作模式（8位波特率可变）
--串口1接收过程

当软件将接收允许标志位REN置1后，接收器就用选定的波特率的16分频的速率采样串行接收引脚RxD。

- 当检测到RxD端口从1到0的负跳变后，就启动接收器准备接收数据。
- 同时，复位16分频计数器，将值0x1FF加载到移位寄存器中。复位16分频计数器使得它与输入位时间同步。
- 分频计数器的16个状态时将每位接收的时间平均为16等份。在每位时间的第7、8和9状态由检测器对RxD端口进行采样，所接收的值是这次采样值经过“三中取二”的值，即：三次采样中，至少有两次相同的值，用来抵销干扰信号，提高接收数据的可靠性。
- 在起始位，如果接收到的值不为0，则起始位无效，复位接收电路，并重新检测1到0的跳变。如果接收到的起始位有效，则将它输入移位寄存器，并接收本帧的其余信息。

串口1工作模式（8位波特率可变）
--串口1接收过程



- 接收的数据从接收移位寄存器的右边移入，将已装入的0x1FF向左边移出。当起始位0移动到移位寄存器的最左边时，使RX控制器做最后一次移位，完成一帧的接收。若同时满足以下两个条件时：
  - RI=0；
  - SM2=0或接收到的停止位为1。

串口1工作模式（8位波特率可变）
--串口1接收过程

则接收到的数据有效，实现加载到SBUF，停止位进入RB8，置位RI，向CPU发出中断请求信号。

- 如果这两个条件不能同时满足，则将接收到的数据丢弃，无论条件是否满足，接收机又重新检测RxD端口上的1到0的跳变，继续接收下一帧数据。

注：如果接收有效，则在响应中断后，必须由软件将标志RI清零。

串口1通信实例1

在该设计中，将在STC提供的学习板上，使用串口1，定时器1的模式0实现STC学习板和主机电脑的串口通信。

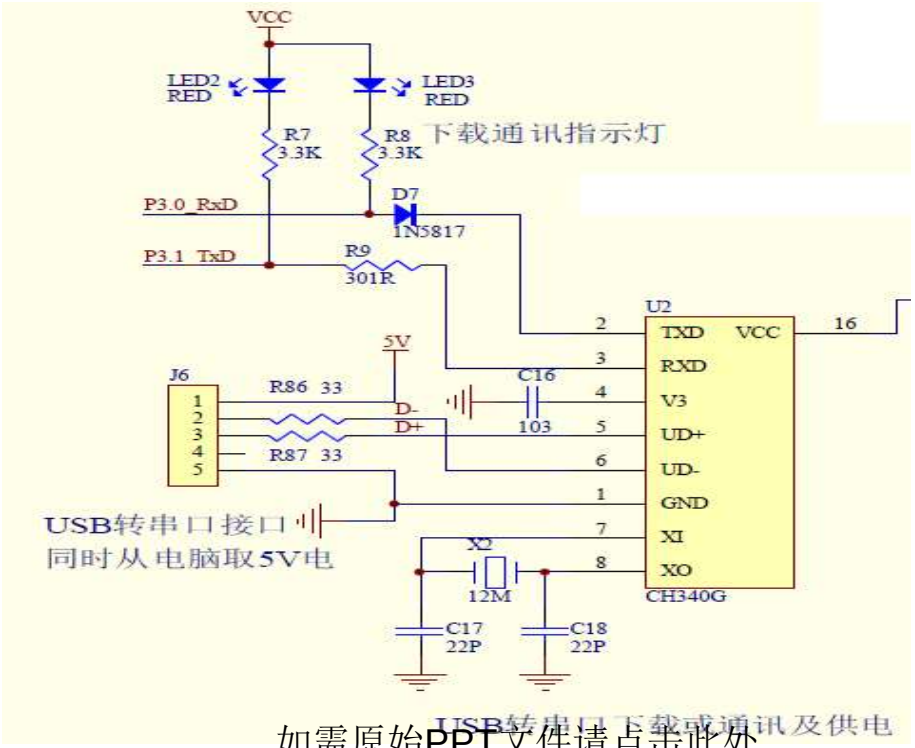
- STC通过串口1向主机发送菜单界面。
  - 在主机上，按1键，用于控制STC学习板上的标记为LED10的LED灯；
  - 按2键，用于控制STC学习板上标记为LED9的LED灯；
  - 按其他键显示退出程序的信息。

```
------main menu----------------
      input 1:  Control LED10
      input 2:  Control LED9
      other  :  exit program
------end menu----------------
```

串口1通信实例1

在该设计中，使用STC学习板上的串口1。

- CH340G芯片用于将IAP15W4K58S4单片机的串口信号TxD和RxD转换成USB信号，方便与电脑USB接口的连接。



串口1通信实例1

- 串口发送信号TxD信号连接到STC单片机的P3.1引脚，该引脚将从STC单片机发送数据给主机；
- 串口接收信号RxD信号连接到STC单片机的P3.0引脚，该引脚将接收来自主机的数据。

- 在该电路设计中，LED2和LED3上拉，并且连接到RxD和TxD信号线
  - 用于指示STC单片机串口和主机之间发送和接收数据的情况。
- 在P3.0引脚上加入IN5817二极管，以及在P3.1引脚串入电阻的作用是为了防止USB器件给芯片供电。

串口1通信实例1

【例】 主机通过串口控制STC板上LED灯C语言描述的例子。

```
#include "reg51.h"
#define FOSC  18432000L                  //声明当前单片机主时钟频率
#define BAUD 115200                      //声明波特率常数115200
sfr AUXR  =0x8E;                         //声明AUXR寄存器的地址0x8E
sfr TH2   =0xD6;                         //声明TH2寄存器的地址0xD6
sfr TL2   =0xD7;                         //声明TL2寄存器的地址0xD7
bit busy=0;                              //声明比特位busy
xdata char menu[]={"\r\n------main menu----------"   //声明字符型数组menu
                   "\r\n     input 1:  Control LED10 "
```

串口1通信实例1

```
            "\r\n     input 2:  Control LED9 "
             "\r\n     other  :  Exit Program"
              "\r\n------end menu----------------"};
void SendData(unsigned char dat)                    //声明SendData子函数，参数dat
{      while(busy);                                 //判断是否忙，忙等待
       SBUF=dat;                                    //将dat写入SBUF发送缓冲器
       busy=1; }                                    //将busy标志置1
void SendString(char *s)                            //声明SendString子函数，参数s
{
       while(*s!= '\0' )                            //判断字符是否结束，如果没结束
       SendData(*s++);}                             //调用SendData子函数发送数据
```

串口1通信实例1

```
void uart1() interrupt 4                    //声明串口1中断服务程序uart1
{
        if(RI)                              //如果接收标志RI为1,有接收数据
                RI=0;                       //将RI标志清零
        if(TI)                              //如果发送标志TI为1,已发送数据
                TI=0;                       //将TI标志清零
         busy=0;                            //busy清零，表示已经发送完数据
}
void main()
{
        unsigned char c;                    //定义无符号字符型变量c
```

串口1通信实例1

```
P46=0;                                  //P4.6端口置0，灯亮
P47=0;                                  //P4.7端口置1，灯亮
SCON=0x50;                              //串口1方式1，允许接收
AUXR=0x14;                                  //允许定时器2，不分频
AUXR|=0x01;                                 //选择定时器2作为波特率发生器
TL2=(65536-((FOSC/4)/BAUD));                //初值低8位赋值给TL2寄存器
TH2=(65536-((FOSC/4)/BAUD))>>8;             //初值高8位赋值给TH2寄存器
ES=1;                                       //允许串口中断
EA=1;                                       //CPU允许响应中断请求
SendString(&menu);                      //在串口中断上打印menu的内容
```

串口1通信实例1

```
P46=0;                                  //P4.6端口置0，灯亮
while(1){                               //无限循环
    if(RI==1)                           //如果接收到上位机发送的数据
    {
        c=SBUF;                         //从SBUF缓冲区读数据到变量c
        if(c==0x31)                     //判断如果接收的数据是字符‘1’
            P46=!P46;                   //P4.6取反
        else if(c==0x32)                //判断如果接收的数据是字符‘2’
            P47=!P47;                   //P4.7取反
        else                            //其它任何输入
        {SendString(“\r\n Exit Program”);   //串口上打印Exit Program信息
}}}}
```

串口1通信实例1

下面说明该代码的设计原理和验证方法，步骤包括：

- 使用T2定时器，根据前面给出的IRC的时钟频率为18.432MHz，波特率为115200，由于，T2的溢出率和波特率存在下面的关系，即：

  串口1的波特率=SYSclk/(65535-[RL_TH2,RL_TL2]/4

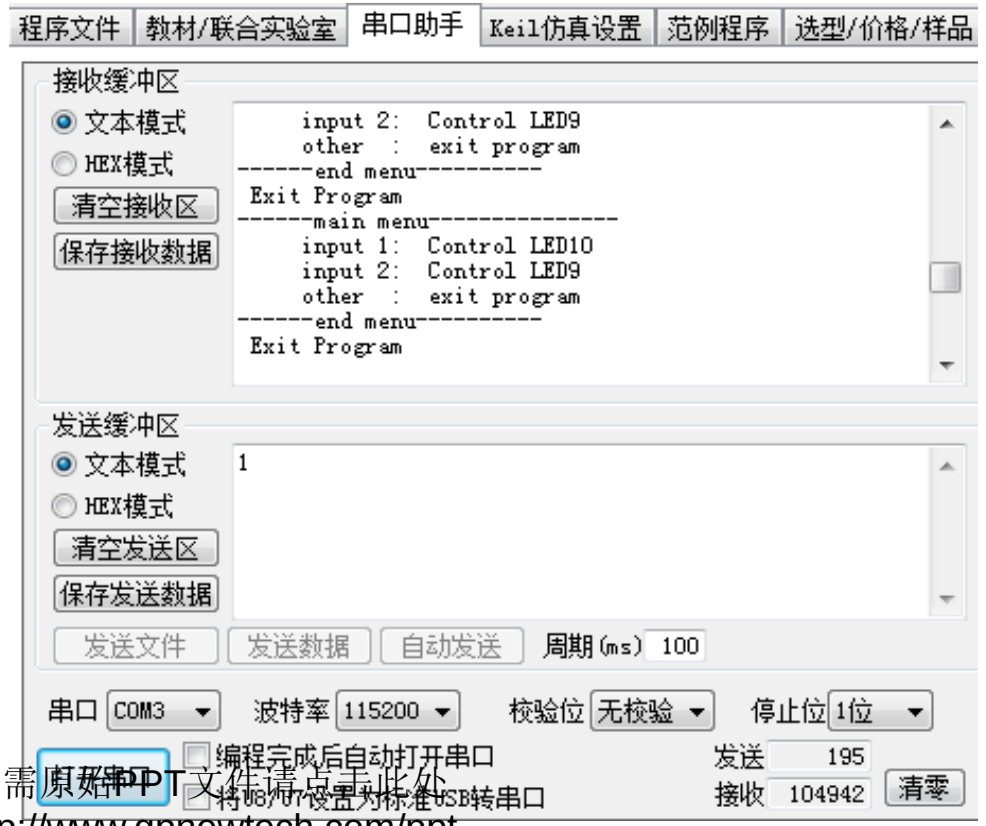
因此，[RL_TH2,RL_TL2]=65536-SYSclk/(串口1波特率×4)

注：RL_TH2是T2H的自动重加载寄存器，RL_TL2是T2L的自动重加载寄存器。

- 打开STC-ISP软件，在该界面内，选择硬件选项。将“输入用户程序运行时的IRC频率设置为18.432MHz。
- 单击下载/编程按钮，按前面的方法下载设计到STC单片机。

串口1通信实例1

- 在STC-ISP软件右侧串口中，选择串口助手标签。在该标签串口界面下，按下面设置参数：
  - 串口：COM3（读者根据自己电脑识别出来的COM端口号进行设置）
  - 波特率：115200。
  - 校验位：无校验。
  - 停止位：1位。



