# 第12章 STC单片机ADC原理及实现

何宾

2015.02

# 本章主要内容

- 模数转换器原理
- STC单片机内ADC的结构原理
- STC单片机内ADC寄存器组
- ADC应用实现1
- ADC应用实现2
- ADC应用实现3
- ADC应用实现4

# 模数转换器原理

数模转换器（Analog to Digital Converter，ADC），简称为A/D。它用于将连续的模拟信号转换为数字形式离散信号。

- 典型地，ADC将模拟信号转换为与电压值成比例表示的数字离散信号。
- 不同厂商所提供的ADC，其输出的数字信号输出使用不同的编码格式。

# 模数转换器原理
# --模数转换器的参数

介绍ADC转换器中几个重要的参数，包括：

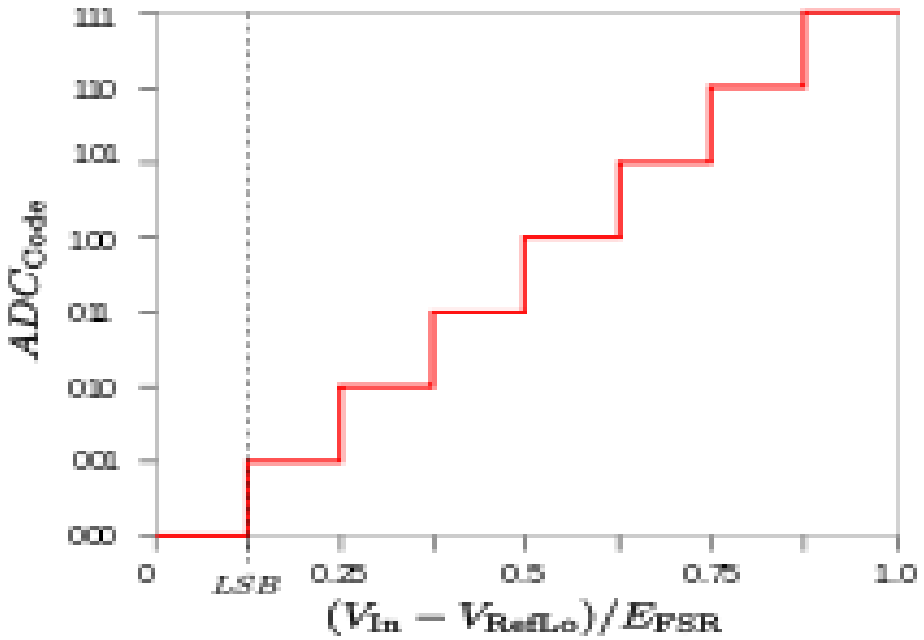
- 分辨率
- 响应类型
- 误差
- 采样率

# 模数转换器的参数
# --分辨率

分辨率是指对于所允许输入的模拟信号范围，它能输出离散数字信号值的个数。

- 这些输出的信号值通用二进制数来表示。因此，分辨率经常用比特作为单位，且这些离散值的个数是2的幂次方。
  - 例如，一个具有8位分辨率的模拟数字转换器可以将模拟信号编码成256个不同的离散值（离散梯度），其范围可以是0~255（即无符号整数）或从 - 128~127（即带符号整数）。至于使用哪一种编码格式，则取决于所选用的ADC器件。

# 模数转换器的参数
# --分辨率



# 模数转换器的参数
# --分辨率

## 分辨率也可以用电气性质来描述

- 比如：使用伏特。使得输出离散信号产生一个变化所需的最小输入电压的差值被称作最低有效位（Least Significant Bit，LSB电压。
  - 这样，模拟数字转换器的分辨率Q等于LSB电压。模拟数字转换器的电压分辨率由下面等式确定：

$$Q = \frac{V_{\mathrm{RefHi}} - V_{\mathrm{RefLow}}}{2^N}$$

  - $V_{RefHi}$和$V_{RefLow}$是转换过程允许输入到ADC的电压上限和下限值。
  - N是模拟数字转换器输出数字量的位数宽度值，以比特为单位。

# 模数转换器的参数
# --分辨率

- 很明显，如果输入电压的变化小于Q值，则ADC是无法分辨出电压的变化。这样，就带来量化误差。
  - 当N值越大，也就是ADC输出数字量的位数越多，则Q越小，即：可分辨的电压变化就越小，分辨能力就越强，量化导致的误差就越小。

# 模数转换器的参数
# --响应类型

**大多数模拟数字转换器的响应类型为线性。**

- **这里的线性是指，输出信号的大小与输入信号的大小成线性比例。**
  - **一些早期的转换器的响应类型呈对数关系，由此来执行A律算法或μ律算法编码。**
- **在一个ADC器件中，没有绝对的线性，只是近似的线性。所以，就会带来线性误差。**
  - **一般情况下，在ADC器件可表示数字量的中间部分线性度较好，而两端线性度较差。**

# 模数转换器的参数
# --误差

模拟数字转换器的误差有若干种来源。

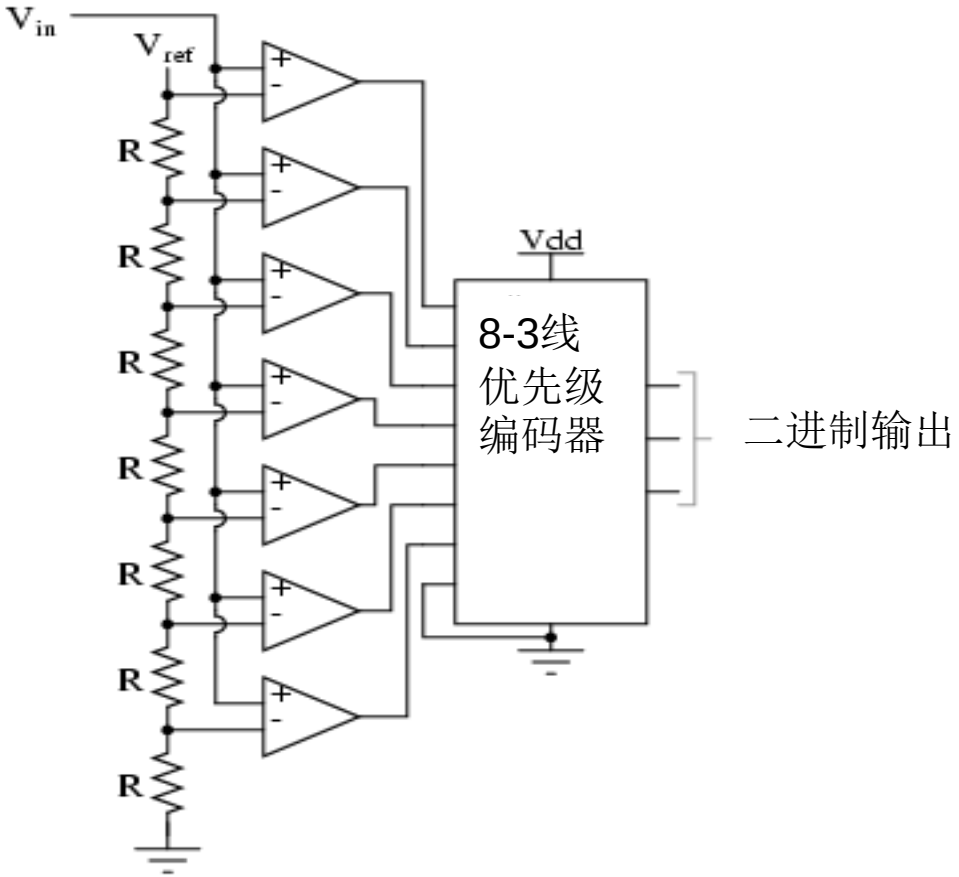
- 量化误差和非线性误差（假设这个模拟数字转换器标称具有线性特征）是任何模拟数字转换中都存在的内在误差。

# 模数转换器的参数
## --采样率

模拟信号在时域上是连续的，因此可以将它转换为时间上离散的一系列数字信号。因此，要求定义一个参数来表示获取模拟信号上每个值并表示成数字信号的速度。通常将这个参数称为ADC的采样率或采样频率。

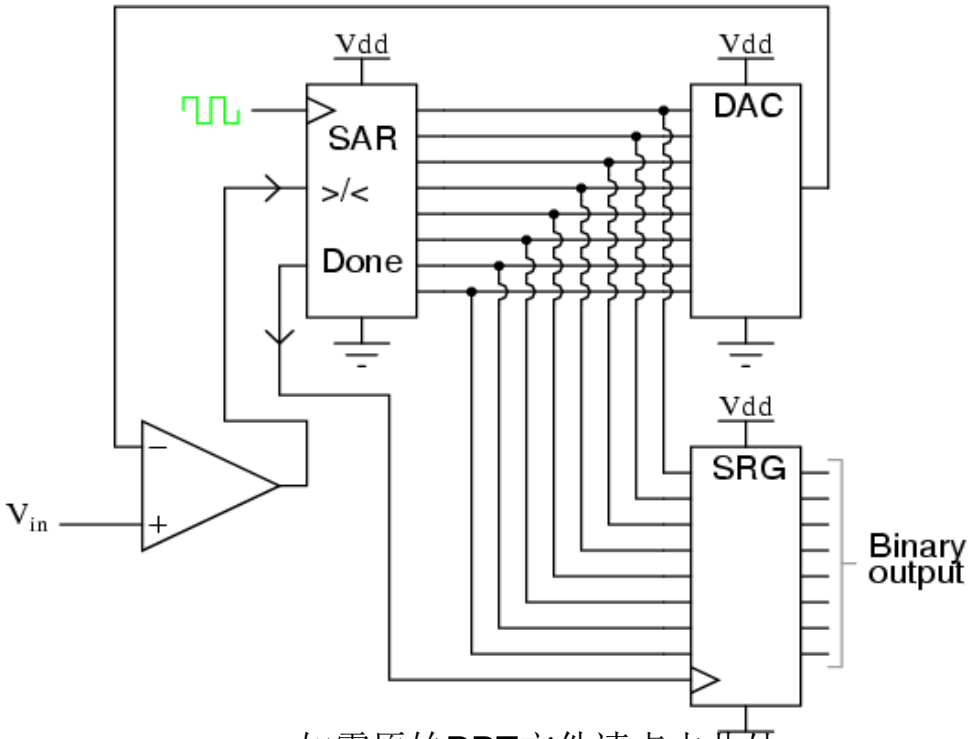
- 根据奈奎斯特采样定理，当采样频率大于所采样模拟信号最高频率的两倍时，信号才不会发生混叠失真。
  - 在实际使用时，为了能更加逼真地恢复出原始的模拟信号，建立采样频率为信号最高频率至少5~10倍。
  - 为了满足采样定理的要求，通常在信号进入ADC之前，要对信号进行带限，即：将信号限制在一个有限的频率范围内。

# 模数转换器的类型
# --Flash ADC

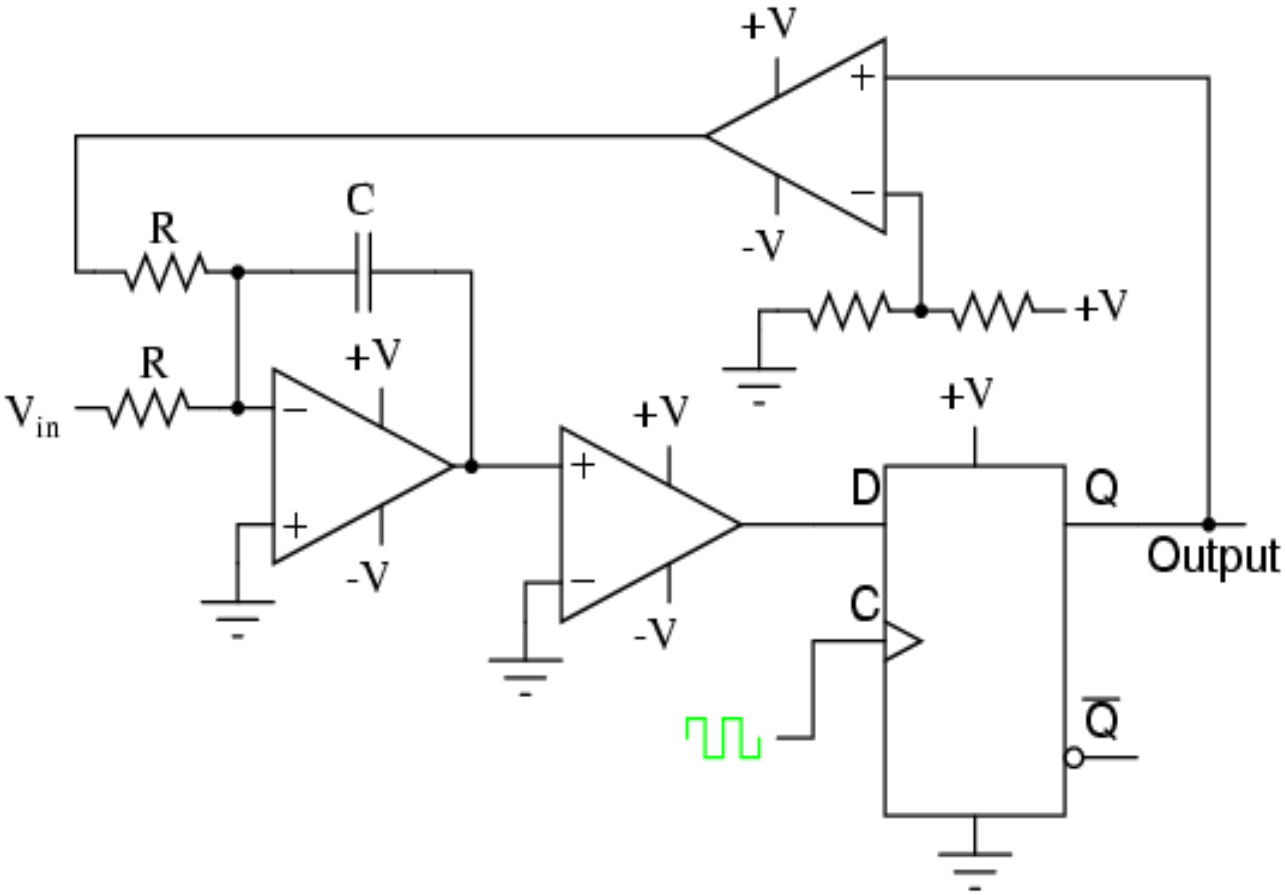


# 模数转换器的类型
# --逐次逼近寄存器型ADC

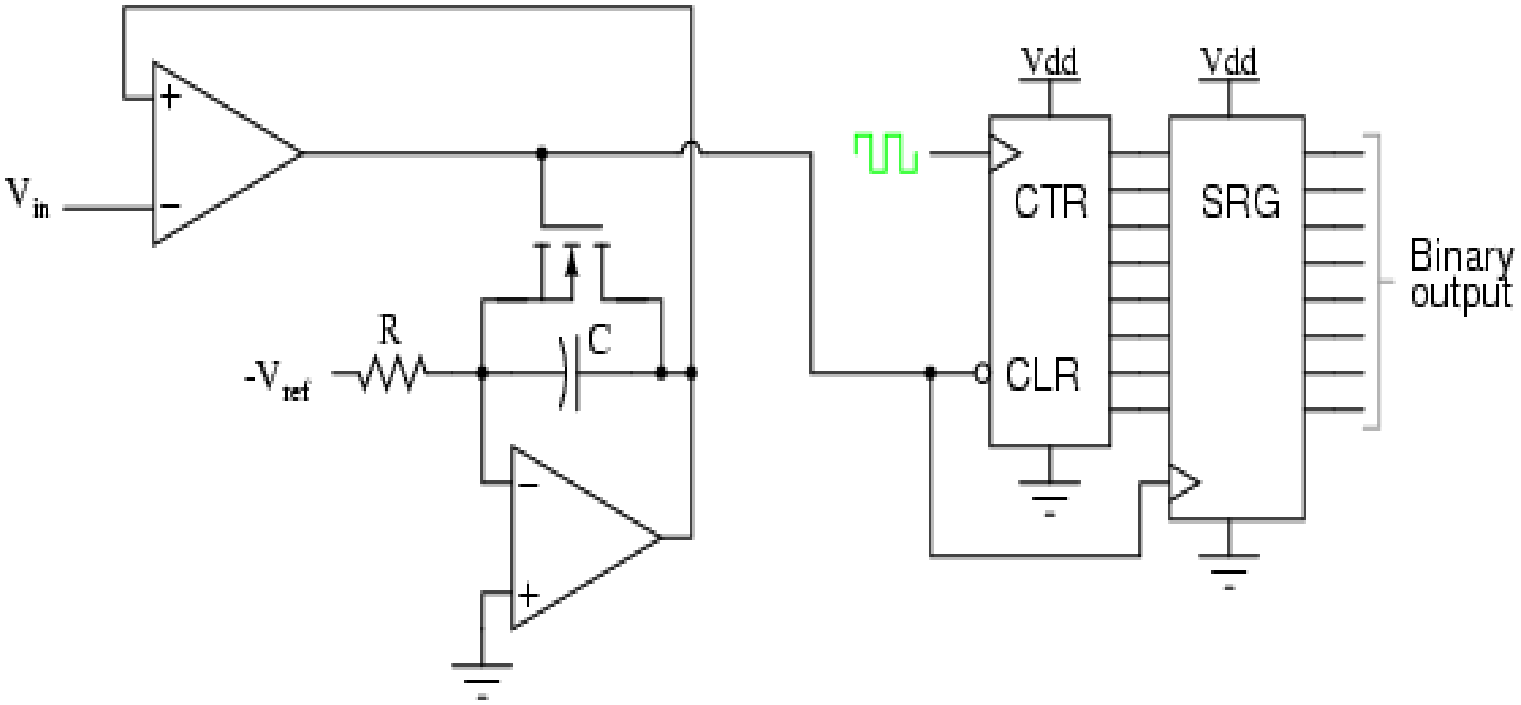
## 逐次逼近寄存器型（Successive Approximation Register，SAR）ADC结构



