第 2-11 讲：模数转换 ADC

1. 学习目的

1. 了解 ADC 的基本概念：分辨率、精度等。
2. 掌握 STC8A8K64D4 单片机 ADC 的配置、采样数据计算为实际电压值的方法。
3. 掌握 ADC 多通道采样。

2. ADC 基本概念

实际应用中，我们经常需要将模拟量转换为数字量供 CPU 处理，如电池电压检测、温度检测等等。对于 CPU 来说，他能处理的是数字量，所以，需要通过 A/D 转换(模数转换)将时间连续、幅值也连续的模拟量转换为时间离散、幅值也离散的数字信号，从而实现 CPU 对模拟信号的处理，这种能够实现 A/D 转换功能的电路称之为模数转换器（ADC：Analog-to-digital converter）。

ADC 的结构和实现原理有多种方式，常见的 ADC 的类型有积分型、逐次逼近型、并行比较型/串并行型、Σ-Δ 调制型等。STC8A8K64D4 集成的是逐次逼近型 ADC，他由一个比较器和 D/A 转换器通过逐次比较逻辑构成，从 MSB 开始,顺序地对每一位将输入电压与内置 D/A 转换器输出进行比较，经 n 次比较而输出数字值。其优点是速度较高、功耗低。

ADC 常用的技术参数有以下几点，这是学习 ADC 必须要掌握的。

1. 分辨率

ADC 分辨率是指输出数字量变化一个最低有效位(LSB)所需的输入模拟电压的变化量。ADC 的分辨率用位数表示，如 10 位的 ADC，分辨率为 2^{10}=1024。如果 ADC 的量程为（0~5）V，那么 ADC 即可“分辨”出（5/1024）V 的电压变化。

2. ADC 精度

ADC 的精度取决于量化误差及系统内其他误差的总和。这里要特别注意 ADC 分辨率和 ADC 精度的区别，“精度”是用来描述物理量的准确程度的，而“分辨率”是用来描述刻度划分的。对于一个给定的器件，他的分辨率是固定的，如 STC8A8K64D4，他的分辨率固定为 12 位的，但是他的精度不仅仅受器件本身的影响，还可能会受 PCB 布线、外界环境（温度、湿度、干扰等）的影响而变化。

3. 转换速度

转换速度是指完成一次从模拟转换到数字的 AD 转换所需的时间的倒数，STC8A8K64D4 单片机最快的速度可以达到 800K（每秒转换 80 万次）。

3. ADC 的应用流程

STC8A8K64D4 单片机内部集成了一个 12 位高速 A/D 转换器，共有 16 个通道，ADC 最快的转换速度可以达到 800K，并且 ADC 转换结果的数据格式可以配置为左对齐或右对齐格式，以方便用户程序进行读取和引用。ADC 应用步骤如下图所示。

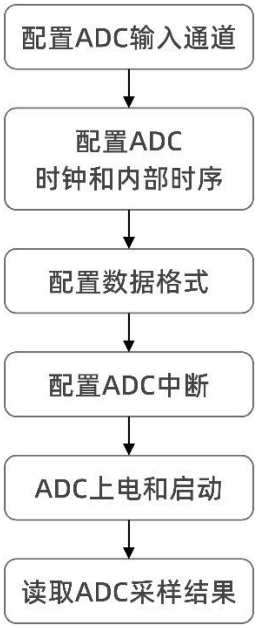


图 1：ADC 应用步骤

3.1. 配置 ADC 通道

使用 STC8A8K64D4 单片机 ADC 时，需要选择使用的 ADC 通道，STC8A8K64D4 单片机 ADC 的 16 个通道如下表所示。这里要注意一下，ADC 的通道 15 是没有输入引脚的，他只能用于检测内部参考信号源。

表 1：STC8A8K64D4 单片机 ADC 引脚分配

ADC 通道	**对应 IO 口**	**功能描述**
ADC0	P1.0	ADC 模拟通道输入 0
ADC1	P1.1	ADC 模拟通道输入 1
ADC2	P1.2	ADC 模拟通道输入 2
ADC3	P1.3	ADC 模拟通道输入 3
ADC4	P1.4	ADC 模拟通道输入 4
ADC5	P1.5	ADC 模拟通道输入 5
ADC6	P1.6	ADC 模拟通道输入 6
ADC7	P1.7	ADC 模拟通道输入 7
ADC8	P0.0	ADC 模拟通道输入 8
ADC9	P0.1	ADC 模拟通道输入 9
ADC10	P0.2	ADC 模拟通道输入 10
ADC11	P0.3	ADC 模拟通道输入 11
ADC12	P0.4	ADC 模拟通道输入 12
ADC13	P0.5	ADC 模拟通道输入 13
ADC14	P0.6	ADC 模拟通道输入 14
ADC15	ADC 的第 15 通道只能用于检测内部参考信号源， 参考信号源值出厂时校	

	准为 1.19V，由于制造误差以及测量误差，导致实际的内部参考信号源相比 1.19V，大约有± 1%的误差。如果用户需要知道每一颗芯片的准确内部参考信号源值，可外接精准参考信号源，然后利用 ADC 的第 15 通道进行测量标定。