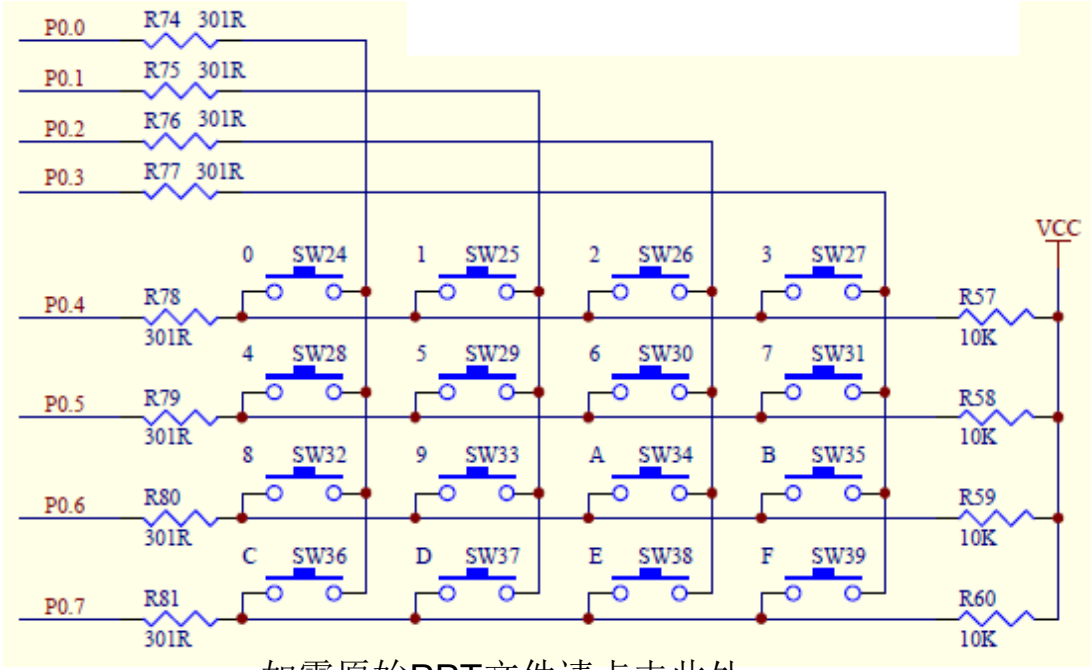
串口1通信实例1

- 单击打开串口按钮。
- 在STC学习板上，找到并按一下SW19按键，重新运行程序。可以看到在上面的接收窗口中，显示出菜单信息。
- 在发送窗口中输入1。
- 单击发送数据按钮。观察LED10的变化。
- 在发送串口中输入2。
- 单击发送数据按钮。观察LED9的变化。
- 在发送串口中输入其它字符。
- 单击发送数据按钮。看到在接收窗口中，显示“Exit Program”提示信息。

串口1通信实例2
---矩阵按键结构及检测原理

矩阵按键结构及检测原理

- 在STC学习板上提供了16个按键，这16个按键按4×4形式排列，即：4行和4列形式。



串口1通信实例2
---矩阵按键结构及检测原理

由图可以判断出在实际中P0.0~P0.3应该为输出，或者逻辑高电平、或者逻辑低电平；而P0.7~0.4为输入，也就是读取P0.7~P0.4引脚的状态。

- 首先如何判断有按键被按下，方法是将P0.0~P0.3引脚拉低，也就是驱动P0.0~P0.3为低。
- 如果16个按键中，没有按下按键，则P0.4、P0.5、P0.6或者P0.7仍然处于上拉状态，即：逻辑高/逻辑1，此时如果读取这四个端口，读取的值应该是1111分别对应于P0.7、P0.6、P0.5、P0.4引脚。
- 只要有一个按键按下，P0.4、P0.5、P0.6或者P0.7就有引脚被拉低，也就是读P0.4、P0.5 、P0.6、 P0.7引脚，它们组合的值一定不等于1111。因此，就可以判断是否有按键被按下。

串口1通信实例2
---矩阵按键结构及检测原理

- 驱动P0.3引脚为低/逻辑0，驱动P0.2、P0.1和P0.0引脚为逻辑1，即它们值的组合为0111，十六进制数7。
  - 当按下标号为0、1、2、4、5、6、8、9、A、C、D、E的按键时，P0.4~P0.7引脚的状态不会发生任何的变化。
  - 如果按下3号按键，则P0.4引脚被拉低，即：变化到逻辑状态0。而其他引脚状态仍然为逻辑高。此时如果读取这四个端口，读取的值应该是1110，十六进制数E分别对应于P0.7、P0.6、P0.5、P0.4引脚。
  - 如果按下7号按键，则P0.5引脚被拉低，即：变化到逻辑状态0。而其他引脚状态仍然为逻辑高。此时如果读取这四个端口，读取的值应该是1101，十六进制数D分别对应于P0.7、P0.6、P0.5、P0.4引脚。

串口1通信实例2
---矩阵按键结构及检测原理

- 如果按下11（B）号按键，则P0.6引脚被拉低，即：变化到逻辑状态0。而其他引脚状态仍然为逻辑高。此时如果读取这四个端口，读取的值应该是1011，十六进制数B分别对应于P0.7、P0.6、P0.5、P0.4引脚。
- 如果按下15（F）号按键，则P0.7引脚被拉低，即：变化到逻辑状态0。而其他引脚状态仍然为逻辑高。此时如果读取这四个端口，读取的值应该是0111，十六进制数7分别对应于P0.7、P0.6、P0.5、P0.4引脚。

串口1通信实例2
---矩阵按键结构及检测原理

- 驱动P0.2引脚为低/逻辑0，驱动P0.3、P0.1和P0.0引脚为1，即它们值的组合为1011，十六进制数B。
  - 当按下标号为0、1、3、4、5、7、8、9、B、C、D、F的按键时，P0.4~P0.7引脚的状态不会发生任何的变化。
  - 如果按下2号按键，则P0.4引脚被拉低，即：变化到逻辑状态0。而其他引脚状态仍然为逻辑高。此时如果读取这四个端口，读取的值应该是1110，十六进制数E分别对应于P0.7、P0.6、P0.5、P0.4引脚。
  - 如果按下6号按键，则P0.5引脚被拉低，即：变化到逻辑状态0。而其他引脚状态仍然为逻辑高。此时如果读取这四个端口，读取的值应该是1101，十六进制数D分别对应于P0.7、P0.6、P0.5、P0.4引脚。

串口1通信实例2
---矩阵按键结构及检测原理

- 如果按下10（A）号按键，则P0.6引脚被拉低，即：变化到逻辑状态0。而其他引脚状态仍然为逻辑高。此时如果读取这四个端口，读取的值应该是1011，十六进制数B分别对应于P0.7、P0.6、P0.5、P0.4引脚。
- 如果按下14（E）号按键，则P0.7引脚被拉低，即：变化到逻辑状态0。而其他引脚状态仍然为逻辑高。此时如果读取这四个端口，读取的值应该是0111，十六进制数7分别对应于P0.7、P0.6、P0.5、P0.4引脚。

串口1通信实例2
---矩阵按键结构及检测原理

- 驱动P0.1引脚为低/逻辑0，驱动P0.3、P0.2和P0.0引脚为1，即它们值的组合为1101，十六进制数D。
  - 当按下标号为0、2、3、4、6、7、8、A、B、C、E、F的按键时，P0.4~P0.7引脚的状态不会发生任何的变化。
  - 如果按下1号按键，则P0.4引脚被拉低，即：变化到逻辑状态0。而其他引脚状态仍然为逻辑高。此时如果读取这四个端口，读取的值应该是1110，十六进制数E分别对应于P0.7、P0.6、P0.5、P0.4引脚。
  - 如果按下5号按键，则P0.5引脚被拉低，即：变化到逻辑状态0。而其他引脚状态仍然为逻辑高。此时如果读取这四个端口，读取的值应该是1101，十六进制数D分别对应于P0.7、P0.6、P0.5、P0.4引脚。

串口1通信实例2
---矩阵按键结构及检测原理

- 如果按下9号按键，则P0.6引脚被拉低，即：变化到逻辑状态0。而其他引脚状态仍然为逻辑高。此时如果读取这四个端口，读取的值应该是1011，十六进制数B分别对应于P0.7、P0.6、P0.5、P0.4引脚。
- 如果按下13（D）号按键，则P0.7引脚被拉低，即：变化到逻辑状态0。而其他引脚状态仍然为逻辑高。此时如果读取这四个端口，读取的值应该是0111，十六进制数7分别对应于P0.7、P0.6、P0.5、P0.4引脚。

串口1通信实例2
---矩阵按键结构及检测原理

- 驱动P0.0引脚为低/逻辑0，驱动P0.3、P0.2和P0.1引脚为1，即它们值的组合为1110，十六进制数E。
  - 当按下标号为1、2、3、5、6、7、9、A、B、D、E、F的按键时，P0.4~P0.7引脚的状态不会发生任何的变化。
  - 如果按下0号按键，则P0.4引脚被拉低，即：变化到逻辑状态0。而其他引脚状态仍然为逻辑高。此时如果读取这四个端口，读取的值应该是1110，十六进制数E分别对应于P0.7、P0.6、P0.5、P0.4引脚。
  - 如果按下4号按键，则P0.5引脚被拉低，即：变化到逻辑状态0。而其他引脚状态仍然为逻辑高。此时如果读取这四个端口，读取的值应该是1101，十六进制数“D”分别对应于P0.7、P0.6、P0.5、P0.4引脚。