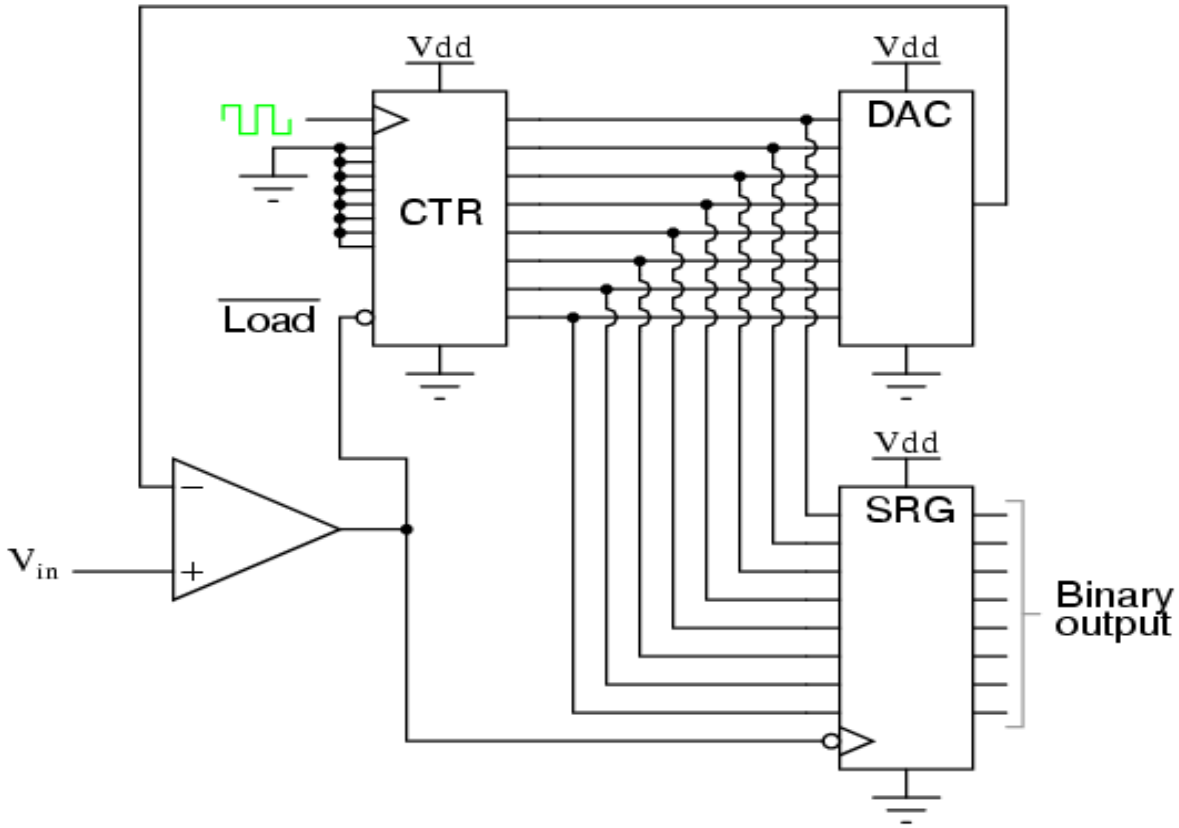
# 模数转换器的类型
# --Σ-Δ ADC（Sigma-delta ADC）



# 模数转换器的类型
# --积分型ADC



# 模数转换器的类型
## --数字跃升型ADC



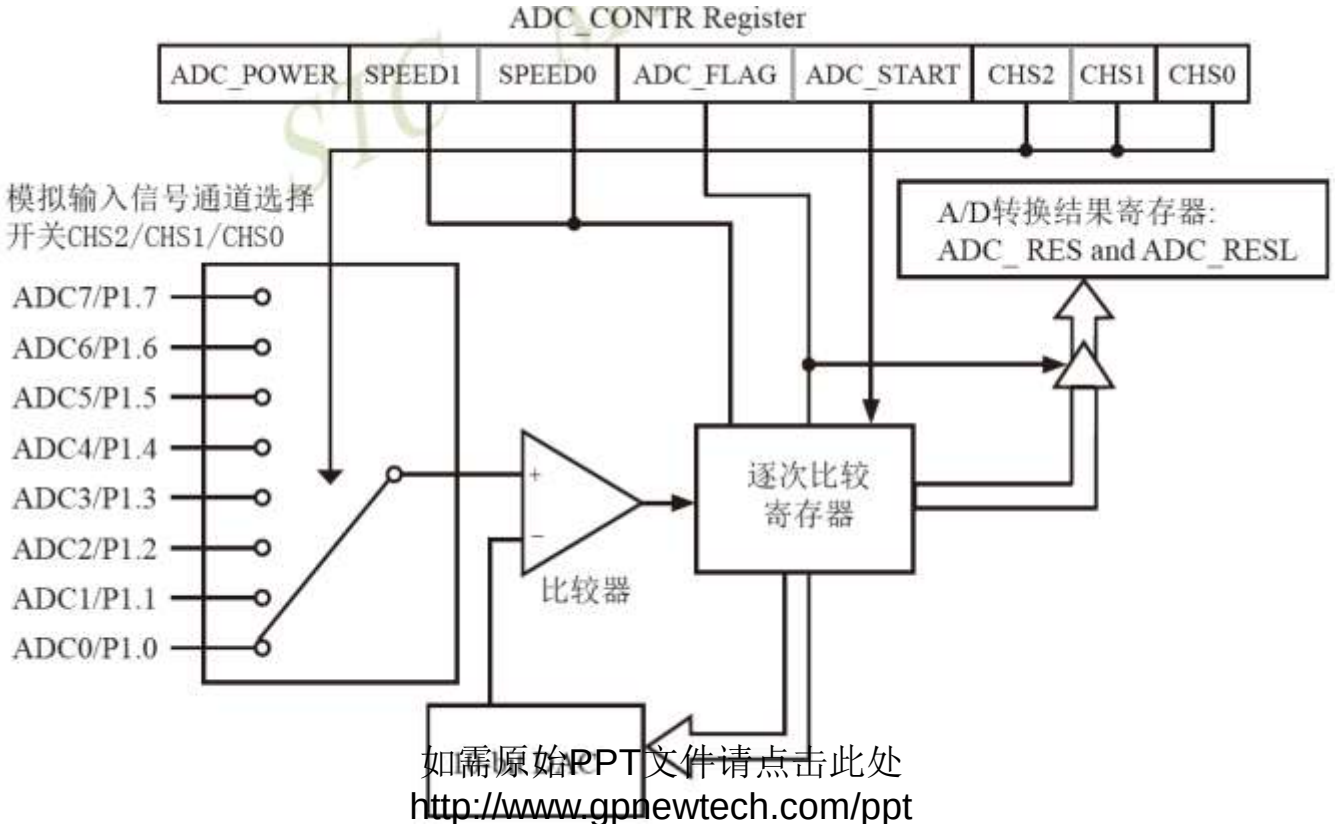
# STC单片机内ADC的结构原理
## --STC单片机内ADC的结构

**STC15系列单片机内集成了8路10位高速ADC转换器模块**

- **通过ADC控制寄存器ADC_CONTR中SPEED1和SPEED0比特位的控制，该ADC模块的最高采样速率可以达到300KHz，即：30万次采样/秒（30kSPS，30k Sample Per Second）。**

# STC单片机内ADC的结构原理
## --STC单片机内ADC的结构

- STC15系列单片机的ADC由多路选择开关、比较器、逐次比较寄存器、10位DAC、转换结果寄存器ADC_RES和ADC_RESL以及ADC控制寄存器ADC_CONTR构成。



# STC单片机内ADC的结构原理
## --STC单片机内ADC的结构

该ADC是典型的SAR结构，这种结构是一种典型的闭环反馈系统。

- 在该ADC的前端提供了一个8通道的模拟多路复用开关。
  - 在ADC控制寄存器ADC_CONTR内的CHS2~CHS0比特位的控制下，将ADC0~ADC7的模拟信号送给比较器。

# STC单片机内ADC的结构原理
## --STC单片机内ADC的结构

- 该结构的ADC包含一个比较器和DAC，通过逐次比较逻辑，从最高有效位MSB开始，顺序地对每一个输入电压与内置DAC输出进行比较。
  - 经过多次比较后，使的转换得到的数字量逼近输入模拟信号所对应的数字量的值。
  - 将最终得到的数字量保存在ADC转换结果寄存器ADC_RES和ADC_RESL中。
  - 同时，将ADC控制寄存器ADC_CONTR中的转换结束标志ADC_FLAG置1，以供程序查询或者向CPU发出中断请求。

# STC单片机内ADC的结构原理
## --ADC转换结果的计算方法

在STC 15系列的单片机中，通过CLK_DIV寄存器的ADRJ位的设置，控制转换结果的计算方式

- 当ADRJ=0时
  - 如果取10位计算结果时，转换结果表示为：

    $(ADC\_RES[7:0],ADC\_RESL[1:0])=1024\times V_{in}/V_{cc}$
  - 如果取8位计算结果

    $ADC\_RES[7:0]=256\times V_{in}/V_{cc}$
- 当ADRJ=1时

  $(ADC\_RES[1:0],ADC\_RESL[7:0])=1024\times V_{in}/V_{cc}$
  - Vin为模拟输入通道输入电压。
  - Vcc为单片机的供电电压

# STC单片机内ADC寄存器组
## --P1口模拟功能控制寄存器

STC15系列单片机的8路模拟信号的输入端口设置在P1端口的8个引脚上，即：P1.0~P1.8。

- 当上电复位后，P1口设置为弱上拉I/O口。
- 可以通过软件将8个引脚上的任何一个设置为ADC模拟输入。

注：没有设置为ADC模拟输入的引脚可以作为普通I/O使用。

# STC单片机内ADC寄存器组
# --P1口模拟功能控制寄存器P1ASF

## P1口模拟功能控制寄存器P1ASF

- 该寄存器位于STC单片机特殊功能寄存器地址为0x9D的位置。
- 当复位后，该寄存器的值为“00000000”。

P1口模拟功能控制寄存器P1ASF各位的含义

| 比特 | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 |
|---|---|---|---|---|---|---|---|---|
| 名字 | P17ASF | P16ASF | P15ASF | P14ASF | P13ASF | P12ASF | P11ASF | P10ASF |

- P17ASF
  - 模拟输入通道7控制位。当该位为1时，P1.7引脚用于模拟信号输入；当该位为0时，P1.7引脚用作普通I/O。

# STC单片机内ADC寄存器组
## --P1口模拟功能控制寄存器P1ASF

- **P16ASF**
  - 模拟输入通道6控制位。当该位为1时，P1.6引脚用于模拟信号输入；当该位为0时，P1.6引脚用作普通I/O。
- **P15ASF**
  - 模拟输入通道5控制位。当该位为1时，P1.5引脚用于模拟信号输入；当该位为0时，P1.5引脚用作普通I/O。
- **P14ASF**
  - 模拟输入通道4控制位。当该位为1时，P1.4引脚用于模拟信号输入；当该位为0时，P1.4引脚用作普通I/O。
- **P13ASF**
  - 模拟输入通道3控制位。当该位为1时，P1.3引脚用于模拟信号输入；当该位为0时，P1.3引脚用作普通I/O。

# STC单片机内ADC寄存器组
## --P1口模拟功能控制寄存器P1ASF

- **P12ASF**
  - 模拟输入通道2控制位。当该位为1时，P1.2引脚用于模拟信号输入；当该位为0时，P1.2引脚用作普通I/O。
- **P11ASF**
  - 模拟输入通道1控制位。当该位为1时，P1.1引脚用于模拟信号输入；当该位为0时，P1.1引脚用作普通I/O。
- **P10ASF**
  - 模拟输入通道0控制位。当该位为1时，P1.0引脚用于模拟信号输入；当该位为0时，P1.0引脚用作普通I/O。

# STC单片机内ADC寄存器组
# --ADC控制寄存器

## ADC控制寄存器ADC_CONTR

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

ADC控制寄存器ADC_CONTR各位的含义

| 比特 | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 |
|---|---|---|---|---|---|---|---|---|
| 名字 | ADC_POWER | SPEED1 | SPEED0 | ADC_FLAG | ADC_START | CHS2 | CHS1 | CHS0 |

- ADC_POWER
  - ADC电源控制位。为0时，关闭ADC电源；为1时，打开ADC电源。

# STC单片机内ADC寄存器组
# --ADC控制寄存器

## ■ SPEED1和SPEED0

数模转换器速度控制位各位的含义

| SPEED1 | SPEED0 | ADC |
|---|---|---|
| 1 | 1 | 90个时钟周期转换一次。CPU工作频率为27MHz时，ADC的转换速度为300KHz。 |
| 1 | 0 | 180个时钟周期转换一次。CPU工作频率为27MHz时，ADC的转换速度为150KHz。 |
| 0 | 1 | 360个时钟周期转换一次。CPU工作频率为27MHz时，ADC的转换速度为75KHz。 |
| 0 | 0 | 540个时钟周期转换一次。CPU工作频率为27MHz时，ADC的转换速度为50KHz。 |

# STC单片机内ADC寄存器组
# --ADC控制寄存器

- **ADC_FLAG**
  - ADC转换结束标志位。当ADC转换结束时，由硬件置为1，该位需要由软件清零。

**注：不管是中断还是轮询该位，一定要用软件清零。**

- **ADC_START**
  - ADC转换启动控制位。当该位为1时，启动ADC转换；转换结束后，该位为0。

# STC单片机内ADC寄存器组
# --ADC控制寄存器

- **CHS2、CHS1和CHS0**
  - **模拟输入通道选择控制位。**

**CHS2、CHS1和CHS0各位含义**

| CHS2 | CHS1 | CHS0 | 功能 |
|---|---|---|---|
| 0 | 0 | 0 | 选择P1.0引脚作为内部ADC模块采样输入 |
| 0 | 0 | 1 | 选择P1.1引脚作为内部ADC模块采样输入 |
| 0 | 1 | 0 | 选择P1.2引脚作为内部ADC模块采样输入 |
| 0 | 1 | 1 | 选择P1.3引脚作为内部ADC模块采样输入 |
| 1 | 0 | 0 | 选择P1.4引脚作为内部ADC模块采样输入 |
| 1 | 0 | 1 | 选择P1.5引脚作为内部ADC模块采样输入 |
| 1 | 1 | 0 | 选择P1.6引脚作为内部ADC模块采样输入 |
| 1 | 1 | 1 | 选择P1.7引脚作为内部ADC模块采样输入 |

# STC单片机内ADC寄存器组
# --时钟分频寄存器

## 时钟分频寄存器CLK_DIV

- 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“0000x000”。

**注：该寄存器中的ADRJ位用于控制ADC转换结果存放的位置。**

时钟分频寄存器CLK_DIV各位的含义

| 比特 | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 |
|---|---|---|---|---|---|---|---|---|
| 名字 | MCLKO_S1 | MCLKO_S0 | ADRJ | Tx_Rx | -- | CLKS2 | CLKS1 | CLKS0 |

- ADRJ为0时，ADC_RES[7:0]存放高8位结果，ADC_RESL[1:0]存放低2位结果。
- ADRJ为1时，ADC_RES[1:0]存放高2位结果，ADC_RESL[7:0]存放低8位结果。

# STC单片机内ADC寄存器组
# --ADC结果高位寄存器

## ADC结果高位寄存器ADC_RES

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

ADC结果高位寄存器ADC_RES各位的含义

| 比特 | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 |
|---|---|---|---|---|---|---|---|---|
| 名字 | 内容由ADRJ控制 | | | | | | | |

# STC单片机内ADC寄存器组
# --ADC结果低位寄存器

## ADC结果低位寄存器ADC_RESL。

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

ADC结果低位寄存器ADC_RESL各位的含义

| 比特 | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 |
|---|---|---|---|---|---|---|---|---|
| 名字 | 内容由ADRJ控制 | | | | | | | |

# STC单片机内ADC寄存器组
# --中断使能寄存器

## 中断使能寄存器

- 该寄存器位于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”。

中断控制寄存器IE各位含义

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

- EADC
  - 为ADC转换中断允许位。当该位为1时，允许ADC转换中断；当该位位0时，禁止ADC转换中断。

# STC单片机内ADC寄存器组
# --中断优先级寄存器

## 中断优先级寄存器

- 该寄存器位于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各位含义

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

- PADC为ADC转换优先级控制位。
  - 当该位为0时，ADC转换中断为最低优先级中断（优先级0）；当该位为1时，ADC转换中断为最高优先级中断（优先级1）。

# ADC应用实现1

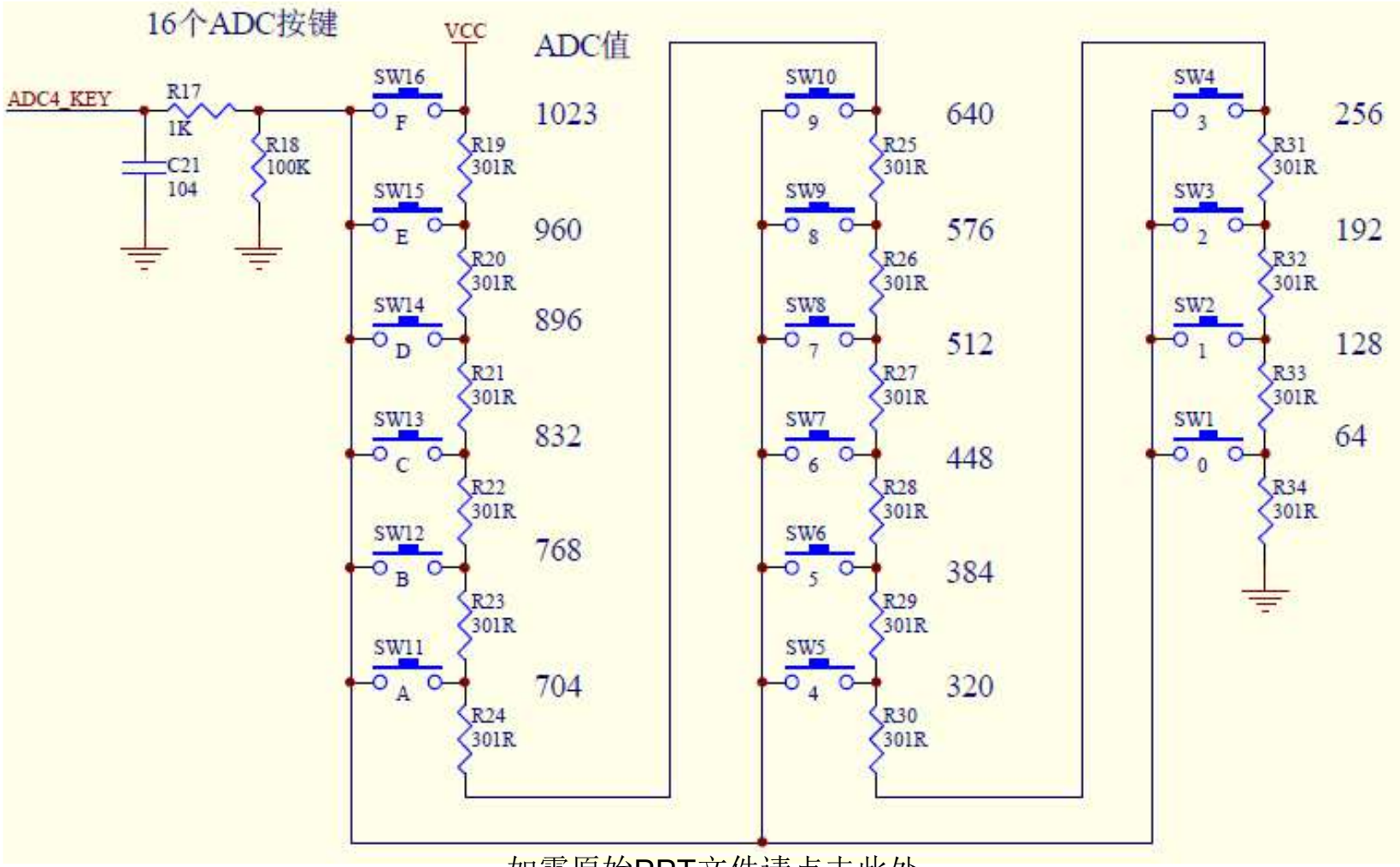
该设计将读取STC学习板上按下不同按键所得到的直流电压值，经过ADC转换器转换后，得到数字量的值，经过计算后，通过串口1发送到主机串口界面显示得到的直流电压值。

# ADC应用实现1
# --直流分压电路原理

在电源VCC和GND之间连接着由16个电阻构成的电阻梯度网络。存在下面的条件：

- 从STC15系列单片机ADC结构图可以知道，ADC输入连接着模拟比较器。因此，ADC的输入阻抗为无穷大，即：在电阻R17上没有电流。因此，对电阻梯度网络没有影响。
- 由于R18电阻为100K，远大于电阻梯度网络的总电阻300×16=4.8K。所以，流经R18的电流很小，可以忽略不计。
- 在该设计中使用10位ADC，即$2^{10}=1024$。

# ADC应用实现1
# --直流分压电路原理



# ADC应用实现1
# --直流分压电路原理

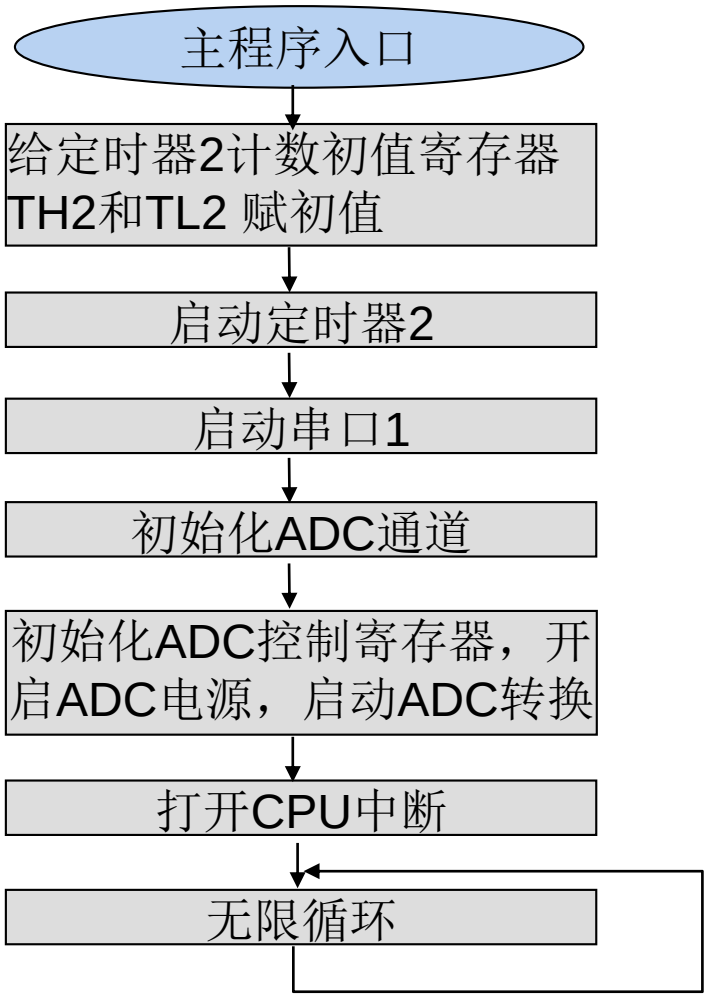
**在该结构中，存在SW1~SW16个按键，当：**

- **按下按键SW16时，将VCC的电压送到单片机P1.4引脚上**
- **按下按键SW15时，将VCC电压的15/16送到单片机P1.4引脚上**
- **按下按键SW14时，将VCC电压的14/16送到单片机P1.4引脚上**
- **按下按键SW13时，将VCC电压的13/16送到单片机P1.4引脚上**
- **按下按键SW12时，将VCC电压的12/16送到单片机P1.4引脚上**
- **按下按键SW11时，将VCC电压的11/16送到单片机P1.4引脚上**
- **按下按键SW10时，将VCC电压的10/16送到单片机P1.4引脚上**