ADC 输入通道由“ADC 控制寄存器（ADC_CONTR）”中的 ADC_CHS[3:0]位确定，如下所示。另外需要注意，当有 I/O 口被选择为 ADC 输入通道时，需要通过 PxM0/PxM1 寄存器将 I/O 口模式设置为高阻输入模式。

ADC 控制寄存器（ADC_CONTR）：

ADC通道选择位

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
ADC_CONTR	BCH	ADC_POWER	ADC_START	ADC_FLAG	ADC_EPWMT	ADC_CHS[3:0]			

- ADC_CHS[3:0] ADC 通道
  - 0000：P1.0/ADC0。
  - 0001：P1.1/ADC1。
  - 0010：P1.2/ADC2。
  - 0011：P1.3/ADC3。
  - 0100：P1.4/ADC4。
  - 0101：P1.5/ADC5。
  - 0110：P1.6/ADC6。
  - 0111：P1.7/ADC7。
  - 1000：P0.0/ADC8。
  - 1001：P0.1/ADC9。
  - 1010：P0.2/ADC10。
  - 1011：P0.3/ADC11。
  - 1100：P0.4/ADC12。
  - 1101：P0.5/ADC13。
  - 1110：P0.6/ADC14。
  - 1111：测试内部 1.19V。

3.2. 配置 ADC 时钟频率和内部时序

ADC 时钟频率和内部时序决定了 ADC 的转换速度，STC8A8K64D4 的 ADC 转换速度计算公式如下。可以看到 ADC 转换速度和时钟频率“SPEED”以及 ADC 内部时序控制寄存器中的“CSSETUP”、“CSHOLD”和“SMPDUTY”相关。

$$12\text{位ADC转换速度} = \frac{\text{MCU工作频率SYSclk}}{2\times(\text{SPEED[3:0]}+1)\times[(\text{CSSETUP}+1)+(\text{CSHOLD}+1)+(\text{SMPDUTY}+1)+12]}$$

对于 ADC 转换速度，要注意以下几点：

1) 12 位 ADC 的速度不能高于 800KHz。
2) SMPDUTY 的值不能小于 10，建议设置为 15。
3) CSSETUP 可使用上电默认值 0。
4) CHOLD 可使用上电默认值 1（ADCTIM 建议设置为 3FH）。

1. ADC 时钟频率设置

ADC 时钟频率由“ADC 配置寄存器（ADCCFG）”中的 SPEED[3:0]位确定，如下所示。

ADC 配置寄存器（ADCCFG）：

						ADC时钟频率设置位			
符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
ADCCFG	DEH	-	-	RESFMT	-	SPEED[3:0]			

■ SPEED[3:0]：设置 ADC 工作时钟频率 F_{ADC}＝ SYSclk/2/(SPEED+1)。

2. ADC 内部时序设置

ADC 时序由“ADC 时序控制寄存器（ADCTIM）”中的“CSSETUP”、“CSHOLD”和“SMPDUTY”确定，如下所示。

ADC 时序控制寄存器（ADCTIM）：

		ADC通道选择时间控制位	ADC通道选择保持时间控制位		ADC位模拟信号采样时间控制位				
符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
ADCTIM	FEA8H	CSSETUP	CSHOLD[1:0]		SMPDUTY[4:0]				

■ CSSETUP：ADC 通道选择时间控制 T_{setup}
- 0：1（默认值），即占用 1 个 ADC 工作时钟。
- 1：2，即占用 2 个 ADC 工作时钟。

■ CSHOLD[1:0]：ADC 通道选择保持时间控制 T_{hold}
- 00：1，即占用 1 个 ADC 工作时钟。
- 01：2（默认值），即占用 2 个 ADC 工作时钟。
- 10：3，即占用 3 个 ADC 工作时钟。
- 11：4，即占用 4 个 ADC 工作时钟。

■ SMPDUTY[4:0]：ADC 模拟信号采样时间控制 T_{duty}（注意：SMPDUTY 的值必须≥01010B），SMPDUTY 的值对应的占用 ADC 工作时钟数如下表所示。

SMPDUTY[4:0]	占用 ADC 工作时钟数
00000	1
00001	2
...	...
01010	**11（默认值）**
...	...
11110	31
11111	32

3.3. 配置 ADC 数据格式

为了方便用户在程序里面处理 ADC 采样数据，STC8A8K64D4 的 ADC 提供了两种数据格式：左对齐和右对齐，数据格式由“ADC 配置寄存器（ADCCFG）”中的“RESFMT”位设置，如下所示。

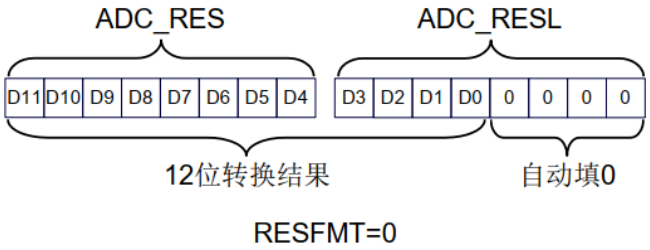
ADC 配置寄存器（ADCCFG）：

ADC结果格式控制位

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
ADCCFG	DEH	-	-	RESFMT	-	SPEED[3:0]			