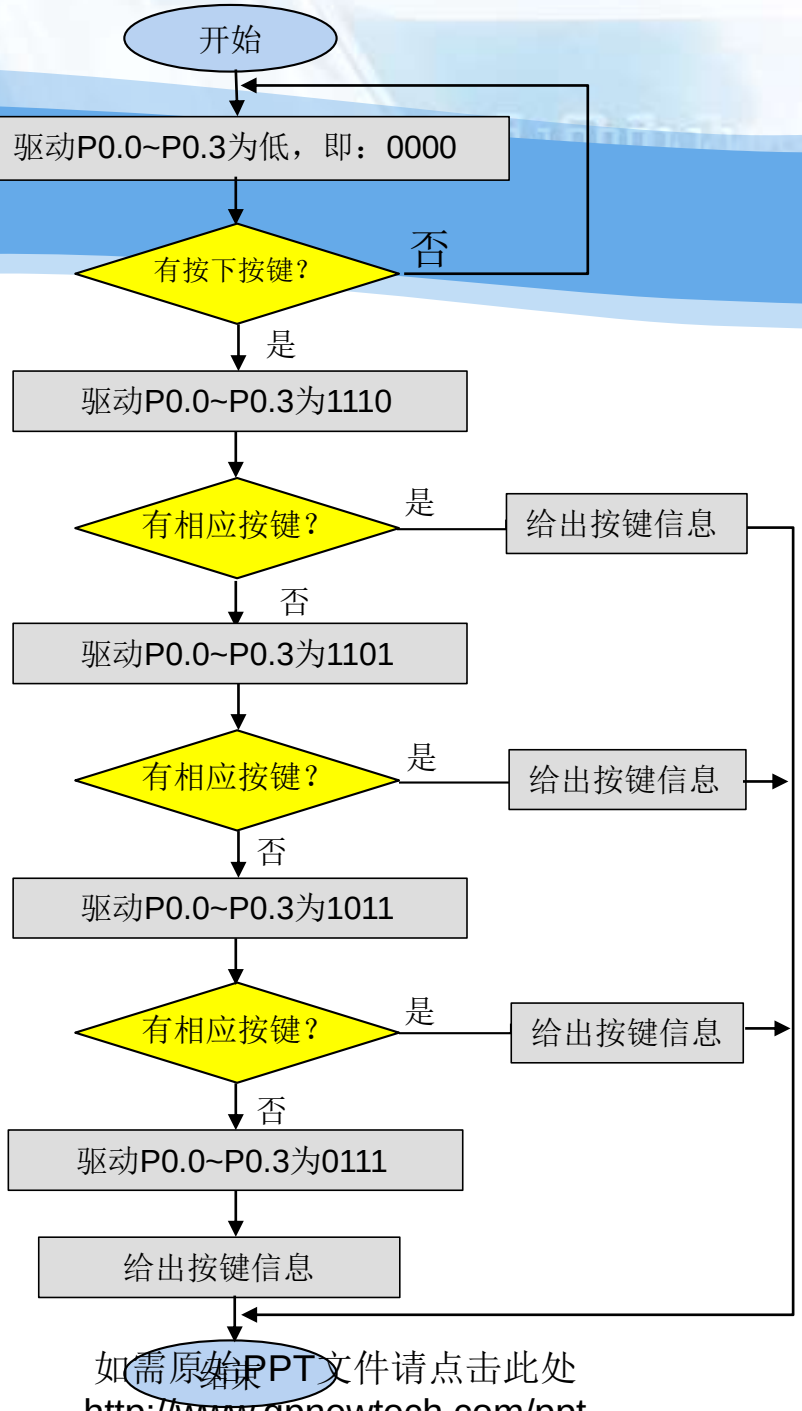
串口1通信实例2
---矩阵按键结构及检测原理

- 如果按下8号按键，则P0.6引脚被拉低，即：变化到逻辑状态0。而其他引脚状态仍然为逻辑高。此时如果读取这四个端口，读取的值应该是1011，十六进制数B分别对应于P0.7、P0.6、P0.5、P0.4引脚。
- 如果按下12（C）号按键，则P0.7引脚被拉低，即：变化到逻辑状态0。而其他引脚状态仍然为逻辑高。此时如果读取这四个端口，读取的值应该是0111，十六进制数7分别对应于P0.7、P0.6、P0.5、P0.4引脚。

串口1通信实例2
---矩阵按键结构及检测原理

所谓的扫描就是让P0.0、P0.1、P0.2和P0.3的驱动值快速的在0111、1011、1101、1110之间进行变化

- 这样就能在按下按键的时候，知道按下那个按键。



串口1通信实例2
---串口1参数设置

串口1参数设置

- 在该设计中串口1工作在模式1下，使用定时器1模式0（16位自动重加载）作为串口1的波特率发生器。

串口1通信实例2
--设计代码和分析

【例】 STC学习板上按键通过串口显示在主机上C语言描述的例子。

```
#include "reg51.h"
#define FOSC  18432000L                    //声明当前单片机主时钟频率
#define BAUD 115200                        //声明波特率常数115200
sfr AUXR  =0x8E;                           //声明AUXR寄存器的地址
bit busy=0;                                //声明bit型变量
xdata char menu[]={"\r\n--Display Press buttons information--\r\n"};
                                           //声明字符数组menu
```

串口1通信实例2
--设计代码和分析

```
void IO_KeyDelay(void)                    //声明IO_KeyDelay子函数，延迟
{
        unsigned char i;
        i = 60;
        while(--i)          ;
}
void SendData(unsigned char dat)          //声明SendData子函数
{
        while(busy);                      //判断是否发送完，没有则等待
        SBUF=dat;                         //否则，将数据dat写入SBUF寄存器
        busy=1;                           //将busy置1
}
```

串口1通信实例2
--设计代码和分析

```
void SendString(char *s)                    //声明SendString子函数
{
        while(*s!='\0')                      //判断是否是字符串的结尾
        SendData(*s++);                      //如果没有结束，调用SendData发送数据
}
void uart1() interrupt 4                    //声明uart串口1中断服务程序
{
        if(RI)                               //通过RI标志，判断是否接收到数据
                RI=0;                        //如果RI为1，则软件清零RI
        if(TI)                               //通过TI标志，判断是否发送完数据
                TI=0;                        //如果TI为1，则软件清零TI
         busy=0;                             //将busy标志清零
}
```

串口1通信实例2
--设计代码和分析

```
void main()
{
    unsigned char c1_new,c1_old=0,c1;        //声明字符型变量
    SCON=0x50;                               //串口1模式1，使能串行接收
    AUXR=0x40;                               //定时器1不分频，作为串口1波特率时钟
    TL1=(65536-((FOSC/4)/BAUD));             //定时器1初值计数器低8位
    TH1=(65536-((FOSC/4)/BAUD))>>8;          //定时器1初值计数器高8位
    TR1=1;                                   //使能定时器1工作
    ES=1;                                    //允许串口1中断
   EA=1;                                     //CPU允许响应中断请求
```

串口1通信实例2
--设计代码和分析

```
SendString(&menu);          //在串口调试界面中打印字符串信息
while(1)
 {                          //无限循环
      P0=0xF0;              //将P0.0~P0.3拉低，在读P0.4~P0.7前，发‘F’
      IO_KeyDelay();        //延迟读
      c1_new=P0&0xF0;       //得到矩阵按键的信息
```

串口1通信实例2
--设计代码和分析

```
if(c1_new!=c1_old)                  //如果新按键和旧按键状态不一样，则继续
    {
        c1_old=c1_new;              //把新按键的状态变量保存作为旧的按键
        if(c1_new!=0xF0)            //如果有按键按下，继续
        {
            P0=0xFE;                //将P0[3-0]置“1110”，在读P0.4~P0.7前，发‘F’
            IO_KeyDelay();          //延迟读
            c1_new=P0;              //获取P0端口的值
            switch (c1_new)
              {
```

串口1通信实例2
--设计代码和分析

```
 case 0xee: c1=0;  break;       //如果值为0xee，则表示按下0号按键
 case 0xde:  c1=4;  break;      //如果值为0xde，则表示按下4号按键
 case 0xbe:  c1=8;  break;      //如果值为0xbe，则表示按下8号按键
 case 0x7e:  c1=12;  break;     //如果值为0x7e，则表示按下12号按键
 default  : ;
 }
P0=0xFD;                  //将P0[3-0]置“1101”，在读P0.4~P0.7前，发‘F’
 IO_KeyDelay();                //延迟读
 c1_new=P0;                    //获取P0端口的值
 switch (c1_new)
  {
```

串口1通信实例2
--设计代码和分析

```
switch (c1_new)
 {
  case 0xed: c1=1; break;        //如果值为0xed，则表示按下1号按键
   case 0xdd: c1=5; break;       //如果值为0xdd，则表示按下5号按键
   case 0xbd: c1=9; break;       //如果值为0xbd，则表示按下9号按键
   case 0x7d: c1=13; break;      //如果值为0x7d，则表示按下13号按键
   default  : ;
 }
 P0=0xFB;                  //将P0[3-0]置“1011”，在读P0.4~P0.7前，发‘F’
 IO_KeyDelay();                  //延迟读
 c1_new=P0;                      //获取P0端口的值
```

串口1通信实例2
--设计代码和分析

```
switch (c1_new)
 {
  case 0xeb: c1=2; break;       //如果值为0xeb，则表示按下2号按键
  case 0xdb: c1=6; break;       //如果值为0xdb，则表示按下6号按键
  case 0xbb: c1=10; break;      //如果值为0xbb，则表示按下10号按键
  case 0x7b: c1=14; break;      //如果值为0x7b，则表示按下14号按键
  default  : ;
 }
P0=0xF7;                  //将P0[3-0]置“0111”，在读P0.4~P0.7前，发‘F’
 IO_KeyDelay();                 //延迟读
 c1_new=P0;                     //获取P0端口的值
```

串口1通信实例2
--设计代码和分析

```
switch (c1_new)
 {
  case 0xe7: c1=3; break;      //如果值为0xe7，则表示按下3号按键
  case 0xd7: c1=7; break;      //如果值为0xd7，则表示按下7号按键
  case 0xb7: c1=11;break;      //如果值为0xb7，则表示按下11号按键
  case 0x77: c1=15; break;     //如果值为0x77，则表示按下15号按键
   default  : ;
 }
 SendString("\r\n press #"); //发送字符串信息
```

串口1通信实例2
--设计代码和分析

```
if(c1<10)                    //如果按键变量小于10，即：0~9
    SendData(c1+0x30);       //转换为对应的ASCII，调用SendData发送
else if(c1==10)              //如果按键值为10
    SendString(“10”);        //调用SendString函数，发送字符串“10”
else if(c1==11)              //如果按键值为11
    SendString(“11”);        //调用SendString函数，发送字符串“11”
else if(c1==12)              //如果按键值为12
    SendString(“12”);        //调用SendString函数，发送字符串“12”
else if(c1==13)              //如果按键值为13
    SendString(“13”);        //调用SendString函数，发送字符串“13”
else if(c1==14)      //如果按键值为14
```

串口1通信实例2
--设计代码和分析

```
                SendString(“14”);      //调用SendString函数，发送字符串“14”
            else if(c1==15)                //如果按键值为15
                SendString(“15”);      //调用SendString函数，发送字符串“15”
                SendString(" button\r\n"); //调用SendString函数，发送字符串
        }
    }
  }
}
```

串口1通信实例2
--设计代码和分析

下面说明该代码的设计原理和验证方法，步骤包括：

- 使用T1定时器，根据前面给出的IRC的时钟频率为18.432MHz，波特率为115200，由于，T1的溢出率和波特率存在下面的关系，即：

  串口1的波特率=SYSclk/(65535-[RL_TH1,RL_TL1]/4

  因此，[RL_TH1,RL_TL1]=65536-SYSclk/(串口1波特率×4)

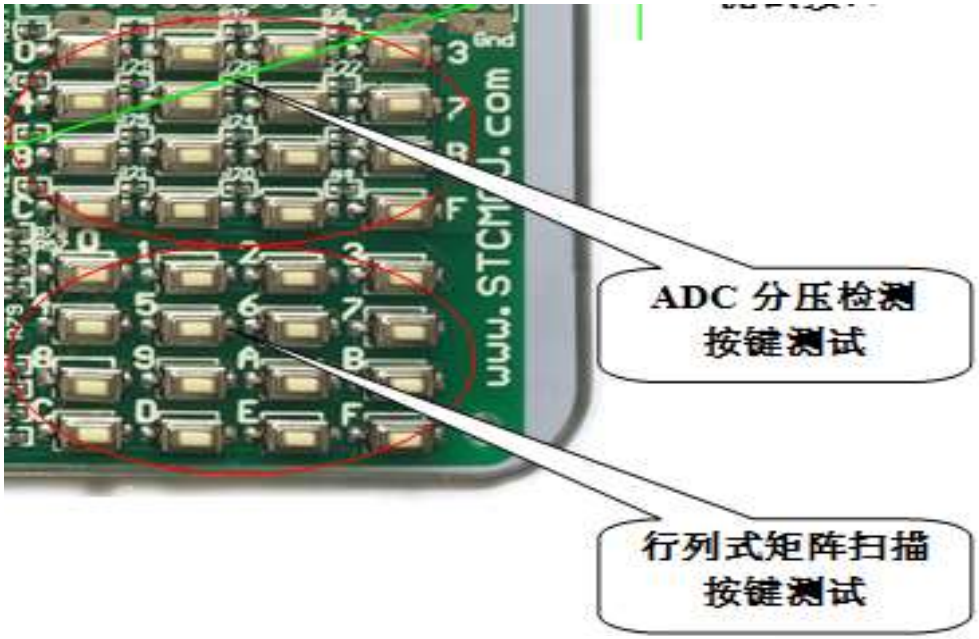
- 打开STC-ISP软件，在该界面内，选择硬件选项。将“输入用户程序运行时的IRC频率设置为18.432MHz。
- 单击下载/编程按钮，按前面的方法下载设计到STC单片机。

串口1通信实例2
--设计代码和分析

- 在STC-ISP软件右侧串口中，选择串口助手标签。在该标签串口界面下，按下面设置参数：
  - 串口：COM3（读者根据自己电脑识别出来的COM端口号进行设置）
  - 波特率：115200。
  - 校验位：无校验。
  - 停止位：1位。
- 单击打开串口按钮。
- 在STC学习板上，找到并按一下SW19按键，重新运行程序。可以看到在上面的接收窗口中，显示出提示信息“—Display Press buttons information—。