# ADC应用实现1
# --直流分压电路原理

- 按下按键SW9时，将VCC电压的9/16送到单片机P1.4引脚上
- 按下按键SW8时，将VCC电压的8/16送到单片机P1.4引脚上
- 按下按键SW7时，将VCC电压的7/16送到单片机P1.4引脚上
- 按下按键SW6时，将VCC电压的6/16送到单片机P1.4引脚上
- 按下按键SW5时，将VCC电压的5/16送到单片机P1.4引脚上
- 按下按键SW4时，将VCC电压的4/16送到单片机P1.4引脚上
- 按下按键SW3时，将VCC电压的3/16送到单片机P1.4引脚上
- 按下按键SW2时，将VCC电压的2/16送到单片机P1.4引脚上
- 按下按键SW1时，将VCC电压的1/16送到单片机P1.4引脚上

# ADC应用实现1
# -- 软件设计流程

# ADC应用实现1
# --具体实现过程

【例】采集分压网络的电压值在串口上显示的C语言描述的例子。

```
#include "reg51.h"
#include "stdio.h"
#define OSC       18432000L        //定义OSC振荡器频率18432000Hz
#define BAUD    9600               //定义BAUD波特率9600
#define URMD   0                   //定义URMD的值
#define ADC_POWER    0x80          //定义ADC_POWER的值0x80
#define ADC_FLAG       0x10        //定义ADC_FLAG的值0x10
#define ADC_START      0x08        //定义ADC_START的值0x08
#define ADC_SPEEDLL   0x00         //定义ADC_SPEEDLL的值0x00
#define ADC_SPEEDL     0x20        //定义ADC_SPEEDL的值0x20
#define ADC_SPEEDH    0x40         //定义ADC_SPEEDH的值0x40
```

# ADC应用实现1
# --具体实现过程

```
#define ADC_SPEEDHH0x60      //定义ADC_SPEEDHH的值0x60
sfr T2H=0xD6;                //定义T2H寄存器的地址0xD6
sfr T2L =0xD7;               //定义T2L寄存器的地址0xD7
sfr AUXR=0x8E;               //定义AUXR寄存器的地址0x8E
sfr ADC_CONTR=0xBC;          //定义ADC_CONTR寄存器的地址0xBC
sfr ADC_RES=0xBD;            //定义ADC_RES寄存器的地址0xBD
sfr ADC_RESL=0xBE;           //定义ADC_RESL寄存器的地址0xBE
sfr P1ASF=0x9D;              //定义P1ASF寄存器的地址0x9D
unsigned char ch=4;          //定义无符号变量ch，指向P1.4端口
float voltage=0;             //定义浮点变量voltage
unsigned char tstr[5];       //定义无符号字符数组tstr
unsigned int tmp=0;          //定义无符号整型变量tmp
float old_voltage=0;         //定义浮点变量old_voltage
```

# ADC应用实现1
# --具体实现过程

```
void SendData(unsigned char dat)          //定义函数SendData
{
        while(!TI);                       //判断发送是否结束，没有则等待
        TI=0;                             //清除发送标志TI
        SBUF=dat;                         //将数据dat写到寄存器SBUF
}

void adc_int() interrupt 5                  //声明adc中断服务程序
{       unsigned char i=0;                  //定义无符号字符变量i
        ADC_CONTR &=!ADC_FLAG;              //清除ADC中断标志位
```

# ADC应用实现1
# --具体实现过程

```
tmp=(ADC_RES*4+ADC_RESL); //得到ADC转换的数字量
voltage=(tmp*5.0)/1024;                 //计算得到对应的浮点模拟电压值
sprintf(tstr,“%1.4f”,voltage);          //将浮点数转换成对应的字符
if(voltage!=old_voltage)                //如果新的转换值不等于旧的转换值
{       old_voltage=voltage;            //将新的转换值赋值给旧的转换值
        SendData(‘\r’);                 //发送回车和换行符
        SendData('\n');
        for(i=0;i<5;i++)                //发送五个对应的浮点数的字符，一个
        SendData(tstr[i]);              //整数，4个小数
}                                       //重新启动ADC转换
ADC_CONTR=ADC_POWER |ADC_SPEEDLL | ADC_START | ch;
}
```

# ADC应用实现1
# --具体实现过程

```
void main()                                    //主程序
{       unsigned int i;
        SCON=0x5A;                             //串口1为8位可变波特率模式
        T2L=65536-OSC/4/BAUD;                  //写定时器2低8位寄存器T2L
        T2H=(65536-OSC/4/BAUD)>>8;//写定时器2高8位寄存器T2H
        AUXR=0x14;                             //定时器2不分频，启动定时器2
        AUXR|=0x01;                            //选择定时器2为串口1的波特率发生器
        P1ASF=0xFF;                            //P1端口作为模拟输入
        ADC_RES=0;                             //清ADC_RES寄存器
        ADC_CONTR=ADC_POWER|ADC_SPEEDLL | ADC_START | ch;
                                               //启动ADC
        for(i=0;i<10000;i++);                  //延迟
        IE=0xA0;                              //CPU允许响应中断请求，允许ADC中断
        while(1);                              //无限循环
}
```

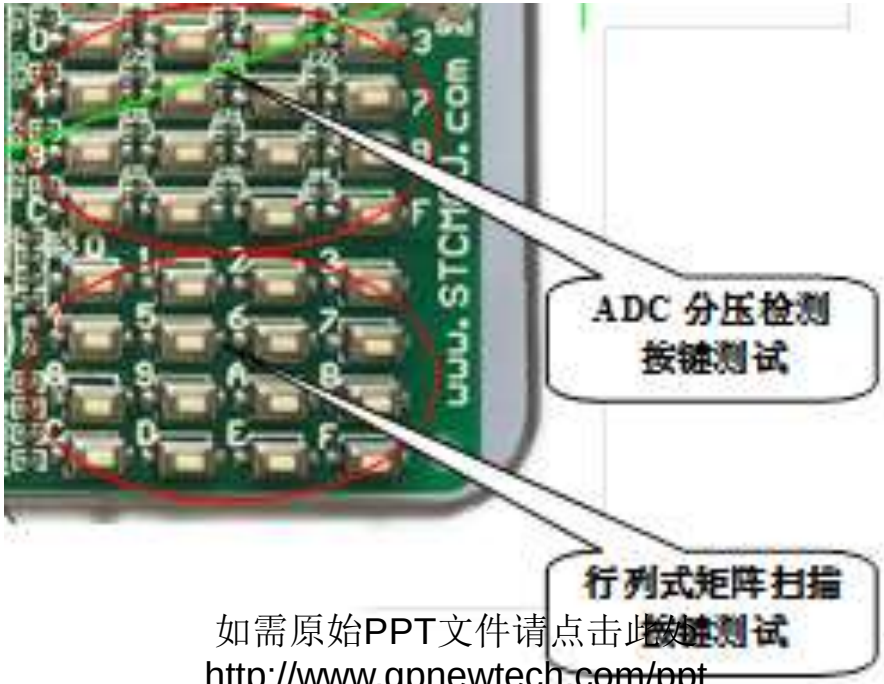
# ADC应用实现1
# --具体实现过程

下载和分析设计的步骤主要包括：

- 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。
- 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：9600。
  - 校验位：无校验。
  - 停止位：1位。

# ADC应用实现1
# --具体实现过程

- 单击打开串口按钮。
- 在STC学习板上右下方，找到并按一下ADC分压检测按键。可以看到在上面的接收窗口中，显示出按键所对应的模拟电压的值。



# ADC应用实现2

本设计将读取STC学习板上按下不同按键所得到的直流电压值，经过ADC转换器转换后，得到数字量的值，经过计算后，通过1602字符LCD屏显示得到的直流电压值。

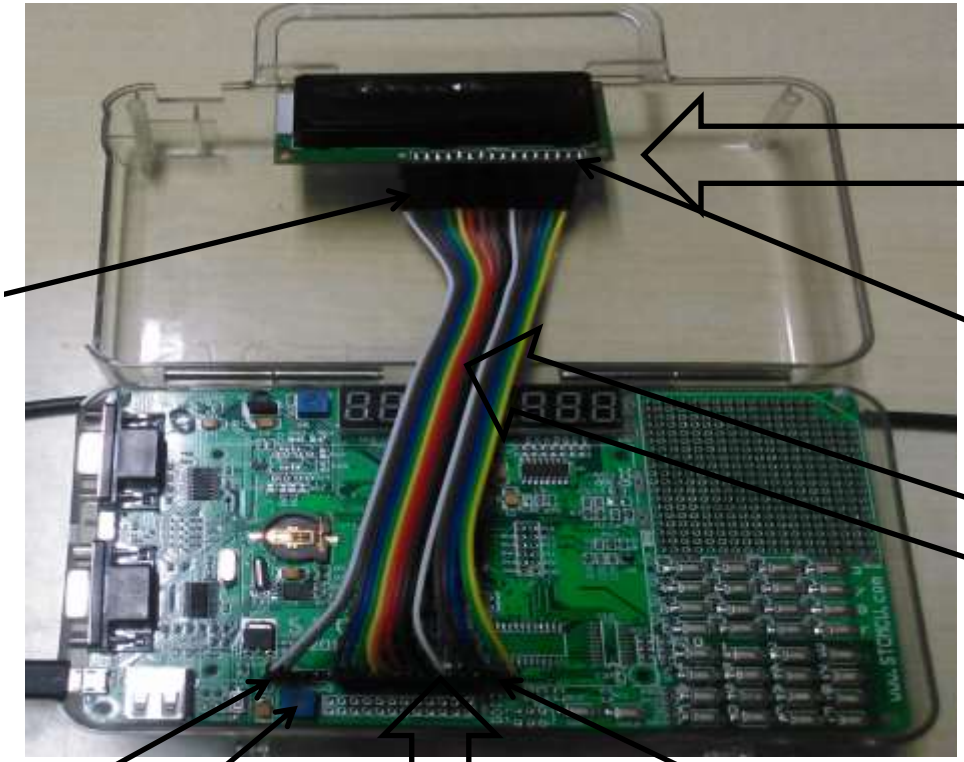
# ADC应用实现2
# --硬件电路设计

在该设计中，1602字符屏通过排线电缆与STC学习板上的标记为J12的单排插座连接。

- 在图中标出了STC学习板上插针引脚1的位置和1602字符屏引脚1的位置。
- STC学习板上J12提供20个插针，可以直接与12864图形/字符LCD进行连接，对于1602字符屏来说，不能直接进行连接。

注：它们的信号引脚定义如后表所示。

# ADC应用实现2
# --硬件电路设计



# ADC应用实现2
# --硬件电路设计

| STC学习板 J12插座引脚号 | 信号名字 | 与单片机引脚连接关系 | 1602LCD引脚号 | 信号名字 | 功能 |
|---|---|---|---|---|---|
| 1 | GND | 地 | 1 | VSS | 地 |
| 2 | VCC | +5V电源 | 2 | VCC | +5V电源 |
| 3 | V0 | -- | 3 | V0 | LCD驱动电压输入 |
| 4 | RS | P2.5 | 4 | RS | 寄存器选择。RS=1，数据；RS=0，指令 |
| 5 | R/W | P2.6 | 5 | R/W | 读写信号。R/W=1，读操作；R/W=0，写操作 |
| 6 | E | P2.7 | 6 | E | 芯片使能信号 |
| 7 | DB0 | P0.0 | 7 | DB0 | 8位数据总线信号 |
| 8 | DB1 | P0.1 | 8 | DB1 | |
| 9 | DB2 | P0.2 | 9 | DB2 | |
| 10 | DB3 | P0.3 | 10 | DB3 | |
| 11 | DB4 | P0.4 | 11 | DB4 | |
| 12 | DB5 | P0.5 | 12 | DB5 | |
| 13 | DB6 | P0.6 | 13 | DB6 | |
| 14 | DB7 | P0.7 | 14 | DB7 | |
| 15 | PSB | P2.4 | 15 | LEDA | 背光源正极，接+5.0V |
| 16 | N.C | P2.2 | 16 | LEDK | 背光源负极，接地 |
| 17 | /RST | P2.3 | | | |
| 18 | VOUT | -- | | | |
| 19 | A | 背光源正极，接+5.0V | | | |

# 硬件电路设计
# --1602字符LCD原理

## 1602字符LCD指标

1602字符LCD主要技术参数

| | |
|---|---|
| 显示容量 | 16×2个字符，即：可以显示2行字符，每行可以显示16个字符 |
| 工作电压范围 | 4.5V~5.5V。推荐5.0V |
| 工作电流 | 2.0mA@5V |
| 屏幕尺寸 | 2.95×4.35mm（宽×高） |

# 硬件电路设计
## --1602字符LCD原理

### 1602字符LCD内部显存

- 1602液晶内部包含80个字节的显示RAM，存储需要发送的数据
- 第一行存储器地址范围0x00~0x27；第二行存储器地址范围为0x40~0x67。
  - 第一行存储器地址范围0x00~0x0F与1602字符LCD第一行位置对应。
  - 第二行存储器地址范围0x40~0x4F与1602字符LCD第二行位置对应。

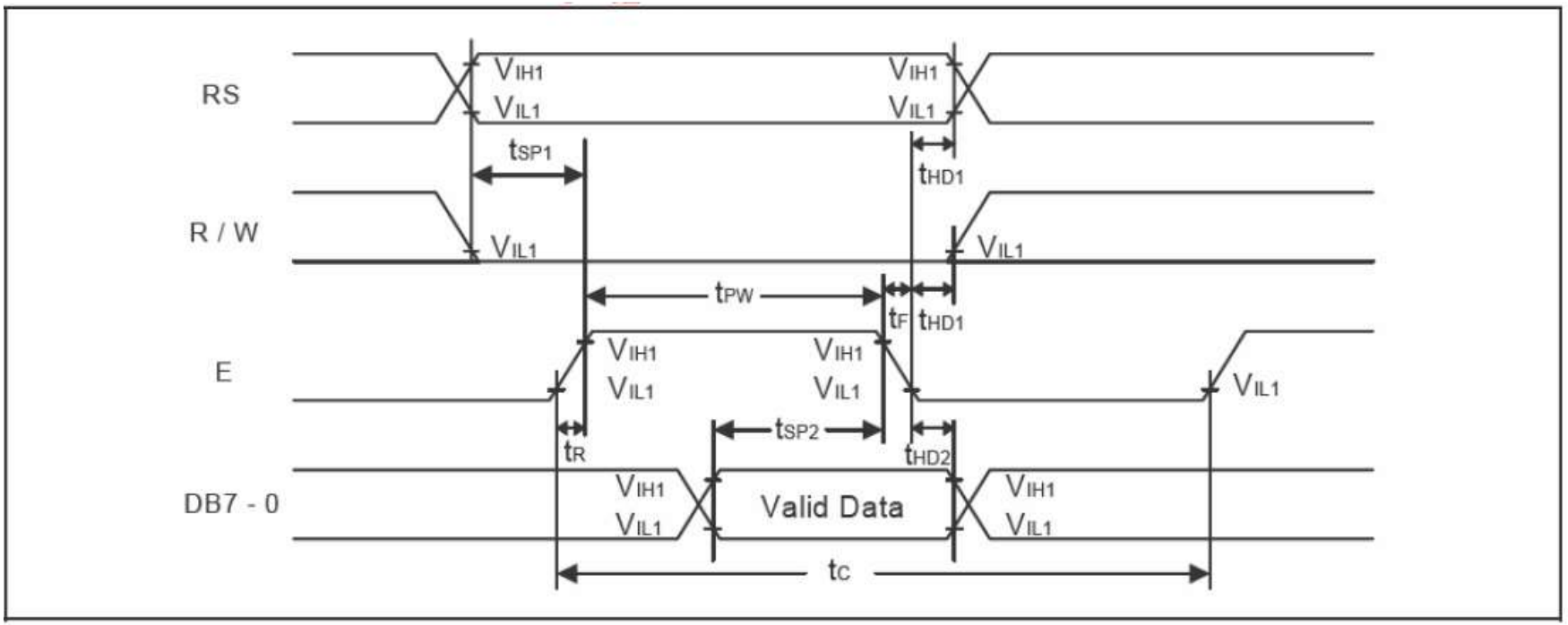
**注：每行多出来的部分是为了显示移动字幕设置。**

16×2字符LCD

| | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 0E | 0F | 10 | 11 | ... | 27 |
| 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 4A | 4B | 4C | 4D | 4E | 4F | 50 | 51 | ... | 67 |

# 硬件电路设计
## --1602字符LCD读写时序

## 写操作时序

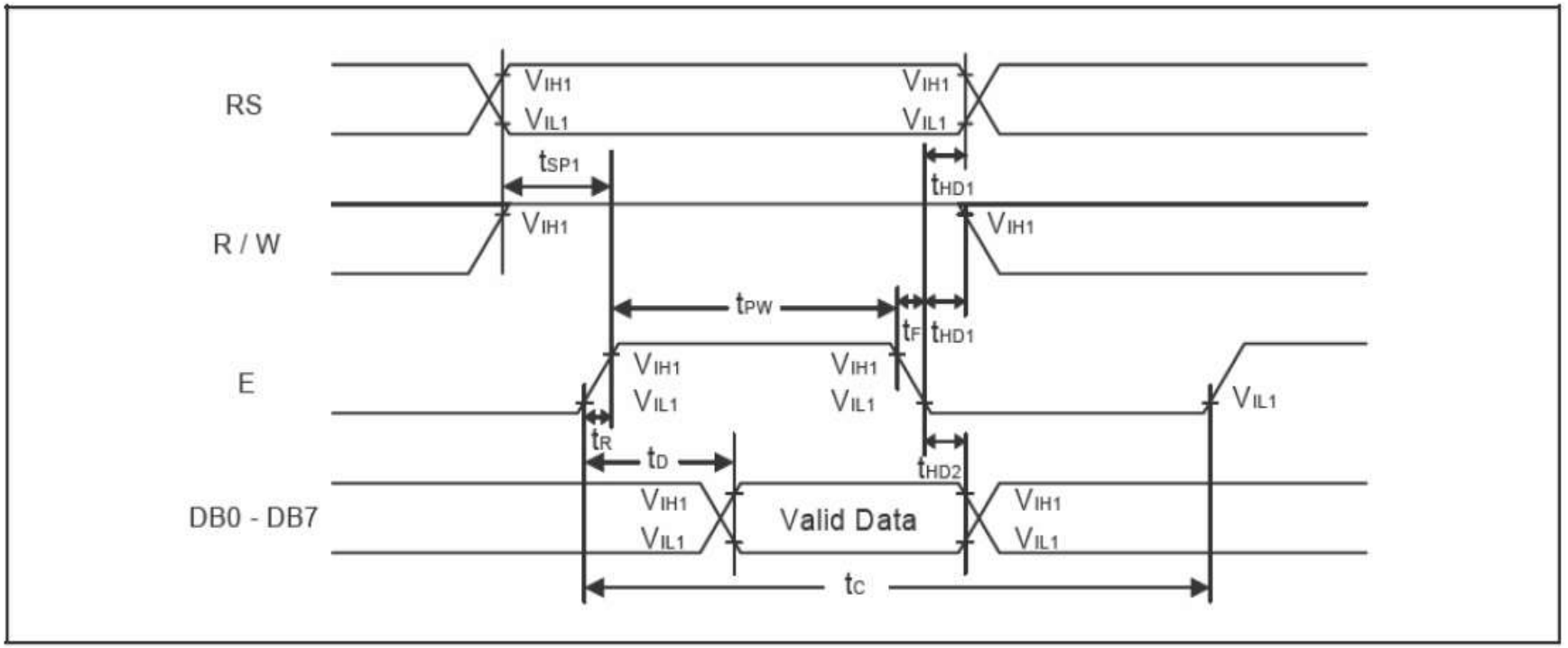


# 硬件电路设计
## --1602字符LCD读写时序

- 首先，将R/W信号拉低。同时，给出RS信号，该信号为1或者0，用于区分数据和命令。
- 然后，将E信号拉高。当E信号拉高后，STC单片机将写入1602字符LCD的数据放在DB7~DB0数据线上。当数据有效一段时间后，首先将E信号拉低。然后，数据继续维持一段时间$T_{HD2}$。这样，数据就写到1602字符LCD中。
- 最后，撤除/保持R/W信号。

# 硬件电路设计
## --1602字符LCD读写时序

## 读操作时序



# 硬件电路设计
## --1602字符LCD读写时序

- 首先，将R/W信号拉高。同时，给出RS信号，该信号为1或者0，用于区分数据和状态。
- 然后，将E信号拉高。当E信号拉高，并且延迟一段时间$t_o$后，1602字符LCD将数据放在DB7~DB0数据线上。当维持一段时间$t_{pw}$后，将E信号拉低。
- 最后，撤除/保持R/W信号。

# 硬件电路设计
# --1602字符LCD读写时序

## 将上面的读和写操作总结

读和写操作总结

| RS | R/W | 操作说明 |
|---|---|---|
| 0 | 0 | 写入指令寄存器（清屏） |
| 0 | 1 | 读BF（忙）标志，以及读取地址计数器的内容 |
| 1 | 0 | 写入数据寄存器（显示各字型等） |
| 1 | 1 | 从数据寄存器读取数据 |

# 硬件电路设计
## --1602字符LCD命令和数据

**在STC单片机对1602字符LCD操作的过程中，会用到下面的命令**

**1602字符LCD命令和数据**

| 指令 | 指令操作码 | | | | | | | | | | 功能 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | RS | RW | DB7 | DB6 | DB5 | DB4 | DB3 | DB2 | DB1 | DB0 | |
| 清屏 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 将“20H”写到DDRAM，将DDRAM地址从AC（地址计数器）设置到“00” |
| 光标归位 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | - | 将DDRAM的地址设置为“00”，光标如果移动，则将光标返回到初始的位置。DDRRAM的内容保持不变 |
| 输入模式设置 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | I/D | S | 分配光标移动的方向，使能整个显示的移动。<br>I=0，递减模式。I=1，递增模；S=0，关闭整个移动。S=1，打开整个移动； |

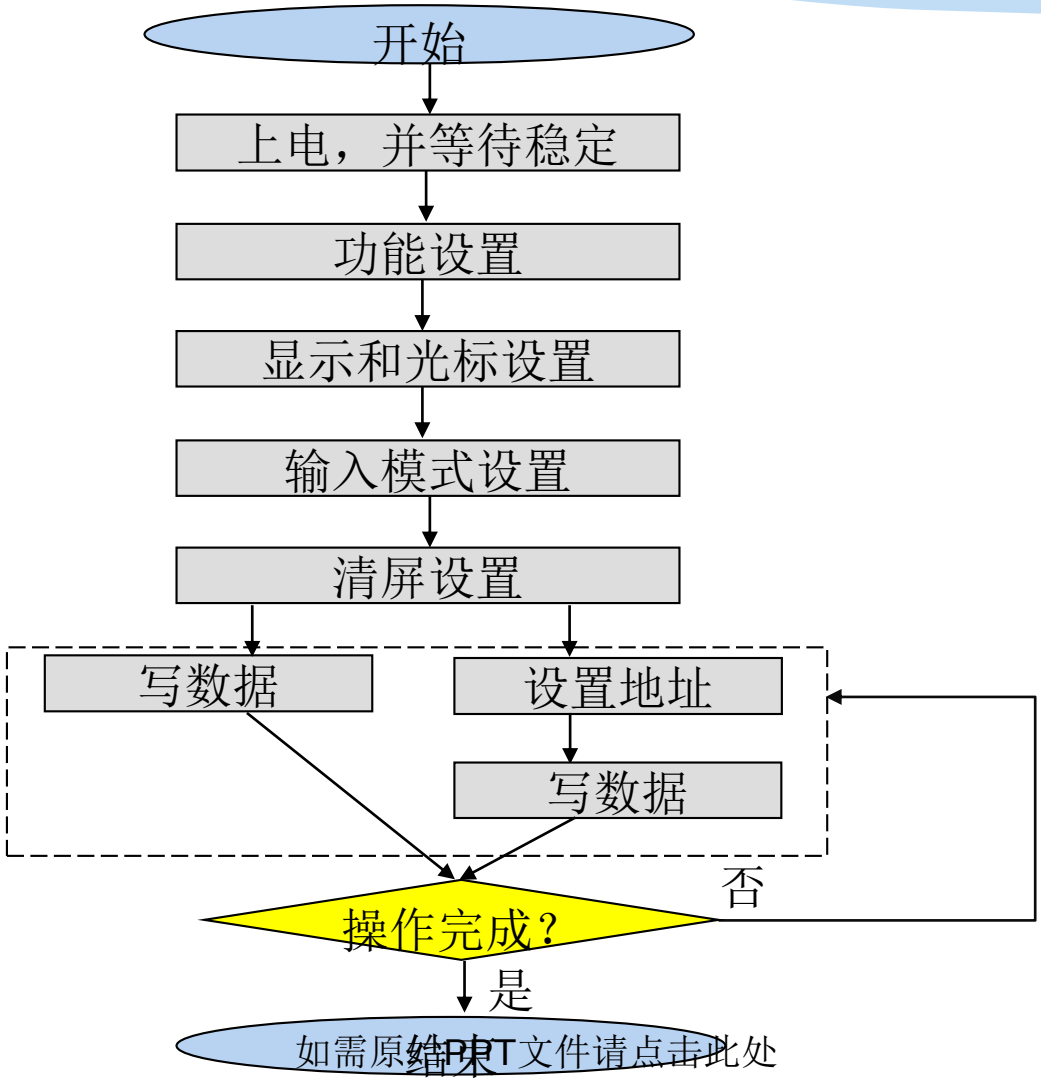
# 硬件电路设计
## --1602字符LCD命令和数据

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 显示打开 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | D | C | B | 设置显示（D），光标（C）和光标闪烁（B）打开/关闭控制。<br>D=0，显示关闭；<br>D=1，打开显示<br>C=0，关闭光标；<br>C=1，打开光标<br>B=0，关闭闪烁；<br>B=1，打开闪烁 |
| 光标或者显示移动 | 0 | 0 | 0 | 0 | 0 | 1 | S/C | R/L | - | - | 设置光标移动和显示移动的控制位，以及方向，不改变DDRAM数据<br>S/C=0，R/L=0，光标左移；<br>S/C=0，R/L=1，光标右移；<br>S/C=1，R/L=0，显示左移，光标跟随显示移动<br>S/C=1，R/L=1，显示右移，光标跟随显示移动 |

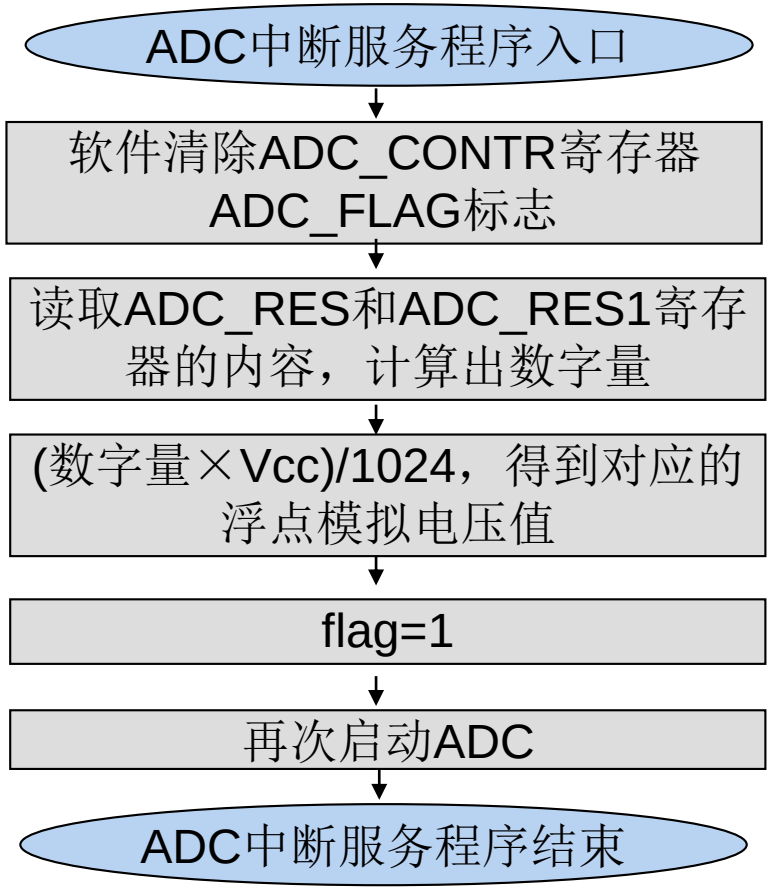
# 硬件电路设计
## --1602字符LCD命令和数据

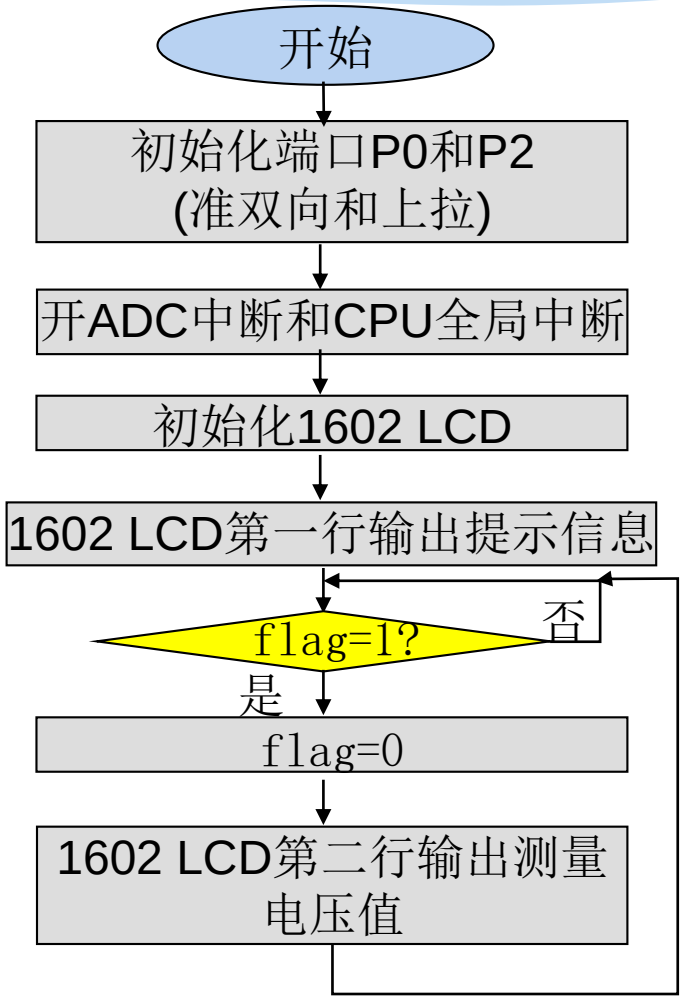
| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 功能设置 | 0 | 0 | 0 | 0 | 1 | DL | N | F | - | - | 设置接口数据宽度，以及显示行的个数 。<br>DL=1，8位宽度；DL=0，4位宽度<br>N=0，1行模式；N=1，2行模式<br>F=0，5×8字符字体；F=1，5×10字符字体 |
| 设置CGRAM地址 | 0 | 0 | 0 | 1 | AC5 | AC4 | AC3 | AC2 | AC1 | AC0 | 在地址计数器中，设置CGRAM地址 |
| 设置DDRAM地址 | 0 | 0 | 1 | AC6 | AC5 | AC4 | AC3 | AC2 | AC1 | AC0 | 在计数器中，设置DDRAM地址 |
| 读忙标志和地址计数器 | 0 | 1 | BF | AC6 | AC5 | AC4 | AC3 | AC2 | AC1 | AC0 | 读BF标志，知道LCD屏内部是否正在操作。也可以读取地址计数器的内容 |
| 将数据写到RAM | 1 | 0 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 | 写数据到内部RAM（DDRAM/ CGRAM） |
| 从RAM读数据 | 1 | 1 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 | 从内部RAM（DDRAM/ CGRAM）读取数据 |

# ADC应用实现2
# --1602字符LCD初始化和操作流程



# ADC应用实现2
# --系统软件处理流程



# ADC应用实现2
# --具体实现过程

【例】采集分压网络的电压值在1602字符LCD上显示的C语言描述的例子。

led1602.h文件

```
#ifndef _1602_                  //条件编译命令，如果没有定义_1602_
#define _1602_                  //定义_1602_
#include "reg51.h"              //包含reg51.h头文件
#include "intrins.h"            //包含intrins.h头文件
sbit LCD1602_RS=P2^5;           //定义LCD1602_RS为P2.5引脚
sbit LCD1602_RW=P2^6;           //定义LCD1602_RW为P2.6引脚
sbit LCD1602_E =P2^7;           //定义LCD1602_E为P2.7引脚
sfr  LCD1602_DB=0x80;           //定义LCD1602_DB为P0端口
sfr  P0M1=0x93;                 //定义P0端口P0M1寄存器地址0x93
```

# ADC应用实现2
# --具体实现过程

```
sfr  P0M0=0x94;                          //定义P0端口P0M0寄存器地址0x94
sfr  P2M1=0x95;                          //定义P2端口P2M1寄存器地址0x95
sfr  P2M0=0x96;                          //定义P2端口P2M0寄存器地址0x96
void lcdwait();                          //定义子函数lcdwait类型
void lcdwritecmd(unsigned char cmd);     //定义子函数lcdwritecmd类型
void lcdwritedata(unsigned char dat);    //定义子函数lcdwritedata类型
void lcdinit();                          //定义子函数lcdinit类型
void lcdsetcursor(unsigned char x, unsigned char y);
                                         //定义子函数lcdsetcursor类型
void lcdshowstr(unsigned char x, unsigned char y,unsigned char *str);
                                         //定义子函数lcdshowstr类型
#endif                                   //条件预编译命令结束
```

# ADC应用实现2
# --具体实现过程

led1602.c文件

```
#include “led1602.h”                    //包含led1602.h头文件
void lcdwait()                          //声明lcdwait函数，用于读取BF标志
{       LCD1602_DB=0xFF;                //读取前，先将P0端口设置为“1”
        _nop_();                        //空操作延迟
        _nop_();                        //空操作延迟
        _nop_();                        //空操作延迟
        _nop_();                        //空操作延迟
        LCD1602_RS=0;                   //将LCD1602的RS信号拉低
        LCD1602_RW=1;                   //将LCD1602的RW信号拉高
        LCD1602_E=1;                    //将LCD1602的E信号拉高
        while(LCD1602_DB & 0x80);       //等待标志BF为低表示LCD1602空闲
        LCD1602_E=0;                    //将LCD1602的E信号拉低
}
```

# ADC应用实现2
# --具体实现过程

```
void lcdwritecmd(unsigned char cmd)
                                        //声明lcdwritecmd函数，写命令到1602
{
        lcdwait();                      //调用lcdwait函数
        _nop_();                        //空操作延迟
        _nop_();                        //空操作延迟
        _nop_();                        //空操作延迟
        _nop_();                        //空操作延迟
        LCD1602_RS=0;                   //将LCD1602的RS信号拉低
        LCD1602_RW=0;                   //将LCD1602的RW信号拉低
        LCD1602_DB=cmd;                 //将命令控制码cmd放到P0端口
        LCD1602_E=1;                    //将LCD1602的E信号拉高
```

# ADC应用实现2
# --具体实现过程

```
    _nop_();                    //空操作延迟
    _nop_();                    //空操作延迟
    _nop_();                    //空操作延迟
    _nop_();                    //空操作延迟
    LCD1602_E=0;                //将LCD1602的E信号拉低
}
```

# ADC应用实现2
# --具体实现过程

```
void lcdwritedata(unsigned char dat) //声明lcdwritedata函数，写数据到1602
{
        lcdwait();                  //调用lcdwait函数
        _nop_();                    //空操作延迟
        _nop_();                    //空操作延迟
        _nop_();                    //空操作延迟
        _nop_();                    //空操作延迟
        LCD1602_RS=1;               //将LCD1602的RS信号拉高
        LCD1602_RW=0;               //将LCD1602的RW信号拉低
        LCD1602_DB=dat;             //将数据码cmd放到P0端口
        LCD1602_E=1;                //将LCD1602的E信号拉高
        _nop_();                    //空操作延迟
        _nop_();                    //空操作延迟
        _nop_();                    //空操作延迟
        _nop_();                    //空操作延迟
        LCD1602_E=0;                //将LCD1602的E信号拉低
}
```

# ADC应用实现2
# --具体实现过程

```
void lcdinit()                  //声明lcdinit子函数，用来初始化1602
{       lcdwritecmd(0x38);      //发命令0x38，2行模式，5*8点阵，8位宽度
        lcdwritecmd(0x0c);      //发命令0x0C，打开显示，关闭光标
        lcdwritecmd(0x06);      //发命令0x06，文字不移动，地址自动加1
        lcdwritecmd(0x01);      //发命令0x01，清屏
}
```

# ADC应用实现2
# --具体实现过程

```
//声明lcdsetcursor函数，设置显示RAM的地址，x和y表示在1602的列和行参数
void lcdsetcursor(unsigned char x, unsigned char y)
{
    unsigned char address;          //声明无符号char类型变量address
    if(y==0)                        //如果第一行
        address=0x00+x;             //存储器地址以0x00开始
    else                            //如果是第二行
        address=0x40+x;             //存储器地址以0x40开始
    lcdwritecmd(address|0x80);      //写存储器地址命令
}
```

# ADC应用实现2
# --具体实现过程

```
void lcdshowstr(unsigned char x, unsigned char y,unsigned char *str)
 //在液晶指定的x和y位置，显示字符
 {
     lcdsetcursor(x,y);              //设置显示RAM的地址
     while((*str)!= ‘\0’ )           //如果不是字符串的结尾，则继续
      {
         lcdwritedata(*str);        //发写数据命令，在LCD上显示数据
          str++;                    //指针加1，指向下一个地址
       }
}
```

# ADC应用实现2
# --具体实现过程

main.c文件

```
#include "reg51.h"
#include "stdio.h"
#include "led1602.h"
#define ADC_POWER    0x80            //定义ADC_POWER的值0x80
#define ADC_FLAG     0x10            //定义ADC_FLGA的值0x10
#define ADC_START    0x08            //定义ADC_START的值0x08
#define ADC_SPEEDLL  0x00            //定义ADC_SPEEDLL的值0x00
#define ADC_SPEEDL   0x20            //定义ADC_SPEEDL的值0x20
#define ADC_SPEEDH   0x40            //定义ADC_SPEEDH的值0x40
#define ADC_SPEEDHH  0x60            //定义ADC_SPEEDHH的值0x60
```

# ADC应用实现2
# --具体实现过程

```
sfr AUXR =0x8E;                 //声明AUXR寄存器的地址0x8E
sfr ADC_CONTR =0xBC;            //声明ADC_CONTR寄存器的地址0xBC
sfr ADC_RES =0xBD;              //声明ADC_RES寄存器的地址0xBD
sfr ADC_RESL =0xBE;             //声明ADC_RESL寄存器的地址0xBE
sfr P1ASF =0x9D;                //声明P1ASF寄存器的地址0x9D
unsigned char ch=4;             //声明char类型变量ch
bit flag=1;                     //声明bit类型变量flag
float voltage=0;                //声明float类型变量voltage
unsigned char tstr[5];          //声明char类型数组tstr
unsigned int tmp=0;             //声明int类型变量tmp
```

# ADC应用实现2
# --具体实现过程

```
void adc_int() interrupt 5                          //声明adc中断服务程序
{
        unsigned char i=0;                          //声明char类型变量i
        ADC_CONTR &=!ADC_FLAG; //将ADC_FLAG标志清零
        tmp=(ADC_RES*4+ADC_RESL);          //读取模拟信号对应的数字量
        voltage=(tmp*5.0)/1024;                     //将数字量转换成模拟电压值
        sprintf(tstr,“%1.4f”,voltage);          //将浮点数，转换成对应的电压值
        flag=1;                                     //将flag置1
        ADC_CONTR=ADC_POWER |ADC_SPEEDLL | ADC_START | ch;
                                                    //启动ADC
}
```

# ADC应用实现2
# --具体实现过程

```
void main()
{       unsigned int i;              //声明int型变量i
        P0M0=0;                      //通过P0M0和P0M1寄存器，将P0口
        P0M1=0;                      //定义为准双向，弱上拉
        P2M0=0;                      //通过P2M0和P2M1寄存器，将P2口
        P2M1=0;                      //定义为准双向，弱上拉
        P1ASF=0xFF;                  //将P1端口用于ADC输入
        ADC_RES=0;                   //将ADC_RES寄存器清零
                                     //配置ADC_CONTR寄存器
        ADC_CONTR=ADC_POWER|ADC_SPEEDLL | ADC_START | ch;
        for(i=0;i<10000;i++);        //延迟一段时间
        IE=0xA0;                     //CPU允许相应中断请求，允许ADC中断
        lcdwait();                   //等待1602字符LCD稳定
```