串口1通信实例2
--设计代码和分析

- 在STC学习板上右下角的位置，找到矩阵按键。



STC学习板上矩阵按键的位置

串口1通信实例2
--设计代码和分析

- 每次按下一个矩阵键盘中的一个按键，可以看到串口调试助手上显示按键信息。

串口2寄存器及工作模式
--串口2控制寄存器S2CON

串口2控制寄存器S2CON

- 该寄存器位于STC单片机特殊功能寄存器地址为0x9A的位置。
- 当复位后，该寄存器的值为“01000000”。

串口2控制寄存器S2CON各位的含义

比特	B7	B6	B5	B4	B3	B2	B1	B0
名字	S2SM0	1	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI

- S2SM0
  - 该位确定串口2工作模式。当该位为0时，为8位UART，可变波特率模式；当该位为1时，为9位UART，可变波特率模式。

串口2寄存器及工作模式
--串口2控制寄存器S2CON

- **S2SM2**

允许方式1多机通信控制位。如果S2SM2位为1且S2REN位为1时，则接收机处于地址帧选状态。此时可以利用接收到的第9位（即S2RB8）来筛选地址帧：

- 当S2RB8=1时，说明该帧为地址帧，地址信息可以进入S2BUF，并使得S2RI置1，进而在中断服务程序中再进行地址号比较；
- 当S2RB8=0时，说明该帧不是地址帧，应丢掉并保持S2RI=0。

注：（1）在方式1中，如果S2SM2位为0且S2REN位为1，接收机处于禁止筛选地址帧状态。不论收到的S2RB8是否为1，均可使接收到的信息进入S2BUF，并使得S2RI=1，此时S2RB8通常为校验位。

（2）方式0为非多机通信方式。在这种模式下，将S2SM2设置为0。

串口2寄存器及工作模式
--串口2控制寄存器S2CON

- **S2REN**
  - 允许/禁止串口2接收控制位。当该位为1时，允许串行接收状态，可以启动串行接收器RxD2，开始接收信息；当该位为0时，禁止串行接收状态，禁止串行接收器RxD2。
- **S2TB8**
  - 当选择方式1时，该位为要发送的第9位数据，按需要由软件置1或者清0。例如：可用作数据的校验位或者多机通信中表示地址帧/数据帧的标志位。
- **S2RB8**
  - 当选择方式1时，该位为接收到的第9位数据，作为奇偶校验位或者地址帧/数据帧的标志位。

串口2寄存器及工作模式
--串口2控制寄存器S2CON

- **S2TI**
  - 发送中断请求标志位。在停止位开始发送时由S2TI置1，向CPU发出中断请求。

注:当CPU响应中断后，必须由软件将该位清0。

- **S2RI**
  - 接收中断请求标志位。在接收到停止位的中间时刻由S2RI置1，向CPU发出中断请求。

注：当CPU响应中断后，必须由软件将该位清0。

串口2寄存器及工作模式
--串口数据缓冲寄存器

- STC15系列单片机的串口2缓冲寄存器S2BUF地址为0x9B，在该地址实际是两个缓冲寄存器。
  - 一个缓冲寄存器用于保存要发送的数据；
  - 而另一个缓冲寄存器用于读取已经接收到的数据。
- 在串口的串行通道内，设置数据寄存器。
  - 在该串口所有工作模式中，在写入信号S2BUF的控制下，把数据加载到相同的9位移位寄存器中，前面8位为数据字节，最低位为移位寄存器的输出位。
  - 根据所设置的工作模式，自动将1或者S2TB8的值加载到移位寄存器的第9位，并进行发送。

串口2寄存器及工作模式
--串口数据缓冲寄存器

- 在串口的接收寄存器是一个输入移位寄存器。
  - 在方式0和方式1时，字长均为9位。
  - 当接收完一帧数据后，将移位寄存器中的串行字节数据加载到数据缓冲寄存器S2BUF中，将其第9位加载到S2CON寄存器的S2RB8位。如果由于S2SM2使得已经接收到的数据无效时，S2RB8和S2BUF中的内容不变。
- 由于在串行通道内设置了输入移位寄存器和S2BUF缓冲寄存器，从而在接收完一帧串行数据将其从移位寄存器加载到并行S2BUF缓冲寄存器后，可以立即开始接收下一帧数据。

串口2寄存器及工作模式
--中断允许寄存器2

中断允许寄存器IE2

- 该寄存器位于STC单片机特殊功能寄存器地址为0xAF的位置。
- 当复位后，该寄存器的值为x000000。

中断允许寄存器IE2各位的含义

比特	B7	B6	B5	B4	B3	B2	B1	B0
名字	--	ET4	ET3	ES4	ES3	ET2	ESPI	ES2

- ES2
  - 串口2中断允许位。当该位为1时，允许串口2中断；当该位为0时，禁止串口2中断。

串口2寄存器及工作模式
--中断优先级控制寄存器2

中断优先级控制寄存器IP2

- 该寄存器位于STC单片机特殊功能寄存器地址为0xB5的位置。当复位后，该寄存器的值为“xxx00000”。

中断优先级控制寄存器IP2各位的含义

比特	B7	B6	B5	B4	B3	B2	B1	B0
名字	--	--	--	PX4	PPWMFD	PPWM	PSPI	PS2

- PS2
  - 串口2中断优先级控制位。当该位为0时，串口2中断为最低优先级中断（优先级为0）；当该位为1时，串口2中断为最高优先级中断（优先级1）。

串口2寄存器及工作模式
--引脚位置控制寄存器

引脚位置控制寄存器P_SW2。

- 该寄存器位于STC单片机特殊功能寄存器地址为0xBA的位置。
- 当复位后，该寄存器的值为“xxxxx000”。

引脚位置控制寄存器P_SW2各位的含义

比特	B7	B6	B5	B4	B3	B2	B1	B0
名字	--	--	--	--	--	S4_S	S3_S	S2_S

- S4_S
  - 串口4引脚位置选择控制位。
  - 当该位为0时，串口4的引脚位置在P0.2/RxD4和P0.3/TxD4；
  - 当该位为1时，串口4的引脚位置在P5.2/RxD4_2和P5.3/TxD4_2。

串口2寄存器及工作模式
--引脚位置控制寄存器

- **S3_S**
  - 串口3引脚位置选择控制位。
  - 当该位为0时，串口3的引脚位置在P0.0/RxD3和P0.1/TxD3；
  - 当该位为1时，串口3的引脚位置在P5.0/RxD3_2和P5.1/TxD3_2。
- **S2_S**
  - 串口2引脚位置选择控制位。
  - 当该位为0时，串口2的引脚位置在P1.0/RxD2和P1.1/TxD2；
  - 当该位为1时，串口2的引脚位置在P4.6/RxD2_2和P4.7/TxD2_2。

串口2工作模式
--串口2工作模式0

模式0为8位可变波特率UART工作方式。

- **在该模式下，10位数据通过RxD2/P1.0(Rx_D2/ P4.6)接收，通过TxD2/P1.1(Tx_D2/ P4.7)发送。**
- **一帧数据包含：一个起始位、8个数据位和一个停止位。**
- **接收数据时，停止位进入S2CON寄存器的S2RB8位。**
- **波特率由定时器2的溢出率确定。**

串口2工作模式
--串口2工作模式1

模式1为9位可变波特率UART工作方式。

- **在该模式下，11位数据通过RxD2/P1.0(Rx_D2/ P4.6)接收，通过TxD2/P1.1(Tx_D2/ P4.7)发送。**
- **一帧数据包含：一个起始位、8个数据位、一个可编程的第9位和一个停止位。**
  - **发送时，第9位数据来自寄存器S2CON的S2TB8位。**
  - **当接收数据时，第9位进入S2CON寄存器的S2RB8位。**
- **波特率由定时器2的溢出率确定。**

串口3寄存器及工作模式
--串口3控制寄存器S3CON

串口3控制寄存器S3CON。

- 该寄存器位于STC单片机特殊功能寄存器地址为0xAC的位置。
- 当复位后，该寄存器的值为“00000000”。

串口3控制寄存器S3CON各位的含义

比特	B7	B6	B5	B4	B3	B2	B1	B0
名字	S3SM0	S3ST3	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI

- S3SM0
  - 该位确定串口3工作模式。当该位为0时，为8位UART，可变波特率模式；当该位为1时，为9位UART，可变波特率模式。

串口3寄存器及工作模式
--串口3控制寄存器S3CON

- **S3ST3**
  - 串口3选择定时器3作为波特率发生器控制位。当该位为0时，串口3选择定时器2作为其波特率发生器；当该位为1时，串口3选择定时器3作为其波特率发生器。
- **S3SM2**

允许方式1多机通信控制位。如果S3SM2位为1且S3REN位为1时，则接收机处于地址帧选状态。此时可以利用接收到的第9位（即S3RB8）来筛选地址帧：

  - 当S3RB8为1时，说明该帧为地址帧，地址信息可以进入S3BUF，并使得S3RI置1，进而在中断服务程序中再进行地址号比较；
  - 当S3RB8为0时，说明该帧不是地址帧，应丢掉并保持S3RI为0。

串口3寄存器及工作模式
--串口3控制寄存器S3CON

注：（1）在方式1中，如果S3SM2位为0且S3REN位为1，接收机处于禁止筛选地址帧状态。不论收到的S3RB8是否为1，均可使接收到的信息进入S3BUF，并使得S3RI=1，此时S3RB8通常为校验位。

（2）方式0为非多机通信方式。在这种模式下，将S3SM2设置为0。

- **S3REN**
  - 允许/禁止串口3接收控制位。当该位为1时，允许串行接收状态，可以启动串行接收器RxD3，开始接收信息；当该位为0时，禁止串行接收状态，禁止串行接收器RxD3。
- **S3TB8**
  - 当选择方式1时，该位为要发送的第9位数据，按需要由软件置1或者清0。例如：可用作数据的校验位或者多机通信中表示地址帧/数据帧的标志位。

串口3寄存器及工作模式
--串口3控制寄存器S3CON

- **S3RB8**
  - 当选择方式1时，该位为接收到的第9位数据，作为奇偶校验位或者地址帧/数据帧的标志位。
- **S3TI**
  - 发送中断请求标志位。在停止位开始发送时由S3TI置1，向CPU发出中断请求。

  注：当CPU响应中断后，必须由软件将该位清0。
- **S3RI**
  - 接收中断请求标志位。在接收到停止位的中间时刻由S3RI置1，向CPU发出中断请求。

  注：当CPU响应中断后，必须由软件将该位清0。

串口3寄存器及工作模式
--串口数据缓冲寄存器

- STC15系列单片机的串口3缓冲寄存器S3BUF地址为0xAD，在该地址实际是两个缓冲寄存器。
  - 一个缓冲寄存器用于保存要发送的数据；
  - 而另一个缓冲寄存器用于读取已经接收到的数据。
- 在串口的串行通道内，设置数据寄存器。
  - 在该串口所有工作模式中，在写入信号S3BUF的控制下，把数据加载到相同的9位移位寄存器中，前面8位为数据字节，最低位为移位寄存器的输出位。根据所设置的工作模式，自动将1或者S3TB8的值加载到移位寄存器的第9位，并进行发送。

串口3寄存器及工作模式
--串口数据缓冲寄存器

- 在串口的接收寄存器是一个输入移位寄存器。
  - 在方式0和方式1时，字长均为9位。当接收完一帧数据后，将移位寄存器中的串行字节数据加载到数据缓冲寄存器S3BUF中，将其第9位加载到S3CON寄存器的S3RB8位。如果由于S3SM2使得已经接收到的数据无效时，S3RB8和S3BUF中的内容不变。
  - 由于在串行通道内设置了输入移位寄存器和S3BUF缓冲寄存器，从而在接收完一帧串行数据将其从移位寄存器加载到并行S3BUF缓冲寄存器后，可以立即开始接收下一帧数据。