# ADC应用实现2
# --具体实现过程

```
lcdinit();                          //初始化1602字符LCD
lcdshowstr(0,0,"Measured Voltage is");
                                    //在1602第一行开始打印信息
lcdshowstr(6,1,“V”);                //在1602第二行第6列打印字符 ‘v’
while(1)                            //无限循环
        {
          if(flag==1)               //判断flag标志是否为1，
           {
             flag=0;                //将flag置0
             lcdshowstr(0,1,tstr);  //在第二行，打印电压对应的字符
           }
        }
}
```

# ADC应用实现2
# --具体实现过程

**下载和分析设计的步骤主要包括：**

- **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。**
- **单击下载/编程按钮，按前面的方法下载设计到STC单片机。**
- **观察1602字符屏上的输出结果。**

# ADC应用实现3

本设计将从外部输入信号源，经过ADC转换器转换后，得到数字量的值，经过计算后，通过12864图形点阵LCD屏，一方面以字符显示得到的交流信号的最大值MAX、最小值MIN、峰峰值PTP（Peak to Peak）；另一方面以图形的方式显示采集交流信号的波形。

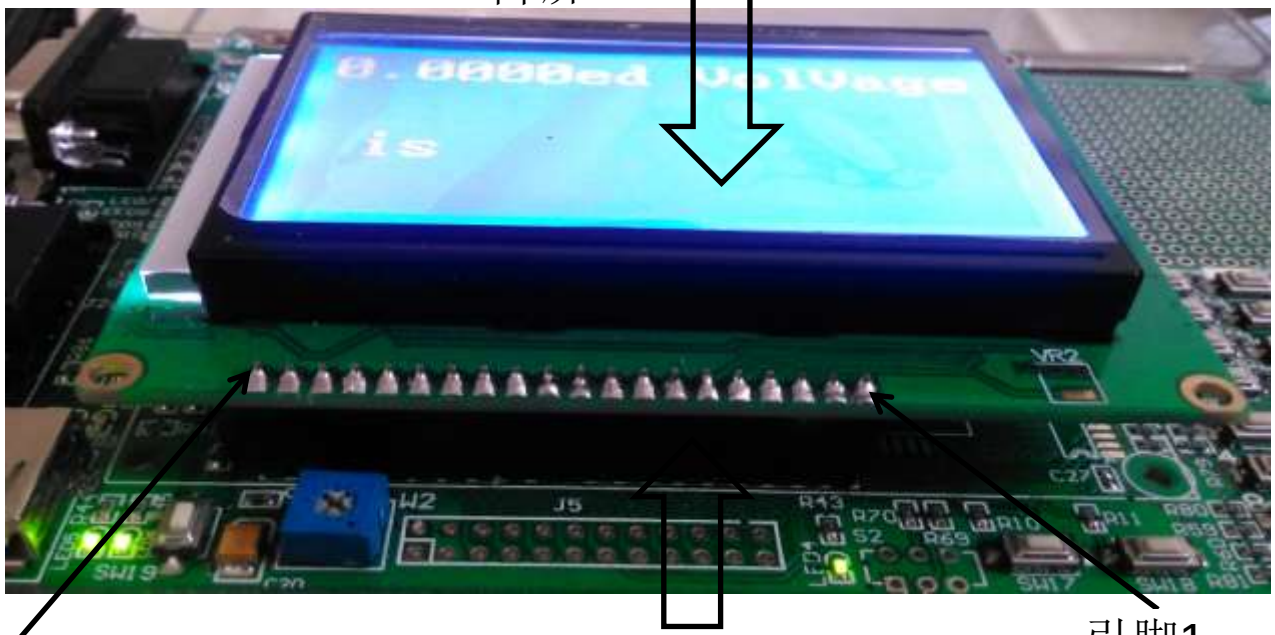
# ADC应用实现3
# --硬件电路设计

在该设计中，12864图形/字符点阵屏通过自己焊接的插座与STC学习板上的标记为J12的单排插座连接。

- 在图中标出了STC学习板上插针引脚1的位置和12864图形/字符点阵屏引脚1的位置。

# ADC应用实现3
# --硬件电路设计

12864图形/字符LCD点阵屏




引脚20

STC学习板上提供的标记为J12的单排插针和12864的单排插座插在一起

引脚1

# ADC应用实现3
# --硬件电路设计

**STC学习板上J12提供20个插针，可以直接与12864图形/字符LCD进行连接。**

# ADC应用实现3
# --硬件电路设计

| STC学习板 J12插座引脚号 | 信号名字 | 与单片机引脚的连接关系 | 12864 图 形 点 阵 LCD引脚号 | 信号名字 | 功能 |
|---|---|---|---|---|---|
| 1 | GND | 地 | 1 | VSS | 地 |
| 2 | VCC | +5V电源 | 2 | VCC | +5V电源 |
| 3 | V0 | -- | 3 | V0 | LCD驱动电压输入 |
| 4 | RS | P2.5 | 4 | RS | 寄存器选择。RS=1，数据 RS=0，指令 |
| 5 | R/W | P2.6 | 5 | R/W | 读写信号。R/W=1，读操作 R/W=0，写操作 |
| 6 | E | P2.7 | 6 | E | 芯片使能信号 |
| 7 | DB0 | P0.0 | 7 | DB0 | 8位数据总线信号 |
| 8 | DB1 | P0.1 | 8 | DB1 | |
| 9 | DB2 | P0.2 | 9 | DB2 | |
| 10 | DB3 | P0.3 | 10 | DB3 | |
| 11 | DB4 | P0.4 | 11 | DB4 | |
| 12 | DB5 | P0.5 | 12 | DB5 | |
| 13 | DB6 | P0.6 | 13 | DB6 | |
| 14 | DB7 | P0.7 | 14 | DB7 | |
| 15 | PSB | P2.4 | 15 | PSB | 并/串模式选择。PSB=1，并行；PSB=0，串行 |
| 16 | N.C | P2.2 | 16 | N.C | 不连接 |
| 17 | /RST | P2.3 | 17 | /RST | 复位，低电平有效 |
| 18 | VOUT | -- | 18 | VOUT | 倍压输出脚 |
| 19 | A | +5V电源 | 19 | BLA | 背光源正极，接+5.0V |
| 20 | K | 地 | 20 | BLK | 背光源负极，接地 |

# ADC应用实现3
# --12864图形点阵LCD指标

## 12864是指LCD的屏幕分辨率为128×64。

- 12864中文汉字图形点阵液晶显示模块，可显示汉字及图形，内置8192 个中文汉字（16×16 点阵）、128个字符（8×16 点阵）及64X256 点阵显示RAM（GDRAM）。JGD12864图形点阵LCD的特性指标。

**12864图形点阵LCD主要技术参数**

| | |
|---|---|
| 显示容量 | 128×64个像素。每屏可显示4行8列共32个16×16点阵的汉字，或者4行16列共64个ASCII字符 |
| 工作电压范围 | 4.5V~+5V。对于STC单片机来说，推荐5.0V给12864供电 |
| 显示颜色 | 黄绿/蓝屏/灰屏 |
| LCD类型 | STN |
| 与MCU接口 | 8/4位并行，或者3位串行 |
| 屏幕尺寸 | 93×70×12.5mm（长×宽×高） |
| 多重模式 | 显示光标、画面移动、自定义字符、睡眠模式等 |

# ADC应用实现3
# --12864图形点阵LCD内部存储空间

在介绍下面内容前，对12864涉及到的存储空间进行简单的说明：

- 数据显示RAM
  - 即：Data Display Ram ，DDRAM。往里面写什么数据，LCD就会显示写入的数据。
- 字符发生ROM
  - 即：Character Generation ROM ，CGROM。里面存储了中文汉字的字模，也称作中文字库，编码方式有GB2312（中文简体）和BIG5（中文繁体）。

# ADC应用实现3
# --12864图形点阵LCD内部存储空间

- **字符发生RAM**
  - 即：Character Generation RAM， CGRAM。 12864内部提供了64×2字节的CGRAM，可用于用户自定义4个16×16字符，每个字符占用32个字节。
- **图形显示RAM**
  - 即：Graphic Display RAM，GDRAM。这一块区域用于绘图，往里面写什么数据，12864屏幕就会显示相应的数据，它与DDRAM的区别在于，往DDRAM中写的数据是字符的编码，字符的显示先是在CGROM中找到字模，然后映射到屏幕上，而往GDRAM中写的数据时图形的点阵信息，每个点用1比特来保存其显示与否。

# ADC应用实现3
# --12864图形点阵LCD内部存储空间

- **半宽字符发生器**
  - 即：Half height Character Generation ROM ，HCGROM。就是字母与数字，也就是ASCII码。
  - 12864内部有4行×32字节的DDRAM空间。但是某一时刻，屏幕只能显示2行×32字节的空间，剩余的这些空间呢可以用于缓存，在实现卷屏显示时就可以利用这些空间。DDRAM结构如下所示：

# ADC应用实现3
# --12864图形点阵LCD内部存储空间

## DDRAM结构如下所示：

80H、81H、82H、83H、84H、85H、86H、87H、88H、89H、8AH、8BH、8CH、8DH、8EH、8FH

90H、91H、92H、93H、94H、95H、96H、97H、98H、99H、9AH、9BH、9CH、9DH、9EH、9FH

A0H、A1H、A2H、A3H、A4H、A5H、A6H、A7H、A8H、A9H、AAH、ABH、ACH、ADH、AEH、AFH

B0H、B1H、B2H、B3H、B4H、B5H、B6H、B7H、B8H、B9H、BAH、BBH、BCH、BDH、BEH、BFH

## 地址与屏幕显示对应关系如下：

12864第一行：80H、81H、82H、83H、84H、85H、86H、87H

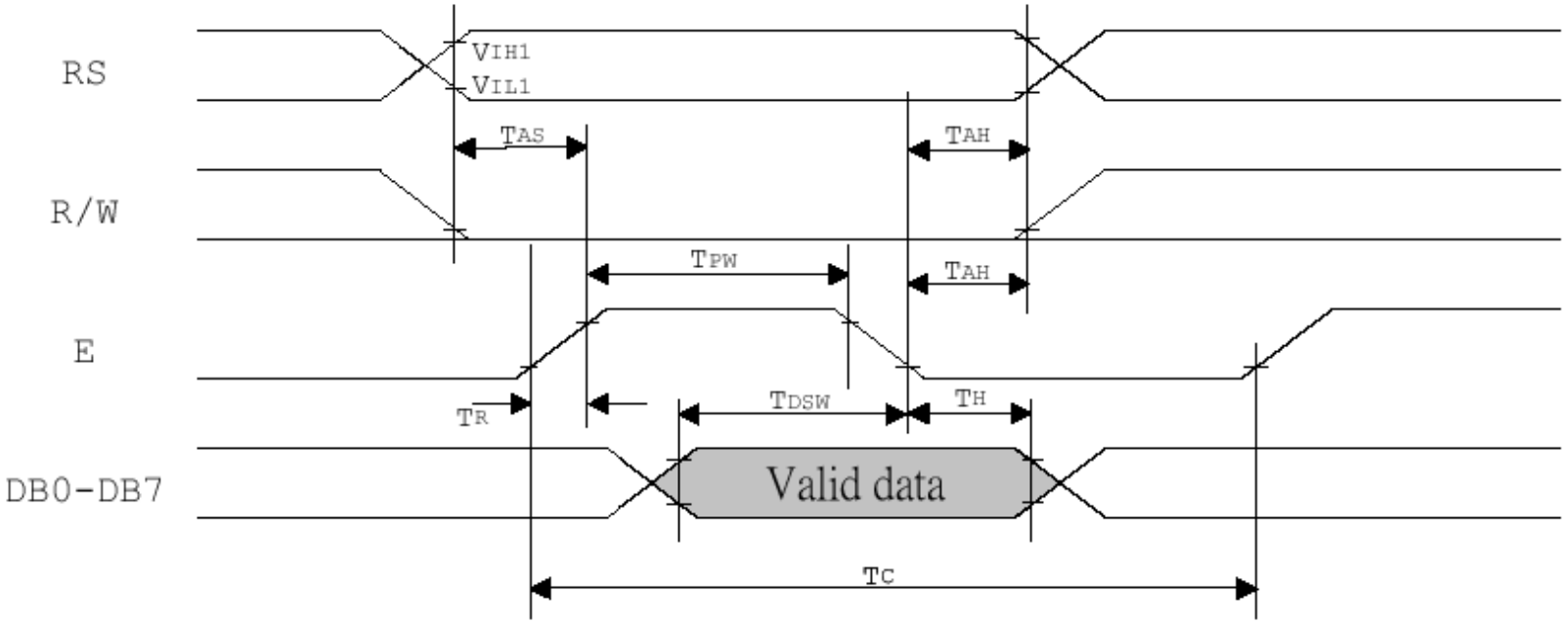
12864第二行：90H、91H、92H、93H、94H、95H、96H、97H

12864第三行：88H、89H、8AH、8BH、8CH、8DH、8EH、8FH

12864第四行：98H、99H、9AH、9BH、9CH、9DH、9EH、9FH

# ADC应用实现3
# --12864图形点阵LCD读写时序

## 写操作时序

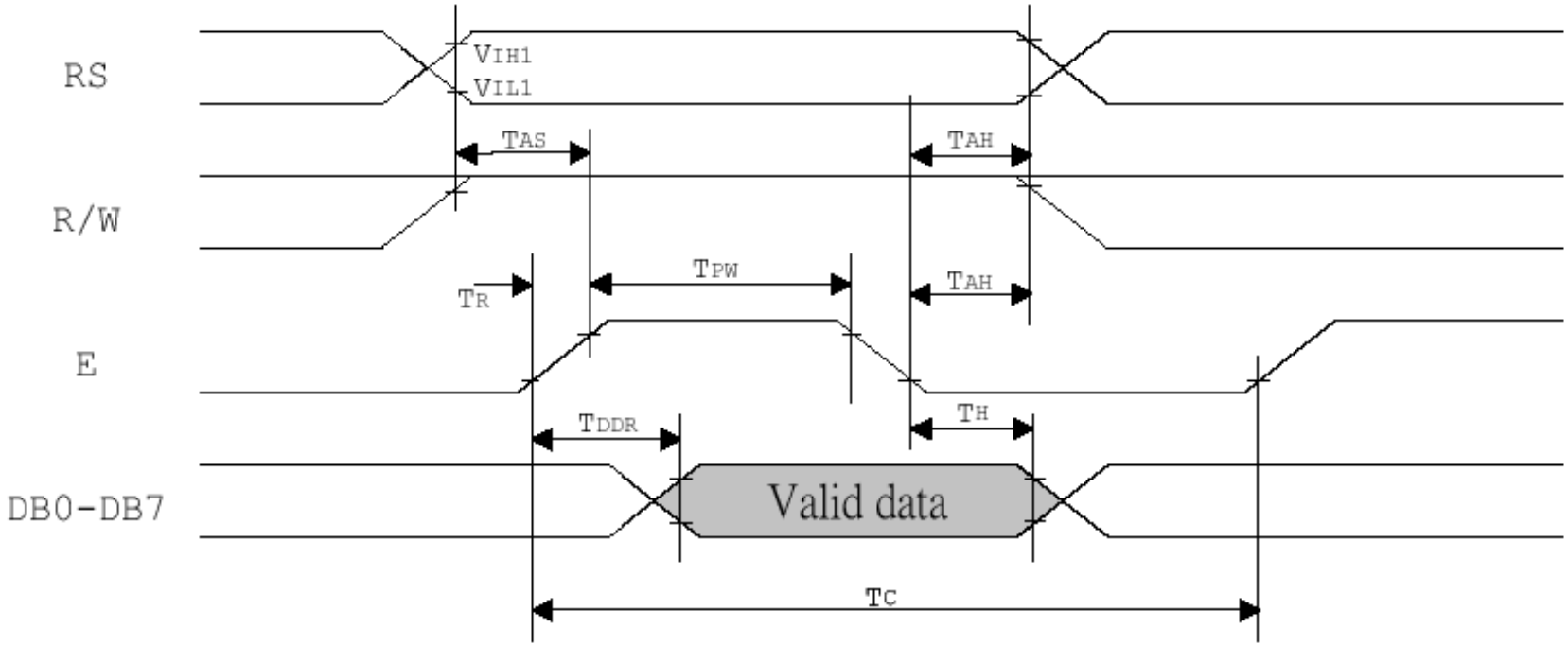


# ADC应用实现3
# --12864图形点阵LCD读写时序

- 首先，将R/W信号拉低。同时，给出RS信号，该信号为1或者0，用于区分数据和命令。
- 然后，将E信号拉高。当E信号拉高后，STC单片机将写入12864图形点阵 LCD的数据放在DB7~DB0数据线上。当数据有效一段时间$T_{DSW}$后，首先将E信号拉低。然后，数据再维持一段时间$T_H$。这样，数据就写到12864图形点阵LCD中。
- 最后，撤除/保持R/W信号。

# ADC应用实现3
# --12864图形点阵LCD读写时序

## 读操作时序



# ADC应用实现3
# --12864图形点阵LCD读写时序

- 首先，将R/W信号拉高。同时，给出RS信号，该信号为1或者0，用于区分数据和状态。
- 然后，将E信号拉高。当E信号拉高，并且延迟一段时间$t_{DDR}$后，12864图形点阵LCD将数据放在DB7~DB0数据线上。当维持一段时间$t_{TH}$后，将E信号拉低。
- 最后，撤除/保持R/W信号。

# ADC应用实现3
# ---12864图形点阵LCD命令和数据

- 在STC单片机对12864图形点阵LCD操作的过程中，会用到下面的命令

**注：12864图形点阵LCD提供了基本命令和扩展命令。**

**12864图形点阵LCD基本命令（RE=0）**

| 指令 | 指令操作码 | | | | | | | | | | 功能 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | RS | RW | DB7 | DB6 | DB5 | DB4 | DB3 | DB2 | DB1 | DB0 | |
| 清屏 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 将“20H”写到DDRAM，将DDRAM地址从AC（地址计数器）设置到“00” |
| 光标归位 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | - | 将DDRAM的地址设置为“00”，光标如果移动，则将光标返回到初始的位置。DDRRAM的内容保持不变 |
| 进入点设定 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | I/D | S | 指定在读数据和写数据时，设定光标移动的方向以及指定显示的移位。<br>I=0，递减模式。I=1，递增模；S=0，关闭整个移动。S=1，打开整个移动；S=0，关闭整个移动； |

# ADC应用实现3
# ---12864图形点阵LCD命令和数据

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 显示打开/关闭控制 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | D | C | B | 设置显示（D），光标（C）和光标闪烁（B）打开/关闭控制。<br>D=0，显示关闭；<br>D=1，打开显示；<br>C=0，关闭光标；<br>C=1，打开光标；<br>B=0，关闭闪烁；<br>B=1，打开闪烁； |
| 光标或者显示移动 | 0 | 0 | 0 | 0 | 0 | 1 | S/C | R/L | - | - | 设置光标移动和显示移动的控制位，以及方向，不改变DDRAM数据<br>S/C=0，R/L=0，光标左移；<br>S/C=0，R/L=1，光标右移；<br>S/C=1，R/L=0，显示左移，光标跟随显示移动<br>S/C=1，R/L=1，显示右移，光标跟随显示移动 |
| 功能设置 | 0 | 0 | 0 | 0 | 1 | DL | -- | RE=0 | - | - | DL=1（必须设置为1） RE=1，扩展指令集；RE=0，基本指令集 |

# ADC应用实现3
# ---12864图形点阵LCD命令和数据

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 设置CGRAM地址 | 0 | 0 | 0 | 1 | AC5 | AC4 | AC3 | AC2 | AC1 | AC0 | 在地址计数器中，设置CGRAM地址 |
| 设置DDRAM地址 | 0 | 0 | 1 | AC6 | AC5 | AC4 | AC3 | AC2 | AC1 | AC0 | 在计数器中，设置DDRAM地址 |
| 读忙标志和地址计数器 | 0 | 1 | BF | AC6 | AC5 | AC4 | AC3 | AC2 | AC1 | AC0 | 读BF标志，知道LCD屏内部是否正在操作。也可以读取地址计数器的内容 |
| 将数据写到RAM | 1 | 0 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 | 写数据到内部RAM（DDRAM/CGRAM/IRAM/GDRAM） |
| 从RAM读数据 | 1 | 1 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 | 从内部RAM（DDRAM/CGRAM/IRRAM/GDRAM）读取数据 |

# ADC应用实现3
# ---12864图形点阵LCD命令和数据

12864图形点阵LCD扩展命令（RE=1）

| 指令 | 指令操作码 | | | | | | | | | | 功能 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | RS | RW | DB7 | DB6 | DB5 | DB4 | DB3 | DB2 | DB1 | DB0 | |
| 待命模式 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 将“20H”写到DDRAM，将DDRAM地址从AC（地址计数器）设置到“00” |
| 卷动地址或IRAM地址选择 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | SR | SR=1，允许输入垂直卷动地址；SR=0，允许输入IRAM地址 |
| 反白选择 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | R1 | R0 | 选择4行中的任意一行做反白显示，并可决定是否 反白 |
| 休眠模式 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | SL | - | - | SL=1，脱离休眠模式；SL=0，进入休眠模式 |