串口3寄存器及工作模式
--中断允许寄存器2

中断允许寄存器IE2

- 该寄存器位于STC单片机特殊功能寄存器地址为0xAF的位置。
- 当复位后，该寄存器的值为“x0000000”。

中断允许寄存器IE2各位的含义

比特	B7	B6	B5	B4	B3	B2	B1	B0
名字	--	ET4	ET3	ES4	ES3	ET2	ESPI	ES2

- ES3
  - 串口3中断允许位。当该位为1时，允许串口3中断；当该位为0时，禁止串口3中断。

串口3工作模式
--串口3工作模式0

模式0为8位可变波特率UART工作方式。

- **在该模式下，10位数据通过RxD3/P0.0(RxD3_2/ P5.0)接收，通过TxD3/P0.1(TxD3_2/ P5.1)发送。**
- **一帧数据包含：一个起始位、8个数据位和一个停止位。**
- **接收数据时，停止位进入S3CON寄存器的S3RB8位。**
- **波特率由定时器2或者定时器3的溢出率确定。**

串口3工作模式
--串口3工作模式1

模式1为9位可变波特率UART工作方式

- 在该模式下，11位数据通过RxD3/P0.0(RxD3_2/ P5.0)接收，通过TxD3/P0.1(TxD3_2/ P5.1)发送。
- 一帧数据包含：一个起始位、8个数据位、一个可编程的第9位和一个停止位。
  - 发送时，第9位数据来自特殊功能寄存器S3CON的S3TB8位。
  - 当接收数据时，第9位进入S3CON寄存器的S3RB8位。
- 波特率由定时器2/3的溢出率确定。

串口4寄存器及工作模式
--串口4控制寄存器S4CON

串口4控制寄存器S4CON

- 该寄存器位于STC单片机特殊功能寄存器地址为0x84的位置。
- 当复位后，该寄存器的值为“00000000”。

串口4控制寄存器S4CON各位的含义

比特	B7	B6	B5	B4	B3	B2	B1	B0
名字	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI

- S4SM0
  - 该位确定串口4工作模式。当该位为0时，为8位UART，可变波特率模式；当该位为1时，为9位UART，可变波特率模式。

串口4寄存器及工作模式
--串口4控制寄存器S4CON

- **S4ST4**
  - 串口4选择定时器4作为波特率发生器控制位。当该位为0时，串口4选择定时器2作为其波特率发生器；当该位为1时，串口4选择定时器4作为其波特率发生器。
- **S4SM2**

允许方式1多机通信控制位。如果S4SM2位为1且S4REN位为1时，则接收机处于地址帧选状态。此时可以利用接收到的第9位（即S4RB8）来筛选地址帧：

  - 当S4RB8位为1时，说明该帧为地址帧，地址信息可以进入S4BUF，并使得S4RI置1，进而在中断服务程序中再进行地址号比较；
  - 当S4RB8位为0时，说明该帧不是地址帧，应丢掉并保持S4RI=0。

串口4寄存器及工作模式
--串口4控制寄存器S4CON

注：（1）在方式1中，如果S4SM2位为0且S4REN位为1，接收机处于禁止筛选地址帧状态。不论收到的S4RB8是否为1，均可使接收到的信息进入S4BUF，并使得S4RI=1，此时S4RB8通常为校验位。

（2）方式0为非多机通信方式。在这种模式下，将S4SM2设置为0。

- **S4REN**
  - 允许/禁止串口4接收控制位。当该位为1时，允许串行接收状态，可以启动串行接收器RxD4，开始接收信息；当该位为0时，禁止串行接收状态，禁止串行接收器RxD4。
- **S4TB8**
  - 当选择方式1时，该位为要发送的第9位数据，按需要由软件置1或者清0。例如：可用作数据的校验位或者多机通信中表示地址帧/数据帧的标志位。

串口4寄存器及工作模式
--串口4控制寄存器S4CON

- **S4RB8**
  - 当选择方式1时，该位为接收到的第9位数据，作为奇偶校验位或者地址帧/数据帧的标志位。
- **S4TI**
  - 发送中断请求标志位。在停止位开始发送时，由S4TI置1，向CPU发出中断请求。

注：当CPU响应中断后，必须由软件将该位清0。

- **S4RI**
  - 接收中断请求标志位。在接收到停止位的中间时刻由S4RI置1，向CPU发出中断请求。

注：当CPU响应中断后，必须由软件将该位清0。

串口4寄存器及工作模式
--串口数据缓冲寄存器

- STC15系列单片机的串口4缓冲寄存器S4BUF地址为0x85，在该地址实际是两个缓冲寄存器。
  - 一个缓冲寄存器用于保存要发送的数据；
  - 而另一个缓冲寄存器用于读取已经接收到的数据。
- 在串口的串行通道内，设置数据寄存器。
  - 在该串口所有工作模式中，在写入信号S4BUF的控制下，把数据加载到相同的9位移位寄存器中，前面8位为数据字节，最低位为移位寄存器的输出位。根据所设置的工作模式，自动将1或者S4TB8的值加载到移位寄存器的第9位，并进行发送。

串口4寄存器及工作模式
--串口数据缓冲寄存器

- 在串口的接收寄存器是一个输入移位寄存器。
  - 在方式0和方式1时，字长均为9位。当接收完一帧数据后，将移位寄存器中的串行字节数据加载到数据缓冲寄存器S4BUF中，将其第9位加载到S4CON寄存器的S4RB8位。如果由于S4SM2使得已经接收到的数据无效时，S4RB8和S4BUF中的内容不变。
  - 由于在串行通道内设置了输入移位寄存器和S4BUF缓冲寄存器，从而在接收完一帧串行数据将其从移位寄存器加载到并行S4BUF缓冲寄存器后，可以立即开始接收下一帧数据。

串口4寄存器及工作模式
--中断允许寄存器2

中断允许寄存器IE2

- 该寄存器位于STC单片机特殊功能寄存器地址为0xAF的位置。
- 当复位后，该寄存器的值为“x0000000”。

中断允许寄存器IE2各位的含义

比特	B7	B6	B5	B4	B3	B2	B1	B0
名字	--	ET4	ET3	ES4	ES3	ET2	ESPI	ES2

- ES4
  - 串口4中断允许位。当该位为1时，允许串口4中断；当该位为0时，禁止串口4中断。

串口4工作模式
--串口4工作模式0

模式0为8位可变波特率UART工作方式。

- **在该模式下，10位数据通过RxD4/P0.2(RxD4_2/ P5.2)接收，通过TxD4/P0.3(TxD4_2/ P5.3)发送。**
- **一帧数据包含：一个起始位、8个数据位和一个停止位。**
- **接收数据时，停止位进入S4CON寄存器的S4RB8位。**
- **波特率由定时器2/4的溢出率确定。**

串口4工作模式
--串口4工作模式1

模式1为9位可变波特率UART工作方式。

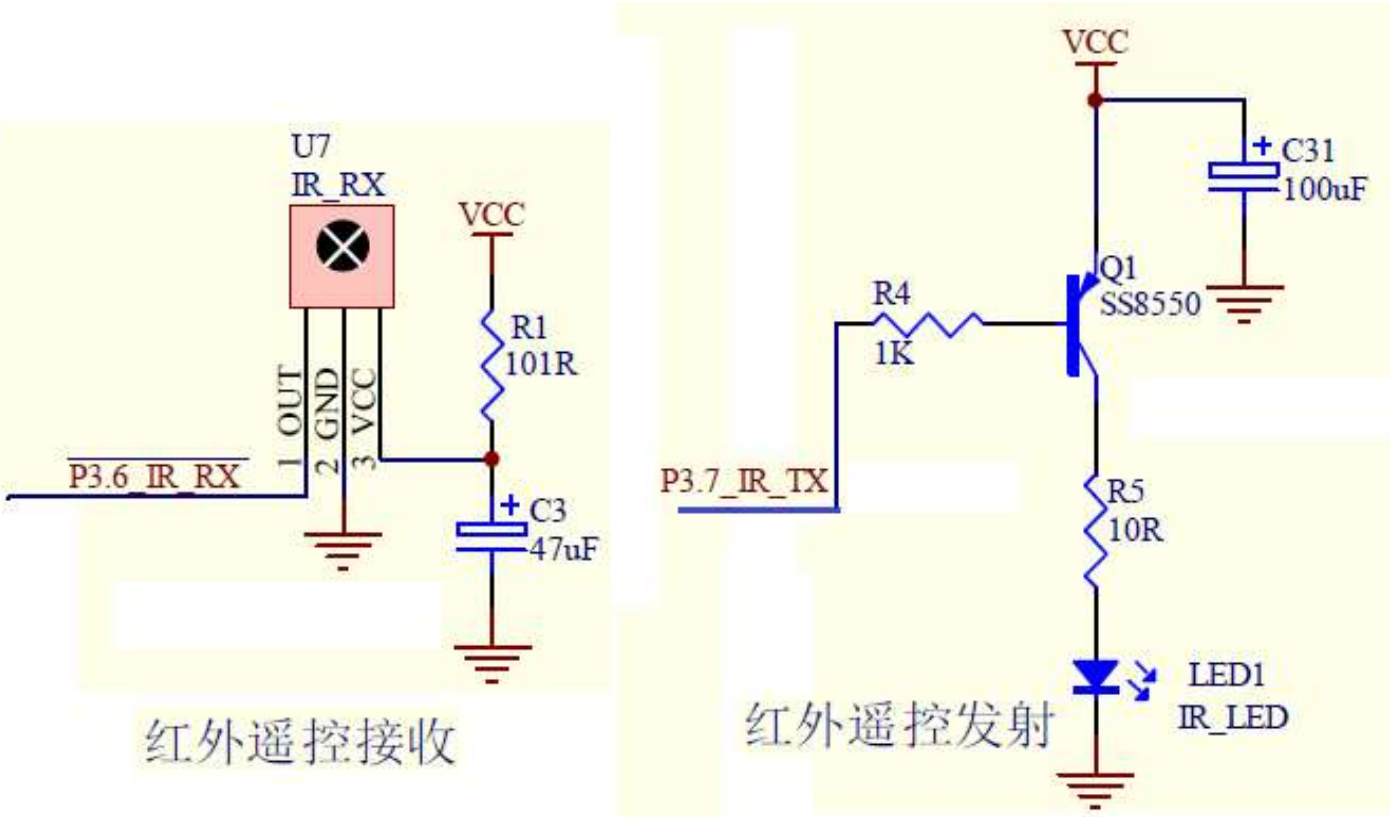
- **在该模式下，11位数据通过RxD4/P0.2(RxD4_2/ P5.2)接收，通过TxD4/P0.3(TxD4_2/ P5.3)发送。**
- **一帧数据包含：一个起始位、8个数据位、一个可编程的第9位和一个停止位。**
  - **发送时，第9位数据来自特殊功能寄存器S4CON的S4TB8位。**
  - **当接收数据时，第9位进入S4CON寄存器的S4RB8位。**
- **波特率由定时器2或者定时器4的溢出率确定。**

串行通信综合实现
--红外收发器的电路原理

在STC提供的学习板上集成了一个红外发生器和红外接器

- 红外发射器和我们所见过的发光二极管外形很像，但是红外发射器发出的光是不可见的红外光。

串行通信综合实现
--红外收发器的电路原理



串行通信综合实现
--红外收发器的电路原理

在STC学习板中，在IAP15W4K58S4芯片的P3.7引脚处，连接了红外遥控发射器。

- 其本质就是由一个PNP的晶体管构成的放大电路，用于发射红外线的二极管通过R5与PNP晶体管的发射极连接。
- 此外，用于接收红外线通信信息的红外接收器连接到单片机的P3.6引脚处。该接收器将红外光携带的信息，转换成电信号，通过P3.6引脚传给单片机进行处理。

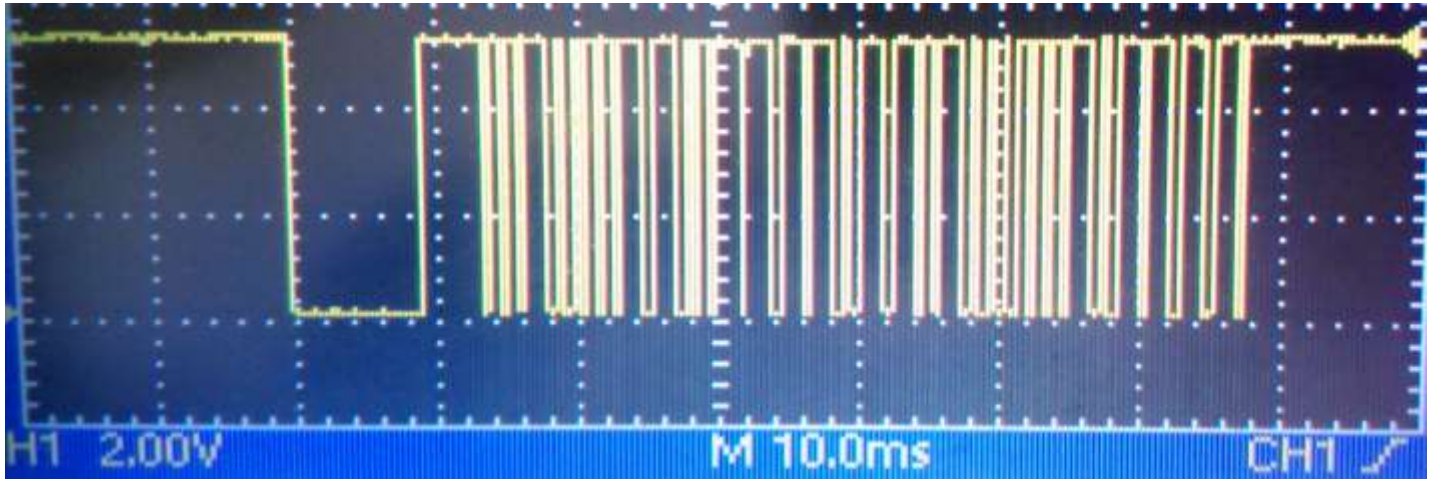
串行通信综合实现
--红外通信波形捕获

通过实验捕获红外通信所传输的信息的步骤包括：

- **给STC学习板上电。**
- **打开示波器。将示波器的一个探头插到STC学习板J9插座标记为P36的插孔中。**
- **找一个用于控制家里电视用的遥控器。**

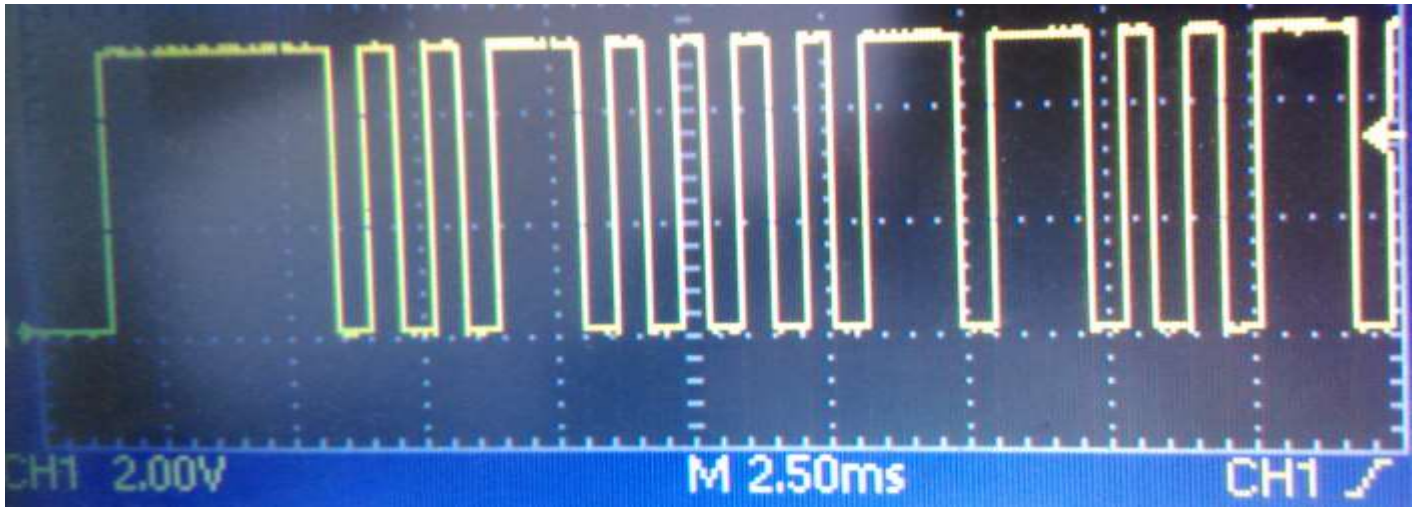
串行通信综合实现
--红外通信波形捕获

- 将遥控器对准红外接收器，并按下红外遥控器的按键。同时，让示波器捕获波形。



串行通信综合实现
--红外通信波形捕获

- 将波形前部进行放大，如图所示。



串行通信综合实现
--红外通信协议

本节将介绍红外通信协议，包括：

- 在红外发射器一侧，为了使红外线在无线传输的过程中避免受到其他红外信号的干扰，通常是将逻辑0和逻辑1调制在某一特定频率的载波上，然后经过红外发光二极管发射出去。
- 在红外接收器一侧，接收到这个被调制后的信号。在本设计中，将通过单片机对接收到的红外信号进行解调，即:去掉载波信号，恢复出原始的二进制脉冲码。

串行通信综合实现
--红外通信波形捕获

红外通信调制方法

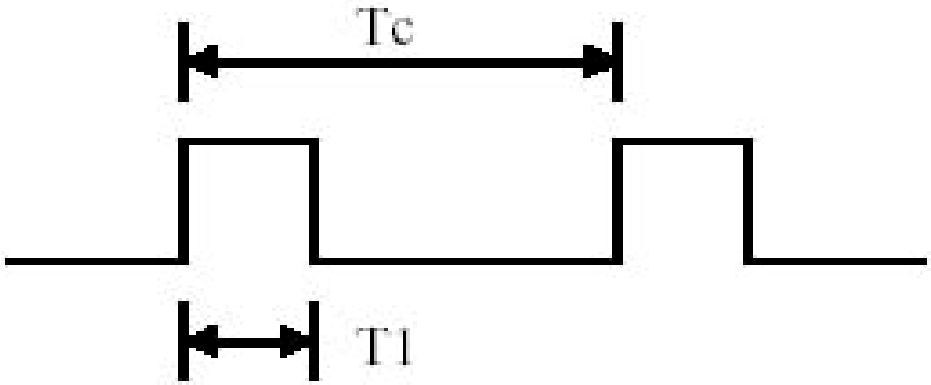
- 脉冲宽度调制（Pluse Width Modulation，PWM）
- 脉冲位置调制（Pulse Position Modulation, PPM）

红外遥控中使用的基带通信协议的类型很多，大概有几十种，常用的就有ITT协议、NEC协议、Sharp协议、Philips协议等。

红外通信协议
--红外发射数据

载波波形

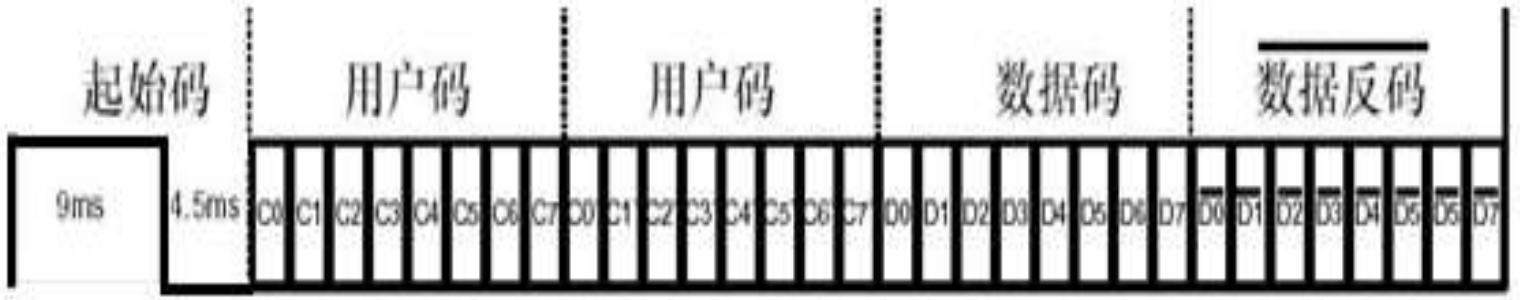
- 可以使用455KHz晶体，经内部分频电路的12分频，将信号调制在37.91kHz，占空比为1/3。



红外通信协议
--红外发射数据

数据格式

- 数据格式包括了起始码、用户码、数据码和数据码反码。
  - 数据反码是对数据码取反后的编码，编码时可用于对数据的纠错。
- 在该数据格式中，编码（包括16位用户码、8位数据码和8位数据反码）长度总共32位。



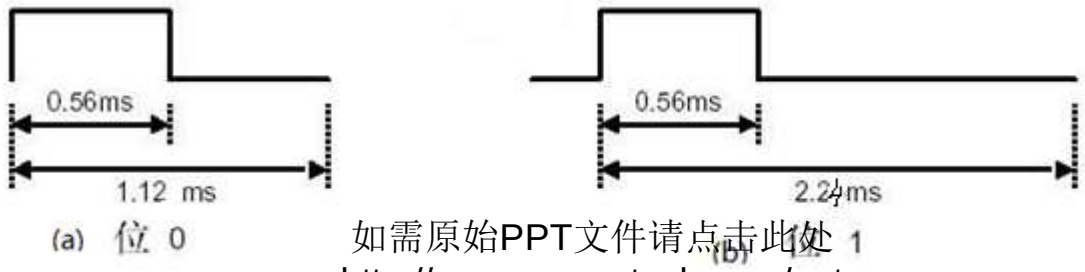
红外通信协议
--红外发射数据

- 红外发射的波形中最前面的是起始码。
  - 起始码的前半部分为高电平，时长大约为9ms；
  - 后半部分为空闲低电平，时长大约为4.5ms。

红外通信协议
--红外发射数据

位定义

- 用户码或数据码中的每一个二进制位或者是1，或者是0。通过脉冲的时间间隔来区分它们。因此，这种编码方式称为PPM调制方式。
  - 对于逻辑0来说，前面的高电平周期为0.56ms，后面的低电平周期大约为0.56ms的空闲时刻。
  - 对于逻辑1来说，前面的高电平周期为0.56ms，后面的低电平周期大约为1.68ms的空闲时刻。



红外通信协议
--红外发射数据

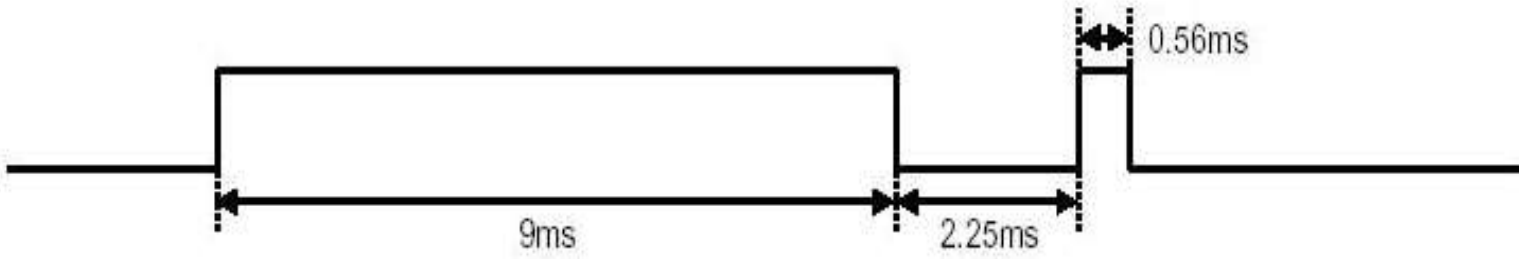
根据对位的定义，得到：

- **16位地址码的最短持续时间为：1.12×16=18ms ；最长持续时间为：2.24ms×16=36ms。**
- **8位数据码和8位数据反码的总时间恒定为：（1.12ms+2.24ms）×8=27ms。**