# ADC应用实现3
# ---12864图形点阵LCD命令和数据

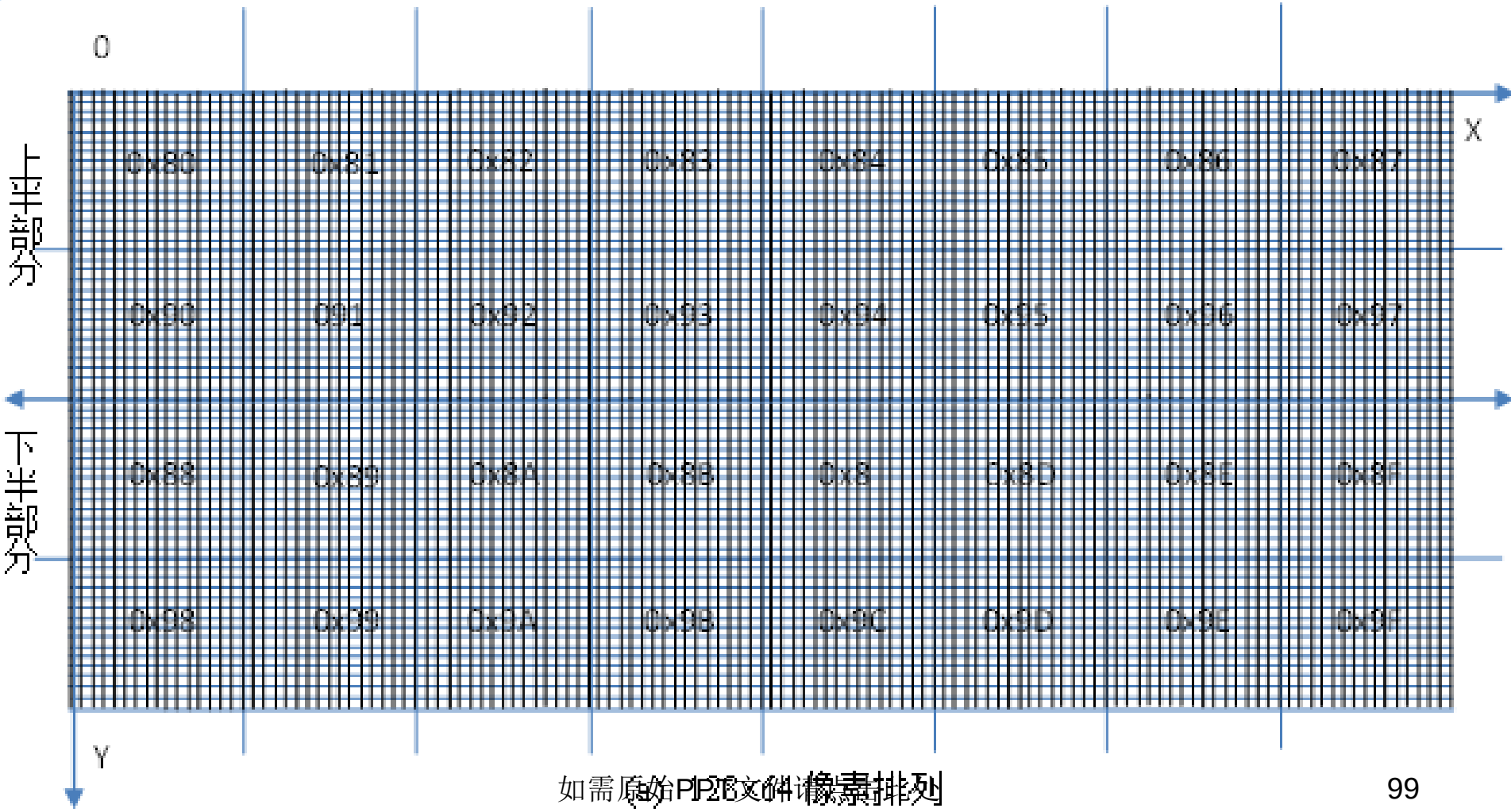
| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 扩充功能设定 | 0 | 0 | 0 | 0 | 1 | 1 | -- | RE=1 | G | 0 | RE=1，扩充指令集；RE=0，基本指令集<br>G=1，打开绘图模式；G=0，关闭绘图模式 |
| 设定IRAM地址或卷动地址 | 0 | 0 | 0 | 1 | AC5 | AC4 | AC3 | AC2 | AC1 | AC0 | SR=1，AC5~AC0为垂直卷动地址<br>SR=0，AC3~AC0为ICON IRAM地址 |
| 设置绘图RAM地址 | 0 | 0 | 1 | AC6 | AC5 | AC4 | AC3 | AC2 | AC1 | AC0 | 在地址计数器中，设置CGRAM地址 |

# ADC应用实现3
# --12864图形点阵字符/图象表示方法

**12864图形点阵128×64像素的构成方式。**

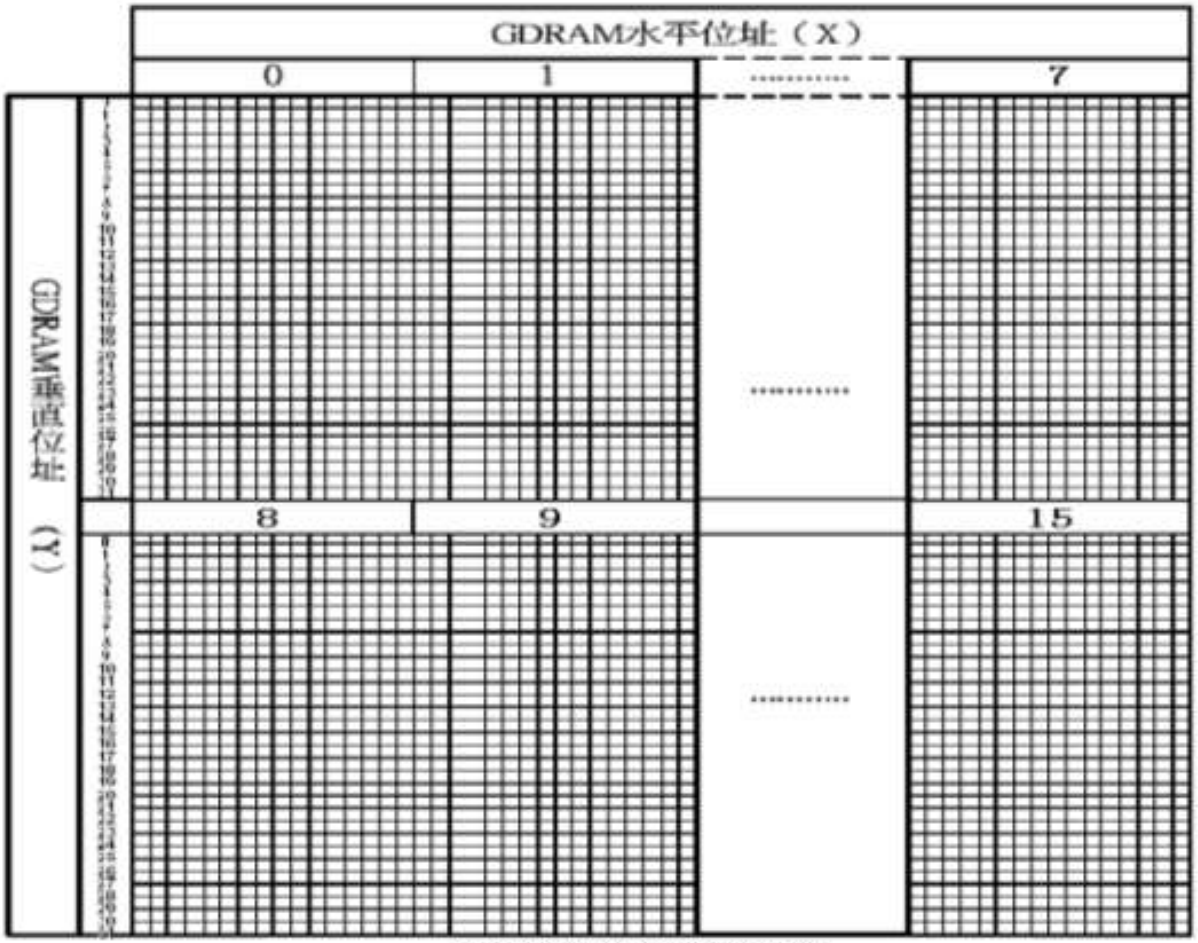
- 从后图(a)和(b)中可以看出，将屏幕分成上下部分，每部分包含32行和128列的像素。
- X方向以字节为单位，而Y方向以位为单位。X方向确定列，Y方向确定行。
- 该绘图显示RAM 提供128×8 个字节的存储空间。

# ADC应用实现3
## --12864图形点阵字符/图象表示方法




像素排列

# ADC应用实现3
## --12864图形点阵字符/图象表示方法




(b) GDRAM 地址分配情況

# ADC应用实现3
# --12864图形点阵字符/图象表示方法

## 字符/汉字表示方法

- 在字符模式下，将12864图形点阵 分为4行8列一共32个区域。
  - 每个区域包含16×16个像素值，可以显示两个ASCII码字符/数字，或者一个汉字。
  - 每行起始地址分别为：0x80、0x90、0x88、0x98。
  - 每行可显示区域分配了8个地址，一个地址包含两个字节。
  - 从图中可以看出，只要从32个地址中选择一个地址，就可以确定所要显示的字母/数字/汉字的位置。

# ADC应用实现3
# --12864图形点阵字符/图象表示方法

## 图像的表示方法

- 现在的问题就是给定了一个(x,y)坐标，如何向该坐标写数据。当向该坐标写0时，该点不亮；而向该坐标写1时，该点变亮。在更改绘图RAM的内容时，步骤包括：
  - 先连续写入水平与垂直的坐标值；
  - 再写入两个字节的数据到绘图RAM，而地址计数器（AC）会自动加一。

# ADC应用实现3
# --12864图形点阵字符/图象表示方法

## 写入所有绘图RAM的步骤包括：

- 关闭绘图显示功能。
- 先将水平的位元组坐标X，即：所在上半部分/下半的行，写入绘图RAM地址。
- 再将垂直坐标Y，即：所在屏幕的上半部分还是下半部分，写入绘图RAM地址；
- 将D15~D8写入到RAM中；
- 将D7~D0写入到RAM中；
- 打开绘图显示功能。

# ADC应用实现3
# --12864图形点阵字符/图象表示方法

在该设计中，在12864点阵中显示波形。策略是：

- 声明一个16×64大小的unsigned char类型数组，即：

  unsigned char pix[16][64];

  - 该声明的目的是，将128×64像素点分成16×64的区域。每个区域8列1行像素。8列正好是8位，一个无符号字节。因此，该数组可以表示12864图形点阵LCD中的所有128×64像素的当前状态。

# ADC应用实现3
## --12864图形点阵字符/图象表示方法

将该pix数组所有元素赋初值为0，即128×64像素的当前状态都是0，不亮。很明显，在12864图形点阵LCD上画波形实际上就是让需要亮的像素点赋值为1即可。显示波形，包括正弦和三角波：

- 为了在LCD上显示一屏数据，因此每次得到128个采样，每个采样对应于x坐标；
- 每个采样所对应的离散的值是10位，范围在[0，1023]之间，经过量化处理，即：将该离散的值/16，范围限制在[0，63]之间。也就是说，每个采样的幅度在[0，63]。在该设计中，每个采样的幅度表示为y[i]，i=0~128。

# ADC应用实现3
# --12864图形点阵字符/图象表示方法

- 下面得到每个采样和pix数组之间的对应关系。pix数组的每个元素的索引和y[i]存在下面的对应关系，即：

  pix[i/8][y[i]], i=0~128

  - i/8将x坐标对应到16个区域中
  - y[i]是每个采样点经过量化处理后的幅度，很明显是pix数组中每个像素的行坐标

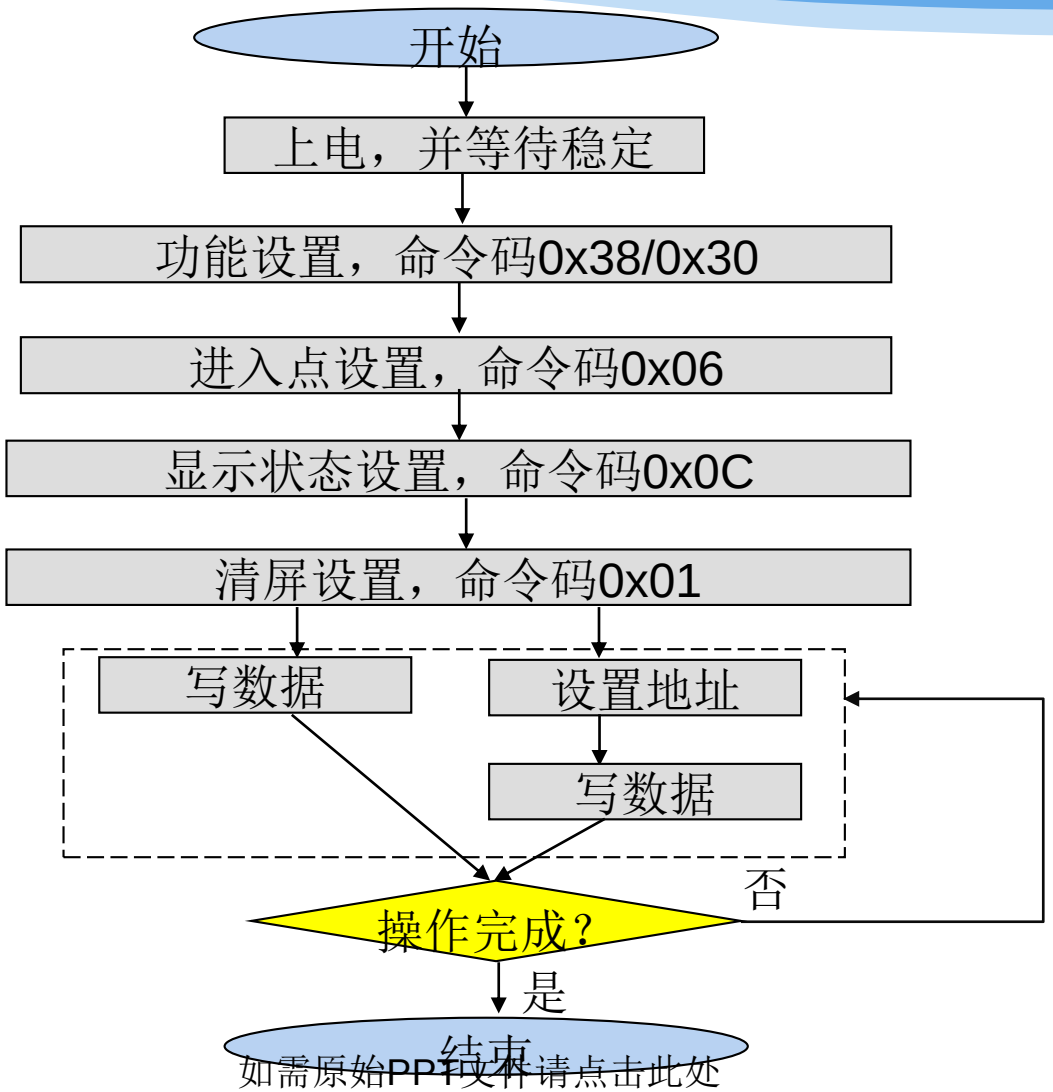
- 下面具体是pix数组的每个元素是unsigned char类型，也就是8个像素。表示为：

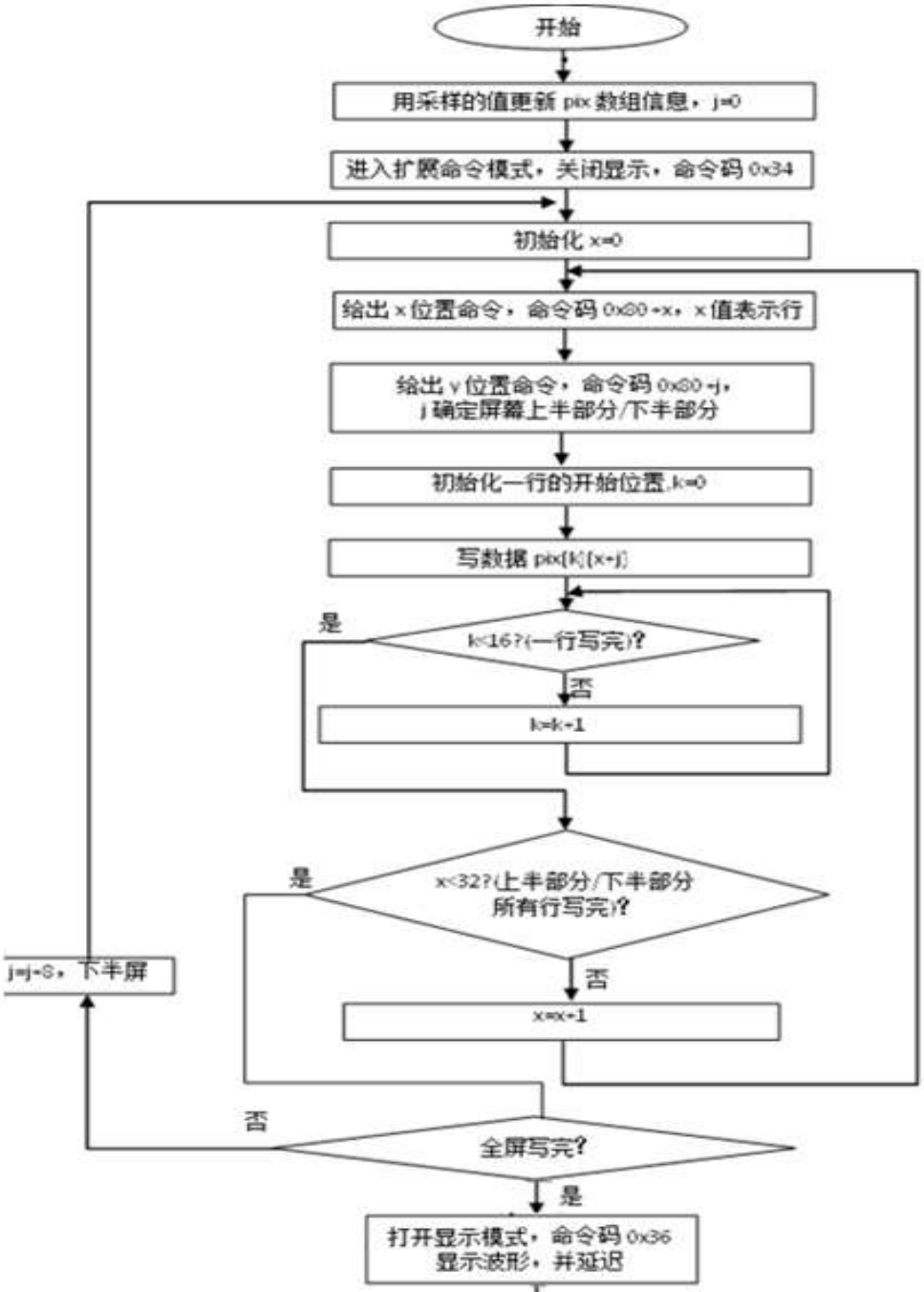
  pix[i/8][y[i]]=(0x80>>(i%8));