因此，所有32位码的持续时间在45~63ms。

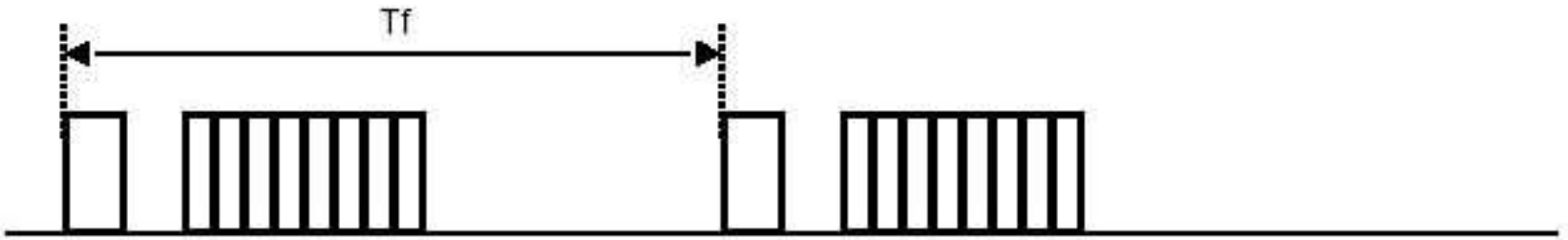
红外通信协议
--按键输出波形

重复码



红外通信协议
--按键输出波形

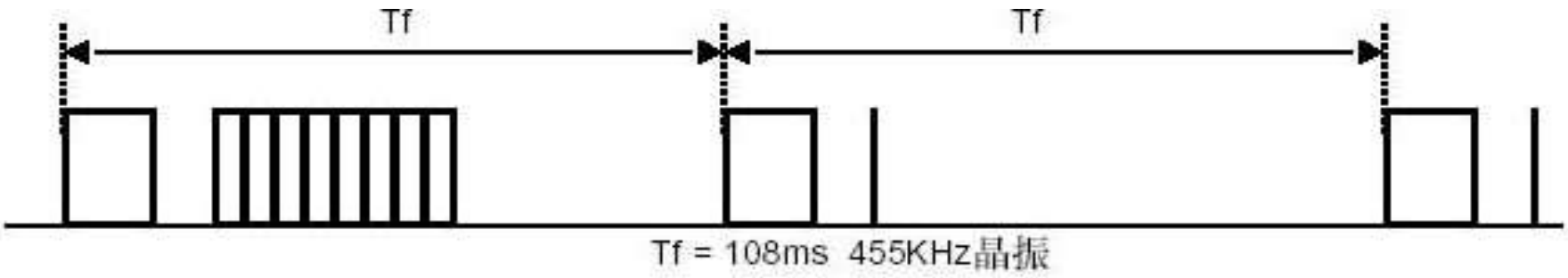
单一按键波形



Tf = 108ms 455KHz晶振

红外通信协议
--按键输出波形

连续按键波形



红外通信协议
--红外接收数据

- 红外接收头将38K载波信号过虑，接收到的波形刚好与发射波形相反
  - 前导码以低电平开始，持续9ms；然后维持4.5ms的高电平。

红外通信协议
--红外接收数据

解码的关键是如何识别0和1

- 从位的定义可以知道，在接收时，0和1均以 0.56ms的低电平开始，不同的是高电平的宽度不同，
  - 对于0来说，持续0.56ms；
  - 对于1来说，持续1.68ms。
  - 所以，必须根据接收信号的高电平时间长度来区分0和1。
- 如果从0.56ms低电平过后，只持续0.56ms的高电平，则为0；如果持续1.68ms的高电平，则为1。

注：根据码的格式，应该等待起始码结束后才能读码。

串行通信综合实现
--红外检测原理

检测红外传输信息的目的是为了获得四个字节的数据，包括：

- 二个字节的用户码；
- 一个字节的数据码；
- 一个字节的数据反码。

串行通信综合实现
--红外检测原理

从硬件设计可以知道，红外接收端接到了P3.6引脚，该引脚也是INT2中断触发引脚的位置。

- 在INT_CLKO（AUXR2）寄存器中的EX2就是外部中断允许位。
  - 当该位为1时，允许中断；
  - 否则，禁止中断。
  - 中断2必须采用下降沿触发的方式。

串行通信综合实现
--红外检测原理

该该设计中，码型特征很明显。

- 对于单一按键来说，只要区分引导码，逻辑0和逻辑1。它们的持续时间有很大的不同。
  - 可以考虑使用定时器通过判断时间的边界来区分它们。
  - 在该设计中，使用定时器/计数器0的模式1（自动16位重加载模式）。
- 红外传输信息的检测在中断2服务程序中实现。
  - 在程序中，一个关键的地方就是设置判决条件。在该设计中，使用的定时0作为判决计数条件。

串行通信综合实现
--红外检测原理

- 当P3.6为0时，表示低电平，启动定时器0一直计数，以此获得低电平的持续时间。
  - 计数器0的时钟是Sysclk/12，系统时钟频率在烧写程序到STC单片机的时候设置为6.000MHz，这是考虑到了计数器的范围是16位，计数范围是0~65535。在每个时钟沿时，计数器0加1。计算机公式为：

    时间长度=(12× Sysclk)/计数值[TH0*256+TL0]

注：在设计中，考虑时钟的误差，将时间长度设置在一个合理的范围内。

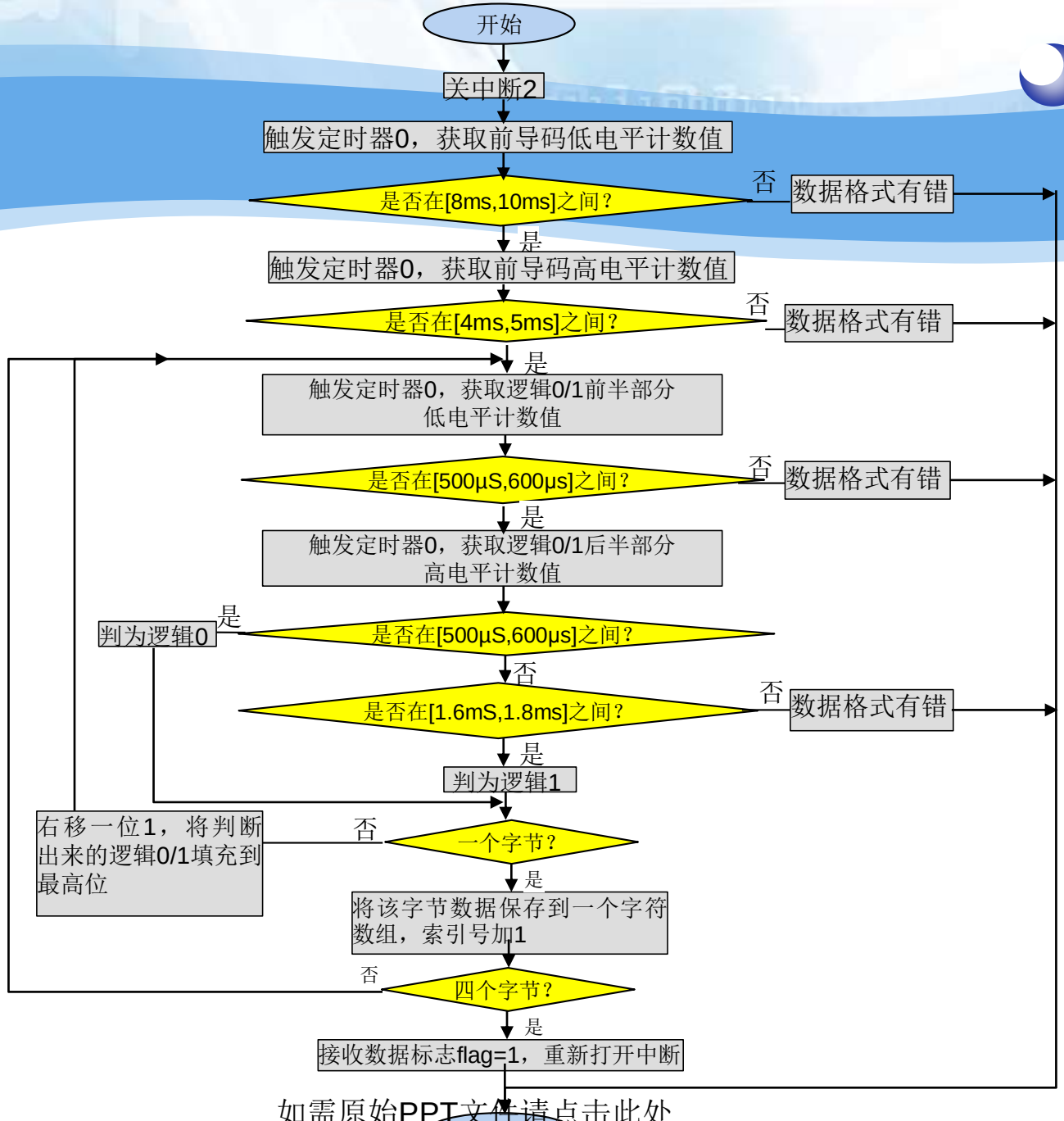
串行通信综合实现
--红外检测原理

- 当P3.6为1时，表示高电平，启动定时器0一直计数，以此获得高电平的持续时间。
  - 计数器0的时钟是Sysclk/12，系统时钟频率在烧写程序到STC单片机的时候设置为6.000MHz，这是考虑到了计数器的范围是16位，计数范围是0~65535。在每个时钟沿时，计数器0加1。计算机公式为：

    时间长度=(12× Sysclk)/计数值[TH0*256+TL0]