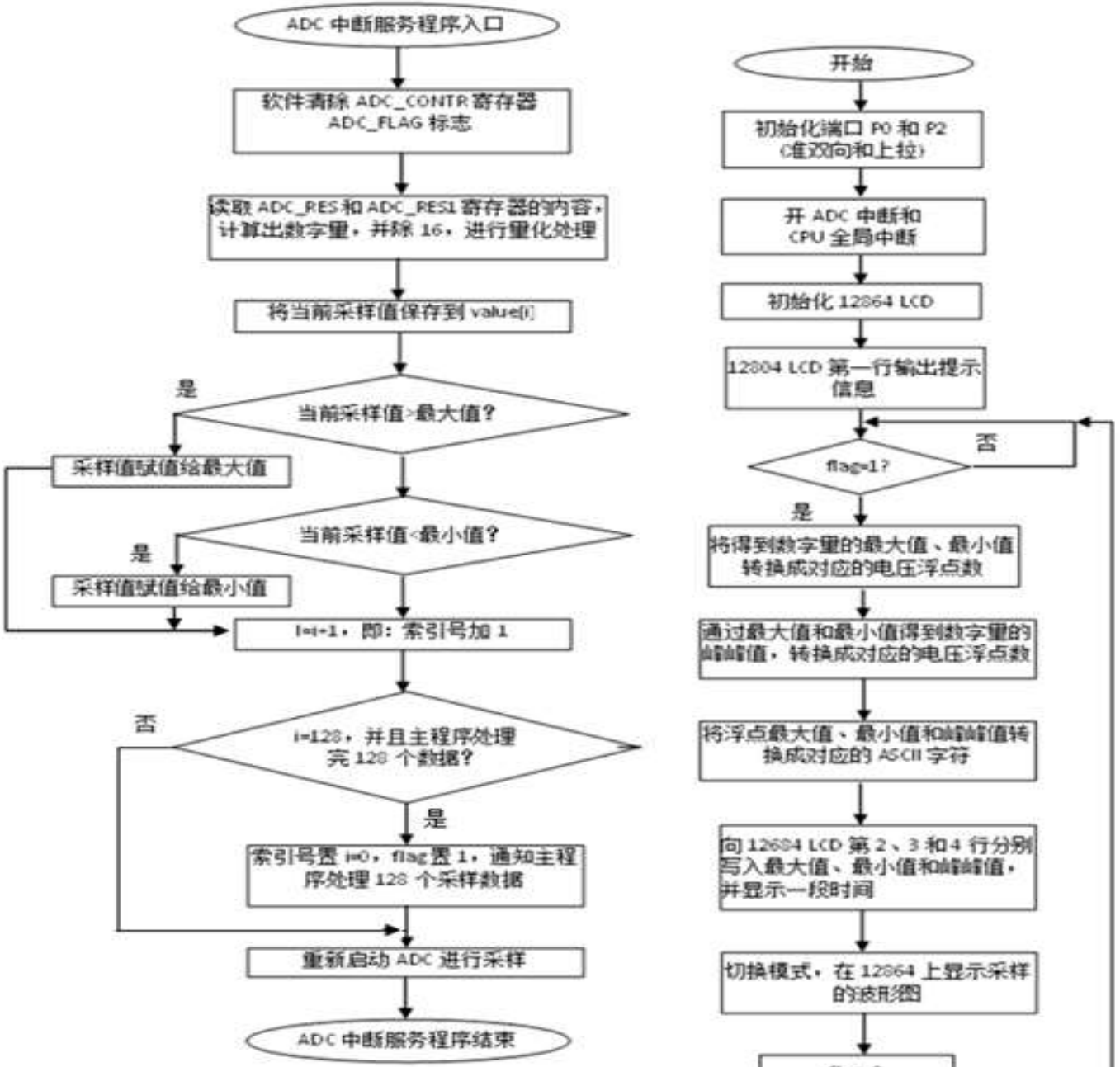
# ADC应用实现3
# --12864图形点阵字符/图象表示方法

- (0x80>>(i%8))具体含义如下：
  - i%8得到每个x坐标i在pix数组每个区域i/8内的偏移位置。
  - 0x80表示一个8位的像素中，有一个像素是亮的，即：亮点。
  - (0x80>>(i%8))意思即表示：在x坐标i在pix数组每个区域i/8内的偏移位置上赋值为1。
- 重复步骤4，128次，将128个采样点的幅度具体表示在pix数组中。

# ADC应用实现3
# --12864 LCD字符模式初始化和显示字符操作过程








（a）ADC 中断处理程序流程　　（b）主程序处理流程

# ADC应用实现3
# --ADC外部输入信号要求

**由于STC15系列单片机是单电源供电，所以，其内部集成的ADC模块也是单电源供电。**

- **典型地，供电电压是+5V。**
  - **因此，直接输入到ADC模块的信号范围在0~Vcc（单片机供电电压）范围内。**

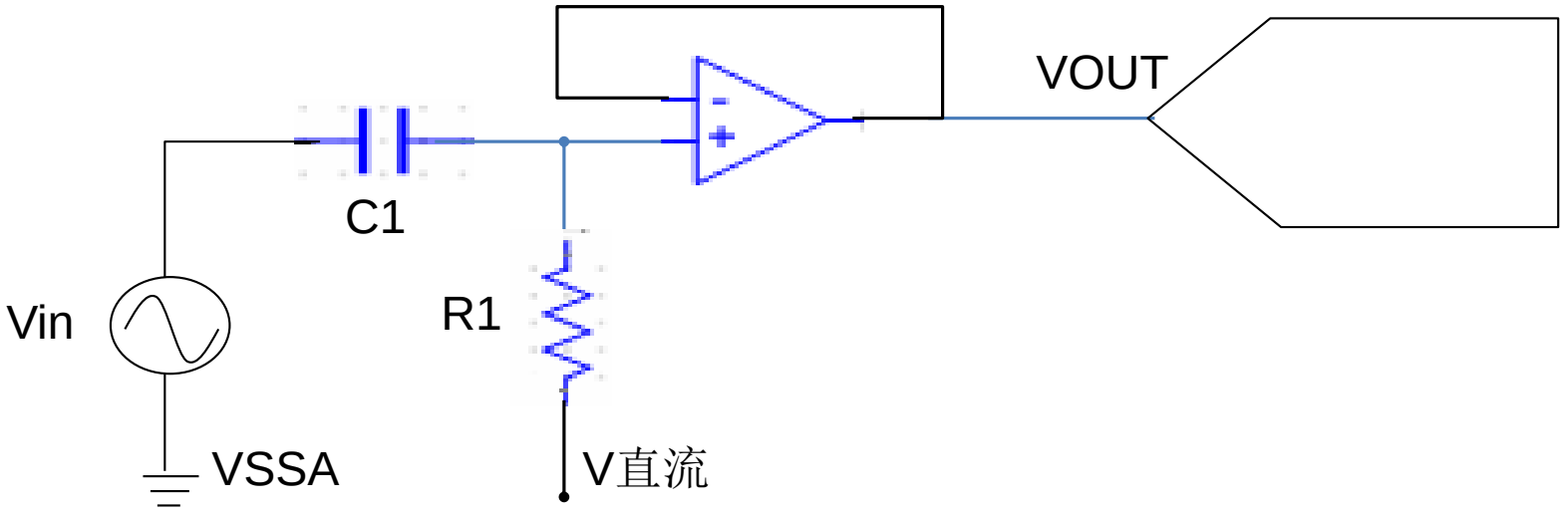
# ADC应用实现3
# --ADC外部输入信号要求

## 输入信号采用的方式包括：

- 在信号源将信号直接接入到STC单片机ADC输入引脚时，在信号里给出直流偏置。
  - 交流信号在这个直流偏置信号上摆动，摆动的范围在0~5V单片机供电电压内，即：给出的信号是单极性的信号。

# ADC应用实现3
## --ADC外部输入信号要求

- 信号源直流偏置为0，直接给出交流信号，这个交流信号经过下面的单电源放大器构成的电压跟随器的处理，引入直流偏置，然后将跟随器的输出信号Vout连接到STC单片机ADC的输入引脚上。



# ADC应用实现3
# --ADC外部输入信号要求

- 当没有交流信号输入时，即信号的直流通路输出满足：

$$V_{out}' = V_{直流}$$

- 当有交流小信号输入时，信号的交流通路（直流偏置接地）输出满足：

$$V_{out}'' = V_{IN}$$

其中：

- 输入信号Vin的电压通过电阻R1降低到GND（地）。使用叠加定理，则总输出为：

$$V_{out} = V_{out}' + V_{out}'' = V_{IN} + V_{直流}$$

# ADC应用实现3
# --具体实现过程

【例】采集外部交流信号，并在18624图形点阵LCD上显示的C语言描述的例子。

12864.h文件

```
#ifndef _12864_                 //条件编译命令，如果没有定义12864
#define _12864_                 //则定义12864
#include "reg51.h"              //包含reg51.h头文件
#include "intrins.h"            //包含intrins.h头文件
sbit LCD12864_RS=P2^5;          //声明sbit类型变量LCD12864_RS为P2.5引脚
sbit LCD12864_RW=P2^6;          //声明sbit类型变量LCD12864_RW为P2.6引脚
sbit LCD12864_E =P2^7;          //声明sbit类型变量LCD12864_E为P2.7引脚
sbit LCD12864_PSB=P2^4;         //声明sbit类型变量LCD12864_PSB为P2.4引脚
sfr  LCD12864_DB=0x80;         //声明LCD12864_DB寄存器地址0x80（P0端口）
sfr  P0M1=0x93;                 //定义P0端口P0M1寄存器地址0x93
sfr  P0M0=0x94;                 //定义P0端口P0M0寄存器地址0x94
```

# ADC应用实现3
# --具体实现过程

```
sfr  P2M1=0x95;                          //定义P2端口P2M1寄存器地址0x95
sfr  P2M0=0x96;                          //定义P2端口P2M0寄存器地址0x96
void lcdwait();                          //定义子函数lcdwait类型
void lcdwritecmd(unsigned char cmd);     //定义子函数lcdwritecmd类型
void lcdwritedata(unsigned char dat);    //定义子函数lcdwritedata类型
void lcdinit();                          //定义子函数lcdinit类型
void lcdsetcursor(unsigned char x, unsigned char y);
                                         //定义子函数lcdsetcursor类型
void lcdshowstr(unsigned char x, unsigned char y,unsigned char *str);
                                         //定义子函数lcdshowstr类型
void drawpoint(unsigned char y[]);       //声明子函数drawpoint类型
#endif                                   //条件预编译命令结束
```

# ADC应用实现3
# --具体实现过程

```
#include "12864.h"
void lcdwait()                          //定义lcdwait函数，用于检测12864的忙标志
{       LCD12864_DB=0xFF;               //读取P0端口前，先给P0端口置位0xFF
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        LCD12864_RS=0;                  //将LCD12864_RS指向的P2.5引脚拉低
        LCD12864_RW=1;                  //将LCD12864_RW指向的P2.6引脚拉高
        LCD12864_E=1;                   //将LCD12864_E指向的P2.7引脚拉高
        while(LCD12864_DB & 0x80);      //如果12864LCD内部忙，则等待
        LCD12864_E=0;                   //将LCD12864_E指向的P2.7引脚拉低
        _nop_();
        _nop_();
        _nop_();
        _nop_();
}
```

# ADC应用实现3
# --具体实现过程

```
void lcdwritecmd(unsigned char cmd)
{       lcdwait();
        _nop_();
        _nop_();
        _nop_();
        _nop_();
        LCD12864_RS=0;          //将LCD12864_RS指向的P2.5引脚拉低
        LCD12864_RW=0;          //将LCD12864_RW指向的P2.6引脚拉低
        LCD12864_DB=cmd;//将命令码cmd放到LCD12864_DB 指向的P0端口
        LCD12864_E=1;           //将LCD12864_E指向的P2.7引脚拉高
```

# ADC应用实现3
# --具体实现过程

```
    _nop_();
    _nop_();
    _nop_();
    _nop_();
     LCD12864_E=0;          //将LCD12864_E指向的P2.7引脚拉低
}
```

# ADC应用实现3
# --具体实现过程

```
void lcdwritedata(unsigned char dat)
{
    lcdwait();
    _nop_();
    _nop_();
    _nop_();
    _nop_();
    LCD12864_RS=1;              //将LCD12864_RS指向的P2.5引脚拉高
   LCD12864_RW=0;               //将LCD12864_RW指向的P2.6引脚拉低
   LCD12864_DB=dat;             //将数据dat放到LCD12864_DB 指向的P0端口
```

# ADC应用实现3
# --具体实现过程

```
    LCD12864_E=1;  //将LCD12864_E指向的P2.7引脚拉高
    _nop_();
    _nop_();
    _nop_();
    _nop_();
   LCD12864_E=0;          //将LCD12864_E指向的P2.7引脚拉低
}
```

# ADC应用实现3
# --具体实现过程

```
void lcdinit()                    //定义子函数lcdinit，用于初始化12864
{
        lcdwritecmd(0x38);//调用函数lcdwritecmd，给12864发命令0x38
         lcdwritecmd(0x06);//调用函数lcdwritecmd，给12864发命令0x06
         lcdwritecmd(0x01);//调用函数lcdwritecmd，给12864发命令0x01
         lcdwritecmd(0x0c);//调用函数lcdwritecmd，给12864发命令0x0c
}
```

# ADC应用实现3
# --具体实现过程

```
//声明lcdsetcursor函数，设置显示RAM的地址，x和y表示在12864的列和行参数
void lcdsetcursor(unsigned char x, unsigned char y)
{       unsigned char address;              //声明无符号char类型变量address
        if(y==0)                            //如果是第一行
                address=0x80+x;             //从存储器地址0x80的位置开始
        else if(y==1)                       //如果是第二行
                address=0x90+x;             //从存储器地址0x90的位置开始
        else if(y==2)                       //如果是第三行
                address=0x88+x;             //从存储器地址0x88的位置开始
```

# ADC应用实现3
# --具体实现过程

```
    else                                    //如果是第四行
        address=0x98+x;                     //从存储器地址0x98的位置开始
        lcdwritecmd(address|0x80);          //写12864存储器地址命令
}
```

# ADC应用实现3
# --具体实现过程

```
void lcdshowstr(unsigned char x, unsigned char y,unsigned char *str)
                                        //在12864上指定的x和y位置，显示字符
{
        lcdsetcursor(x,y);              //设置写RAM的地址
        while((*str)!='\0')             //如果字符没有结束
        {
                lcdwritedata(*str);//将当前字符写到RAM，即在12864上显示
                str++;                  //指向下一个字符
        }
}
```

# ADC应用实现3
# --具体实现过程

```
void drawpoint(unsigned char y[])   //声明子函数drawpoint，在12864上显示波形
{
        unsigned char i,j,k;                    //定义无符号char类型变量i，j和k
        unsigned long int l;                    //定义无符号长整型变量l
        unsigned char x;                        //定义无符号char类型变量x
        xdata unsigned char pix[16][64];    //在xdata定义二维数组pix[16][64]
        for(i=0;i<16;i++)                       //二重循环初始化pix数组为0
          for(j=0;j<64;j++)
                pix[i][j]=0;
                    for(i=0;i<128;i++)//用采样的数据数组y[128]修改pix数组的值
                    pix[i/8][y[i]]=(0x80>>(i%8));//在给定的x和y像素位置上置1
```

# ADC应用实现3
# --具体实现过程

```
for(i=0,j=0;i<9;i+=8,j+=32)                    //该循环确定屏幕上半部分和下半部分
{       for(x=0;x<32;x++)                      //该循环定位所在屏幕所在的行
        {       lcdwritecmd(0x34);             //使用扩展命令，关闭图像显示模式
                lcdwritecmd(0x80+x);           //写x坐标信息
                lcdwritecmd(0x80+i);           //写y坐标信息
                lcdwritecmd(0x30);             //使用基本命令
                for(k=0;k<16;k++);             //该循环连续写指定一行的128个像素
                lcdwritedata(pix[k][x+j]); //16列，每列8个像素
          }
}
lcdwritecmd(0x36);                             //打开显示模式
for(l=0;l<500000;l++);                          //显示图像并持续一段时间
}
```

# ADC应用实现3
# --具体实现过程

main.c文件

```
#include "reg51.h"
#include "stdio.h"
#include "12864.h"
#define ADC_POWER    0x80            //定义ADC_POWER的值0x80
#define ADC_FLAG     0x10            //定义ADC_FLGA的值0x10
#define ADC_START    0x08            //定义ADC_START的值0x08
#define ADC_SPEEDLL  0x00            //定义ADC_SPEEDLL的值0x00
#define ADC_SPEEDL   0x20            //定义ADC_SPEEDL的值0x20
#define ADC_SPEEDH   0x40            //定义ADC_SPEEDH的值0x40
#define ADC_SPEEDHH  0x60            //定义ADC_SPEEDHH的值0x60
```

# ADC应用实现3
# --具体实现过程

```
sfr AUXR =0x8E;                 //声明AUXR寄存器的地址0x8E
sfr ADC_CONTR =0xBC;            //声明ADC_CONTR寄存器的地址0xBC
sfr ADC_RES =0xBD;              //声明ADC_RES寄存器的地址0xBD
sfr ADC_RESL =0xBE;             //声明ADC_RESL寄存器的地址0xBE
sfr P1ASF =0x9D;                //声明P1ASF寄存器的地址0x9D
unsigned char ch=4;             //声明char类型变量ch
bit flag=1;                     //声明bit类型变量flag
unsigned char max_tstr[10],min_tstr[10],avg_tstr[10];
                                //声明全局无符号char类型数组
unsigned int tmp=0;             //声明全局无符号int类型变量tmp
xdata unsigned char value[128]; //xdata区域声明无符号char类型数组value
unsigned int max_value=0 ,min_value=1024,avg_value=10;
                                //声明全局无符号int类型变量
unsigned char inc=0;            //声明全局无符号char类型变量inc
```

# ADC应用实现3
# --具体实现过程

```
void adc_int() interrupt 5                    //声明ADC中断服务程序
{
    unsigned char i=0;                        //定义无符号char类型变量i
    ADC_CONTR &=!ADC_FLAG;                    //清除ADC_FLAG标志
    tmp=(ADC_RES*4+ADC_RESL);                 //得到输入模拟量对应的10位转换数字量
    if(inc!=128 && flag==0)                   //如果没有采够128个数据，并且flag=0
    {
        value[inc]=tmp/16;                    //数字量除16，量化到0~64用于将来显示
        if(tmp>max_value)                     //如果当前采样的数字量大于最大值的数字量
            max_value=tmp;                    //将当前采样的数字量赋值给最大值
        if(tmp<min_value)                     //如果当前采样的数字量小于最小的数字量
            min_value=tmp;                    //将当前采样的数字量赋值给最小值
        inc++;                                //索引号递增1
    }
```

# ADC应用实现3
# --具体实现过程

```
    else                                //采满128个采样数据
        {
            inc=0;                      //索引号归零
            flag=1;                     //将flag位置1
        }
ADC_CONTR=ADC_POWER |ADC_SPEEDLL | ADC_START | ch;
}
```

# ADC应用实现3
# --具体实现过程

```
void main()
{
    long unsigned int i;
    P0M0=0;                 //设置P0M0和P0M1寄存器，将P0端口
    P0M1=0;                 //设置为准双向/弱上拉
    P2M0=0;                 //设置P2M0和P2M1寄存器，将P2端口
    P2M1=0;                 //设置为准双向/弱上拉
    P1ASF=0xFF;             //将P1端口设置为模拟输入
    ADC_RES=0;              //将ADC_RES寄存器清0
```

# ADC应用实现3
# --具体实现过程

```
//配置ADC控制寄存器ADC_CONTR
    ADC_CONTR=ADC_POWER|ADC_SPEEDLL | ADC_START | ch;
    for(i=0;i<10000;i++);                    //延迟一段时间
    IE=0xA0;                                 //CPU允许响应中断，允许ADC中断
    lcdinit();                               //初始化12864
    lcdshowstr(0,0,"测量交流信号 ");  //在12864第一行打印信息
    for(i=0;i<600000;i++);                   //延迟一段时间
    lcdwritecmd(0x01);                       //给12684发清屏命令
```

# ADC应用实现3
# --具体实现过程

```
while(1)                                    //无限循环
{
   if(flag==1)                              //如果flag标志为1
    {
        lcdinit();                          //初始化12864
        sprintf(max_tstr,"%+1.4f",(max_value*5.0)/1024);
                                            //将浮点最大值转换成字符串
        sprintf(min_tstr,"%+1.4f",(min_value*5.0)/1024);
                                            //将浮点最小值转换成字符串
         sprintf(avg_tstr,"%+1.4f",((max_value-min_value)*5.0)/1024);
                                            //将计算得到的浮点峰峰值转换成字符串
         max_value=0;                       //将max_value重新赋值为0
         min_value=1024;                    //将min_value重新赋值为1024
```

# ADC应用实现3
# --具体实现过程

```
lcdshowstr(0,1,"MAX: ");        //在12684第二行的开头打印“MAX:”信息
lcdshowstr(2,1,max_tstr);       //在12684第二行继续打印采集信号的最大值
lcdshowstr(5,1," V");           //在12684第二行继续打印“V:”信息
lcdshowstr(0,2,"MIN: ");        //在12684第三行的开头打印“MIN:”信息
lcdshowstr(2,2,min_tstr);       //在12684第三行继续打印采集信号的最小值
lcdshowstr(5,2," V");           //在12684第三行继续打印“V:”信息
lcdshowstr(0,3,"PTP: ");        //在12684第四行的开头打印“PTP:”信息
lcdshowstr(2,3,avg_tstr);       //在12684第四行继续打印采集信号的峰峰值
lcdshowstr(5,3," V");           //在12684第四行继续打印“V:”信息
for(i=0;i<300000;i++);          //延迟一段时间
```

# ADC应用实现3
# --具体实现过程

```
    lcdwritecmd(0x01);              //清屏
    drawpoint(value);               //绘制采集信号的128个值的波形图
    flag=0;                         //将flag标志置0
    }
  }
}
```

# ADC应用实现3

**下载和分析设计的步骤主要包括：**

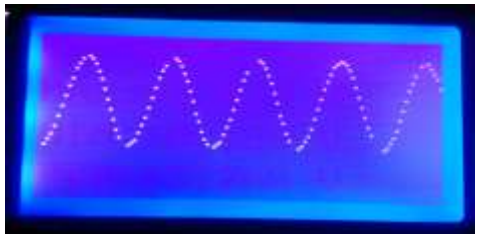
- **打开STC-ISP软件，在该界面内，选择硬件选项。将“输入用户程序运行时的IRC频率”设置为22.000MHz。**
- **单击下载/编程按钮，按前面的方法下载设计到STC单片机。**
- **打开信号源，将信号源的输出分别连接到STC开发板的P1.4和GND。**

# ADC应用实现3

- 观察12864图形点阵LCD上的输出，第一屏、第二屏和第三屏显示，如图所示。


12684输出的第一屏信息


12684输出的第二屏信息



12684输出的第三屏信息

# ADC应用实现4

在前面的设计中，在转换成模拟电压值的时候是基于单片机的供电电压VCC，典型的+5V。如果单片机的外部供电电压发生变化，则转换出来的电压就一定存在误差，而且误差随着Vcc的变化而不确定。

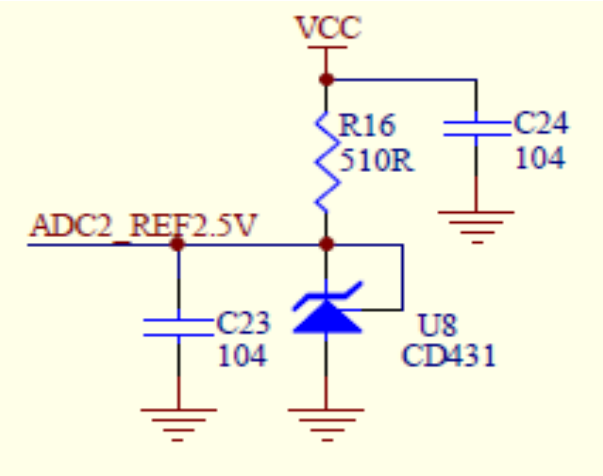
# ADC应用实现4

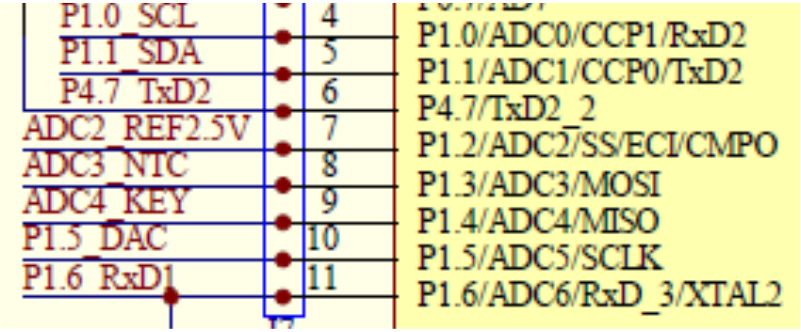
因此，如果能有一个稳定的电压源参考，然后基于此参考电源进行计算，所得到的被测量信号的模拟电压值误差只于STC15系列单片机ADC内部的转换误差，以及参考源的误差有关，和单片机的供电电压无关。这样，就很容易计算出输入模拟信号电压的相对误差和绝对误差。

- 在该设计中，通过基准电压源计算测量信号的结果，并通过串口1进行显示。

# ADC应用实现4
## --测量信号校准原理

在STC学习板上，提供了TL431基准参考电压源，该参考源默认输出+2.5V的参考信号。该信号连接到STC单片机的P1.2引脚上。



# ADC应用实现4
# --测量信号校准原理

**TL431的技术指标主要包括：**

- **在25℃时，误差为0.5%（B级）；误差为1%（A级）；误差为2%（标准级）；**
- **在0~70℃范围内，温漂为6mV；在 - 40~+85℃时，温漂为14mV。**

# ADC应用实现4
# --信号输入电路

在STC学习板上，提供了带有负温度系数NTC热敏电阻SDNT2012X103F3950FTF的信号输入电路。

- 该热敏电阻的负温度系数是指，即当温度升高的时候，热敏电阻值减少；
- 而当温度降低的时候，热敏电阻值增加。当在标称温度（25℃）时，热敏电阻的值为10KΩ。

# ADC应用实现4
# --信号输入电路

- 在温度T时的热敏电阻的值由下面的公式进行计算：

$$R_T = R_N \cdot \exp B(1/T - 1/T_N)$$

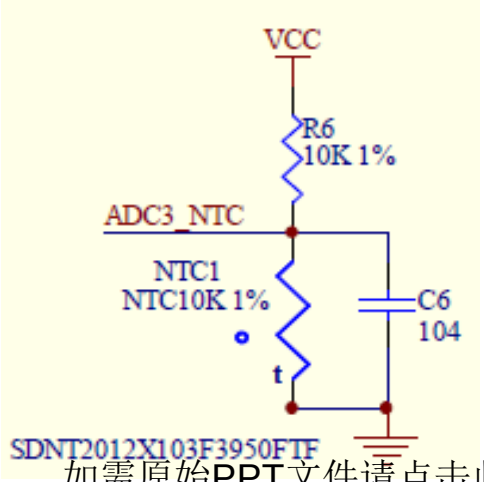
  - $R_T$：是指在温度 T（单位为开氏温度K）时的NTC热敏电阻阻值。
  - $R_N$：在额定温度 $T_N$（单位为开氏温度K）时的 NTC 热敏电阻阻值。
  - T：规定温度（单位为开氏温度K）。
  - B：NTC 热敏电阻的材料常数，又叫热敏指数。
  - exp：以自然数e 为底的指数（ e = 2.71828 …）。

# ADC应用实现4
# --信号输入电路

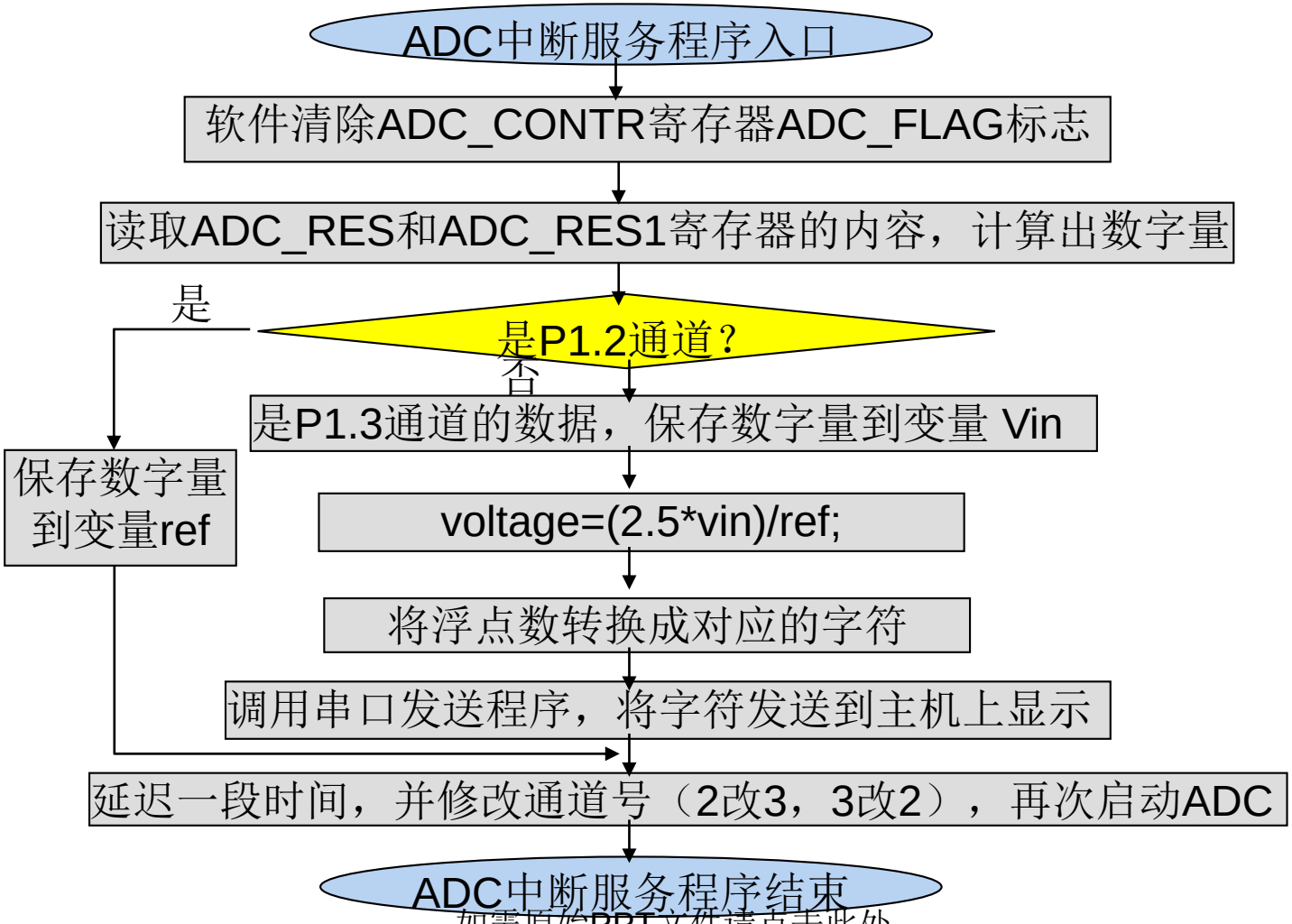
在图所示的电路中，热敏电阻作为分压网络的一部分与R6电阻连接在一起

- ADC_NTC网络连接到单片机P1.3引脚上。
  - ADC_NTC上的电压由下式确定：

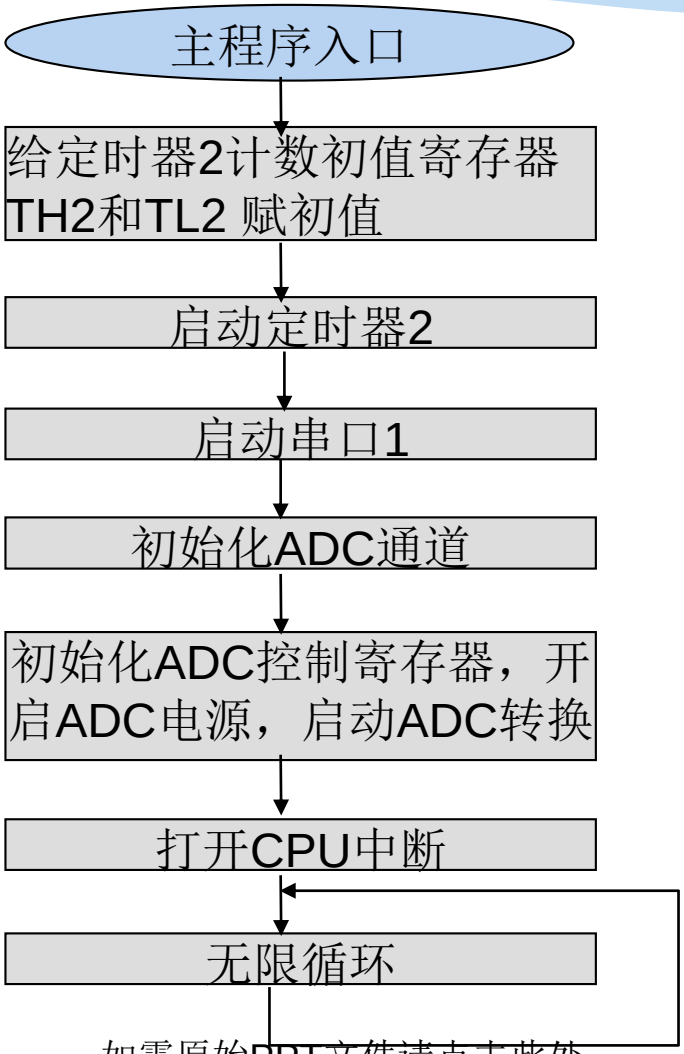
$$V_{ADC\_NTC}=(V_{CC}\times R_{NTC1})/(R_{NTC1}+R_6)$$



# ADC应用实现4
## --ADC中断程序处理流程



# ADC应用实现4
## --主程序处理流程



# ADC应用实现4
# --具体实现过程

## 【例】利用外部参考电压精确测量外部输入电压值C语言描述

```
#include "reg51.h"
#include "stdio.h"
#define OSC       18432000L        //声明单片机主时钟频率为18432000Hz
#define BAUD      9600             //声明单片机串口1通信波特率时钟
#define ADC_POWER     0x80         //声明ADC_POWER的值为0x80
#define ADC_FLAG   0x10            //声明ADC_FLAG的值为0x10
#define ADC_START   0x08           //声明ADC_START的值为0x08
#define ADC_SPEEDLL   0x00         //声明ADC_SPEEDLL的值为0x00
#define ADC_SPEEDL   0x20          //声明ADC_SPEEDH的值为0x20
#define ADC_SPEEDH   0x40          //声明ADC_SPEEDH的值为0x40
#define ADC_SPEEDHH  0x60          //声明ADC_SPEEDHH的值为0x60
```

# ADC应用实现4
# --具体实现过程

```
sfr T2H       =0xD6;          //声明T2H寄存器的地址为0xD6
sfr T2L       =0xD7;          //声明T2L寄存器的地址为0xD7
sfr AUXR      =0x8E;          //声明AUXR寄存器的地址为0x8E
sfr ADC_CONTR =0xBC;          //声明ADC_CONTR寄存器的地址为0xBC
sfr ADC_RES   =0xBD;          //声明ADC_RES寄存器的地址为0xBD
sfr ADC_RESL  =0xBE;          //声明ADC_RESL寄存器的地址为0xBE
sfr P1ASF     =0x9D;          //声明P1ASF寄存器的地址为0x9D
unsigned char ch=2;           //声明无符号char类型全局变量ch=2
float voltage=0;              //声明浮点类型全局变量voltage=0
unsigned char tstr[5];        //声明无符号char类型全局数组tstr
unsigned int ref=0,vin=0;     //声明无符号int类型全局变量ref和vin
```

# ADC应用实现4
# --具体实现过程

```
void SendData(unsigned char dat)          //声明串口发送子函数
{
        while(!TI);                       //如果TI不为1，正在发送数据，则等待
        TI=0;                             //TI标志清零
        SBUF=dat;                         //dat写入串口1发送寄存器SBUF中
}
```

# ADC应用实现4
# --具体实现过程

```
void adc_int() interrupt 5                    //声明ADC中断服务程序adc_int
{
        unsigned char i=0;                    //声明无符号char类型变量i
       unsigned long int j=0;                 //声明无符号long int类型变量j
       ADC_CONTR &=!ADC_FLAG;                 //清ADC_FLAG变量
       if(ch==2)                              //如果是参考电压源TL431通道
       {
           ref=(ADC_RES*4+ADC_RESL);
                                  //将参考电压所对应的数字量保存到变量ref中
       }
```

# ADC应用实现4
# --具体实现过程

```
else if(ch==3)                          //如果是热敏电阻分压输入通道
{
   vin=(ADC_RES*4+ADC_RESL);
                                        //将分压所对应的数字量保存到变量vin中
   voltage=(2.5*vin)/ref;                //计算分压的浮点电压值
   sprintf(tstr,“%1.4f”,voltage);   //转换成对应的字符串tstr
   SendData(‘\r’);                      //串口发送回车符
    SendData(‘\n’);                     //串口发送换行符
    for(i=0;i<5;i++)
     SendData(tstr[i]);                 //串口发送分压对应的ASCII字符
}
```

# ADC应用实现4
# --具体实现过程

```
    if(ch==2)                          //如果当前通道是2
            ch=3;                      //则将通道号修改为3
      else if(ch==3)                   //如果当前通道是3
            ch=2;                      //则将通道号修改为2
      for(j=0;j<=80000;j++);           //延迟一段时间
      ADC_RES=0;                       //ADC_RES寄存器清零
      ADC_RESL=0;                      //ADC_RESL寄存器清零
      ADC_CONTR=ADC_POWER |ADC_SPEEDLL | ADC_START | ch;
}
```

# ADC应用实现4
# --具体实现过程

```
void main()
{       unsigned int i;
        SCON=0x5A;                          //串口1为8位可变波特率模式
        T2L=65536-OSC/4/BAUD;               //写定时器2低8位寄存器T2L
        T2H=(65536-OSC/4/BAUD)>>8;          //写定时器2高8位寄存器T2H
        AUXR=0x14;                          //定时器2不分频，启动定时器2
        AUXR|=0x01;                         //选择定时器2为串口1的波特率发生器
        P1ASF=0xFF;                         //P1端口作为模拟输入
        ADC_RES=0;                          //清ADC_RES寄存器
        ADC_CONTR=ADC_POWER|ADC_SPEEDLL | ADC_START | ch;
        for(i=0;i<10000;i++);                //延迟
        IE=0xA0;                            //CPU允许响应中断请求，使能ADC中断
        while(1);                           //无限循环
}
```

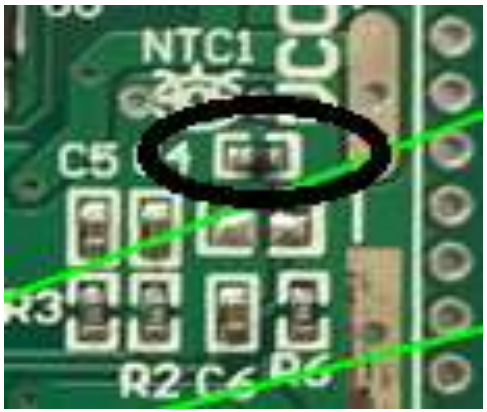
# ADC应用实现4

下载和分析设计的步骤主要包括：

- 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。
- 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：9600。
  - 校验位：无校验。
  - 停止位：1位。
- 单击打开串口按钮。

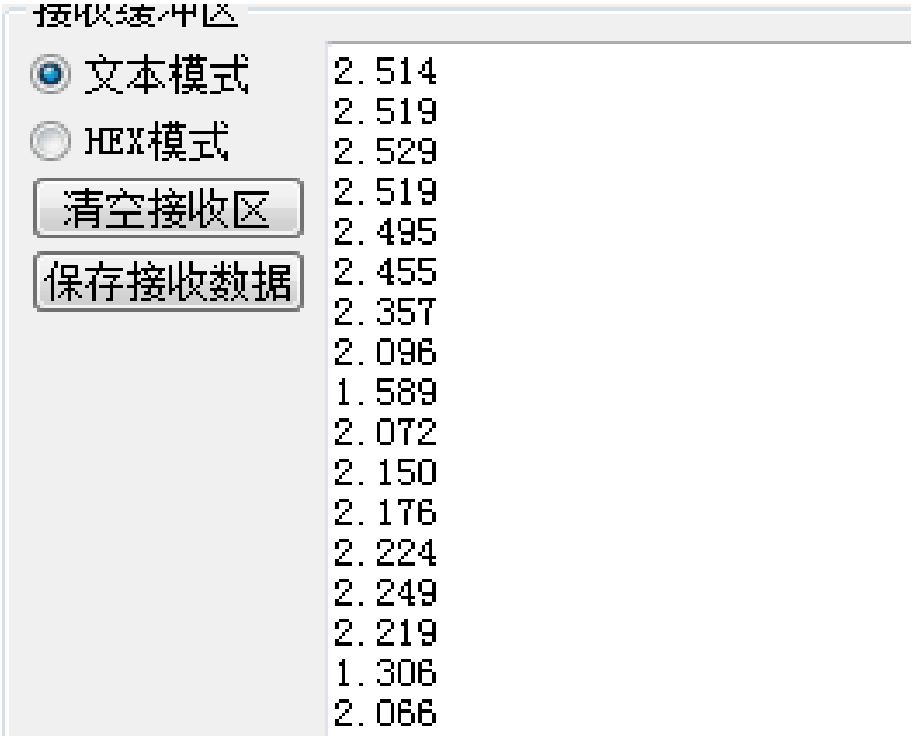
# ADC应用实现4

- 用电热吹风接近STC学习板上的热敏电阻，黑圈的位置。



# ADC应用实现4

- 观察串口的输出结果。
  - 当电烙铁瞬间接触热敏电阻时，其电压可以降低到1.306V。



# ADC应用实现4

- 将单片机的供电电压通过J12插座上的插针和导线连接到到P1.3插孔上，观察串口输出结果。
  - 从图中可以看出，单片机的供电电压非常稳定，值的变化范围为0.01V，即：10mV。