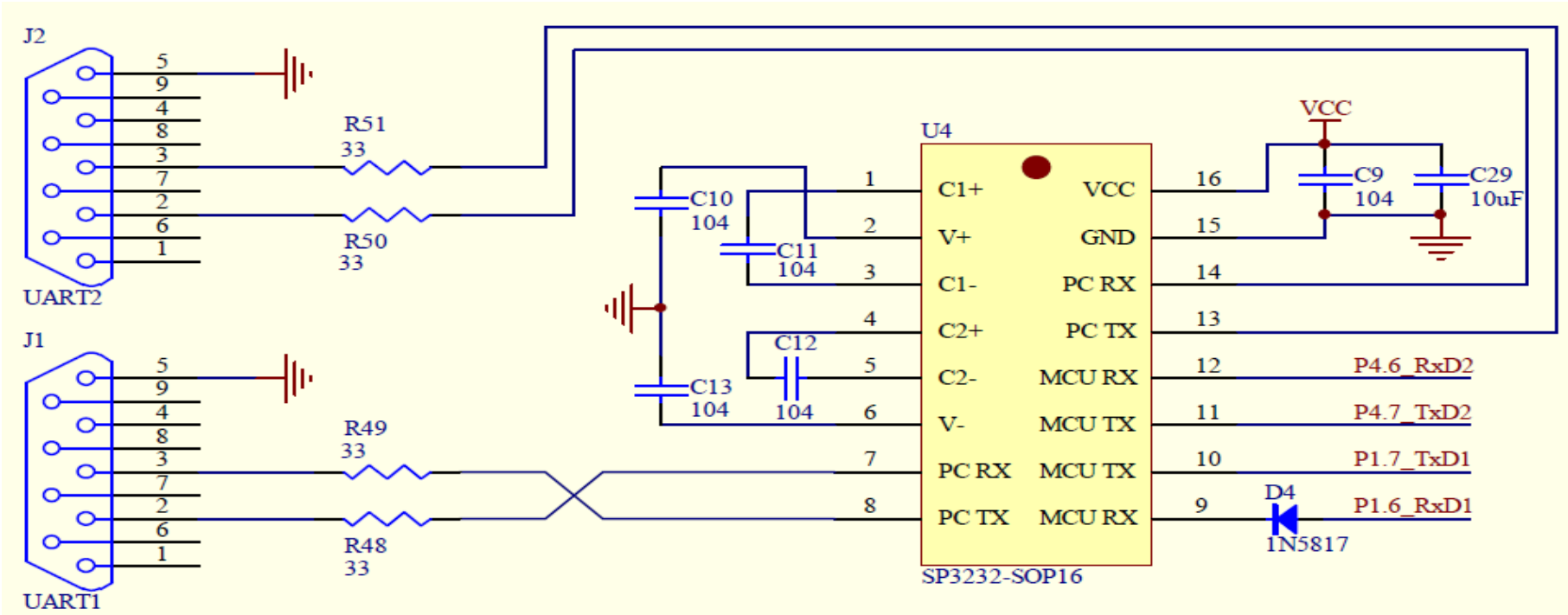
注：在设计中，考虑时钟的误差，将时间长度设置在一个合理的范围内。

综合上述，最终的门限是通过确定时间长度范围后，通过上面公式得到计数值的范围来作为实际的判断条件。


开始
关中断2
触发定时器0，获取前导码低电平计数值
是否在[8ms,10ms]之间？
否
数据格式有错
是
触发定时器0，获取前导码高电平计数值
是否在[4ms,5ms]之间？
否
数据格式有错
是
触发定时器0，获取逻辑0/1前半部分
低电平计数值
是否在[500μS,600μs]之间？
否
数据格式有错
是
触发定时器0，获取逻辑0/1后半部分
高电平计数值
是否在[500μS,600μs]之间？
是
判为逻辑0
否
是否在[1.6mS,1.8ms]之间？
否
数据格式有错
是
判为逻辑1
一个字节？
否
右移一位1，将判断出来的逻辑0/1填充到最高位
是
将该字节数据保存到一个字符数组，索引号加1
四个字节？
否
是
接收数据标志flag=1，重新打开中断
结束

串行通信综合实现
--串口通信原理

在该设计中使用了串口2作为STC单片机和PC机/笔记本电脑通信的接口



串行通信综合实现
--串口通信原理

在该设计中，使用了标识为J2的串口

- 串口2接收和发送信号分别连接到STC单片机IAP15W4K32S4单片机的P4.6/RxD2和P4.7/TxD2引脚上。
- 经过SP3232芯片转换成RS-232电平标准，连接到9针的UART母头连接器上。

串行通信综合实现
--串口通信原理实现

【例】 STC学习板上红外遥控数据通过串口2显示在主机上C语言描述的例子。

```
#include "reg51.h"
#include "intrins.h"
#define FOSC  6000000L          //声明单片机主时钟频率6MHz，为了后面计算
#define BAUD 115200             //声明串口通信的波特率115200，为了后面计算
#define S2RI 0x01               //定义S2RI的值
#define S2TI 0x02               //定义S2TI的值
sfr AUXR  =0x8E;                //声明AUXR寄存器的地址0x8E
sfr AUXR1 =0xA2;                //声明AUXR1寄存器的地址0xA2
```

串行通信综合实现
--串口通信原理实现

```
sfr AUXR2 =0x8F;          //声明AUXR2寄存器的地址0x8F
sfr TH2   =0xD6;          //声明TH2寄存器的地址0xD6
sfr TL2   =0xD7;          //声明TL2寄存器的地址0xD7
sfr S2CON =0x9A;          //声明S2CON寄存器的地址0x9A
sfr S2BUF =0x9B;          //声明S2BUF寄存器的地址0x9B
sfr P3M1  =0xB1;          //声明P3M1寄存器的地址0xB1
sfr P3M0  =0xB2;          //声明P3M0寄存器的地址0xB2
sfr P_SW2 =0xBA;          //声明P_SW2寄存器的地址0xBA
sfr IE2   =0xAF;          //声明IE2寄存器的地址0xAF
sbit P36  =P3^6;          //声明P3.6引脚为P36
bit busy=0;               //声明busy变量为bit
```

串行通信综合实现
--串口通信原理实现

```
unsigned char irdata[4]={0,0,0,0};          //声明数组irdata
                                            //保存红外解码的4字节数据
bit flag=0;                                 //声明flag变量为bit
void SendData(unsigned char dat)            //声明串口2发送数据函数SendData
{       while(busy);                        //如果busy为1，表示忙，则一直等待
        S2BUF=dat;                         //往串口2数据缓冲寄存器S2BUF写数据
        busy=1; }                           //置busy为1
void SendString(char *s)                    //声明串口2发送字符串函数SendString
{       while(*s!= '\0' )                   //判断字符串是否结束
        SendData(*s++);                    //如果没有结束，则调用SendData函数
}
```

串行通信综合实现
--串口通信原理实现

```
unsigned int high_level_time()                    //声明检测红外发送数据的
{                                                 //高电平持续时间函数
        TL0=0;                                    //置定时器0初值低8位寄存器TL0为0
        TH0=0;                                    //置定时器0初值高8位寄存器TH0为0
        TR0=1;                                    //启动定时器0开始计数
        while(P36==1)                  //如果读取P3.6的输入为1，一直继续，否则退出
        {
          if(TH0>=0xEE)                 //如果计数时间太长，系统有问题，退出循环
            break;
         }
        TR0=0;                          //如果读取P3.6的输入为0，则停止定时器0计数
        return(TH0*256+TL0);                      //返回计数计数器0的计数值
}
```

串行通信综合实现
--串口通信原理实现

```
unsigned int low_level_time()                    //声明检测红外发送数据的
{                                                //低电平持续时间函数
    TL0=0;                                       //置定时器0初值低8位寄存器TL0为0
    TH0=0;                                       //置定时器0初值高8位寄存器TH0为0
    TR0=1;                                       //启动定时器0开始计数
    while(P36==0)                          //如果读取P3.6的输入为0，一直继续，否则退出
    {
      if(TH0>=0xEE)                        //如果计数时间太长，系统有问题，退出循环
       break;
    }
    TR0=0;                                 //如果读取P3.6的输入为1，则停止定时器0计数
    return(TH0*256+TL0);                   //返回计数计数器0的计数值
}
```

串行通信综合实现
--串口通信原理实现

```
void int2() interrupt 10                    //声明外部中断2的服务程序
{
    unsigned char i,j;                      //定义无符号字符变量i，j
    unsigned int count=0;                   //定义无符号整型变量count
    unsigned char dat=0;                    //定义无符号字符变量dat
    AUXR2&=0x00;                            //关闭外部中断2，即：禁止中断
   count=low_level_time();                  //读取低电平的计数值
   if(count<4000 || count>5000)             //如果不在给定范围内，退出中断服务程序
    {
        return;
    }
```

串行通信综合实现
--串口通信原理实现

```
count=high_level_time();                    //读取高电平的计数值
    if(count<2000 || count>2500)            //如果不在给定范围内，退出中断服务程序
    return;
    //下面开始处理32位数据
    for(i=0;i<4;i++)                        //四个字节的循环处理
     { P36=1;                               //读取P3.6引脚前，需要置P3.6为高
        dat=0;                              //dat赋值为0
          for(j=0;j<8;j++)                  //8个比特的循环处理
```

串行通信综合实现
--串口通信原理实现

```
if(count<250 || count>300)          //如果不在给定范围内，则退出中断服务程序
    return;
count=high_level_time();            //读取高电平的计数值，即：逻辑位高后半部分
if(count>250 && count<300)          //如果在逻辑“0”的范围内，填充0
dat>>=1;
else if(count>800 && count<1000) //如果在逻辑“1”的范围内
 {
    dat>>=1;                        //右移一位
    dat|=0x80; }                    //高位用“1”填充
    else return;                    //否则，不在给定逻辑位高后半部分范围，退出
 }
```

串行通信综合实现
--串口通信原理实现

```
    {
        count=low_level_time();    //读取低电平的计数值
        irdata[i]=dat;
    }                              //将8位数据保存在irdata数组当前索引号
  flag=1;                          //当四字节填满后，将flag置1，表示有数据
  AUXR2|=0x10;                     //开中断
}
```

串行通信综合实现
--串口通信原理实现

```
void uart2() interrupt 8                //声明串口2中断服务程序
{
    if(S2CON & S2RI)                    //如果S2CON的S2RI为1，表示接收到数据
        S2CON&=~S2RI;                   //将S2CON寄存器的S2RI标志清零
    if(S2CON & S2TI)                    //如果S2CON的S2TI为1，表示发送完数据
    {
        S2CON&=~S2TI;                   //将S2CON寄存器的S2TI标志清零
        busy=0;                         //将busy标志清零
     }
}
```

串行通信综合实现
--串口通信原理实现

```
void main()                          //定义主程序
{    unsigned char k;                //定义无符号的字符变量k
     P36=1;                          //设置P3.6引脚为高
     P3M1=0x00;                      //将P3端口设置为准双向弱上拉
     P3M0=0x00;                      //通过P3M1和P3M0寄存器，设置P3端口模式
     TMOD=0x00;                      //设置定时器0的工作模式
     S2CON=0x50;                     //设置串口2方式0，使能串口2接收
     AUXR=0x14;                      //定时器2/12，启动定时器2，定时器模式
     P_SW2|=0x01;                    //串口2切换到P4.6/RxD2_2和P4.7/TxD2_2
     TL2=(65536-((FOSC/4)/BAUD));   //计数初值低8位给定时器2TL2寄存器
     TH2=(65536-((FOSC/4)/BAUD))>>8;//计数初值高8位给定时器2TH2寄存器
```

串行通信综合实现
--串口通信原理实现

```
IE2|=0x01;                                  //允许串口2中断
AUXR2|=0x10;                                //允许外部中断2
EA=1;                                       //CPU允许响应中断请求
SendString("\r\n--begin-----\r\n");         //打印信息
while(1)                                    //无限循环
{
    if(flag==1)                             //如果flag为1，表示接收到四字节解码信息
    {
        flag=0;                             //将flag位置0
        SendString("\r\n--Received IR data is-----\r\n");  //打印信息
```

串行通信综合实现
--串口通信原理实现

```
        for(k=0;k<4;k++)            //循环四次
        SendData(irdata[k]);        //打印4字节，一共32位的信息
        SendString("\r\n");         //打印回车换行符号
        AUXR2|=0x10;                //使能外部中断2
    }
}
}
```

串行通信综合实现
--串口通信原理实现

下面说明该代码的设计原理和验证方法，步骤包括：

- 准备一根UART-USB的串口电缆，将STC学习板上标记为J2的串口插座和电脑主机进行连接。
- 打开STC-ISP软件，在该界面内，选择硬件选项。将“输入用户程序运行时的IRC频率”设置为6.000MHz。
- 在STC-ISP软件右侧串口中，选择串口助手标签。在该标签串口界面下，选择COM7，波特率为115200，一个停止位，无奇偶校验。
- 在接收缓冲区标题栏下，选中HEX模式前面的复选框。

串行通信综合实现
--串口通信原理实现

- 单击打开串口按钮。
- 单击下载/编程按钮，按前面的方法下载设计到STC单片机。
- 将红外遥控器对准STC红外接收器，按一下按键，出现按键信息。

```
0D 0A 2D 2D 62 65 67 69 6E 2D 2D 2D 2D 2D 0D 0A 0D 0A
2D 2D 52 65 63 65 69 76 65 64 20 49 52 20 64 61 74 61
20 69 73 2D 2D 2D 2D 2D 0D 0A 84 79 14 EB 0D 0A
```

- 第一行是没有按遥控器时，给出的提示信息；
- 根据设计代码，可以看到在最后的回车换行符的前面4个字节的数字“84 79 14 EB”就是解码的红外遥控器的数据。该数据14和EB是取反关系，也就是协议规定的用户和用户反码。