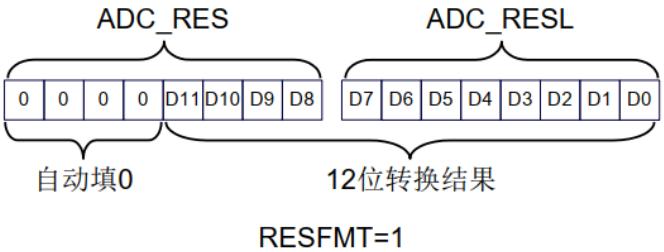
■ RESFMT：ADC 转换结果格式控制位

- 0：转换结果左对齐。ADC_RES 保存结果的高 8 位，ADC_RESL 保存结果的低 4 位，格式如下：



RESFMT=0

- 1：转换结果右对齐。ADC_RES 保存结果的高 4 位，ADC_RESL 保存结果的低 8 位，格式如下：



RESFMT=1

3.4. 配置 ADC 中断

ADC 中断的开启和关闭由中断使能寄存器 IE 的位 5（EADC）控制，如下图所示。另外注意：开启 ADC 中断的情况下，还需要开启总中断“EA=1”，ADC 中断才能起作用。

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IE	A8H	EA	ELVD	EADC	ES	ET1	EX1	ET0	EX0
		总中断允许位		ADC中断允许位					

- EADC：A/D 转换中断允许位。
  - 0：禁止 A/D 转换中断。
  - 1：允许 A/D 转换中断。

3.5. ADC 上电和启动

ADC 上电和启动由“ADC 控制寄存器（ADC_CONTR）”中“ADC_POWER”和“ADC_START”位控制，如下图所示。

ADC 控制寄存器（ADC_CONTR）：

		ADC上电	ADC启动						
符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
ADC_CONTR	BCH	ADC_POWER	ADC_START	ADC_FLAG	ADC_EPWMT	ADC_CHS[3:0]			

- ADC_POWER：ADC 电源控制位
  - 0：关闭 ADC 电源。
  - 1：打开 ADC 电源。

注意事项：

- ADC 在启动之前需要先上电，上电后需等待约 1ms 的时间，让 MCU 内部的 ADC 电源趋于稳定后再启动 ADC 工作。
- 适当加长对外部信号的采样时间，就是对 ADC 内部采样保持电容的充电或放电时间，时间够，内部才能和外部电势相等。
- 通过 ADC 电源控制位，我们可以在单片机进入空闲模式和掉电模式前将 ADC 电源关闭，当需要 ADC 工作的时候再打开电源，以降低功耗。

- ADC_START： ADC 转换启动控制位。写入 1 后开始 ADC 转换，转换完成后硬件自动将此位清零。
  - 0：无影响。即使 ADC 已经开始转换工作，写 0 也不会停止 A/D 转换。
  - 1：开始 ADC 转换，转换完成后硬件自动将此位清零。

3.6. 读取 ADC 采样结果

当 A/D 转换完成后，转换结果会自动保存到 ADC 转换结果寄存器 ADC_RES 和 ADC_RESL 中。保存时会按照“ADC 配置寄存器（ADCCFG）”中的“RESFMT”位设置的数据格式保存，因此，我们在读取数据时，也要按照设置的格式获取数据。下面的代码是 ADC 数据格式为右对齐时的读取示例。

代码清单：读取 ADC 采样结果

```
1.  u8 adc_H,adc_L;    //分别保存读取的 ADC 结果的高字节和低字节数据
2.  u16 adc_value;     //保存 ADC 结果转换为 16 位的数据
3.
```

```
adc_H = ADC_RES & 0x0F;    //从 ADC_RES 寄存器中读取 ADC 结果的高字节数据，只有低 4 位有效
adc_L = ADC_RESL;          //从 ADC_RESL 寄存器中读取 ADC 结果的低字节数据

adc_value = (u16)((adc_H<<8)+adc_L);  //将 ADC 结果转换位 16 位数据，以便于计算电压
```

读取 ADC 采样值后，即可计算电压值。开发板使用的是外部 2.5V 参考电压，采样值转换为电压的公式如下。

$$电压值(V) = \frac{2.5 \times 采样值}{4096} V$$

4. 软件设计

4.1. ADC 采样电位器电压（查询方式）

✧ 注：本节的实验是在“实验 2-6-1：串口 1 数据收发实验”的基础上修改，本节对应的实验源码是：“实验 2-11-1：ADC 采样电位器电压（查询方式）”。

4.1.1. 实验内容

使用 ADC 模拟通道输入 14（即引脚 P0.6）采样电位器抽头电压，程序中每 500 毫秒执行一次电压采样，采样结果计算为电压值后通过串口输出。

本例使用查询方式执行 ADC 采样，所谓查询方式即不开启中断，当 ADC 启动采样后，程序中反复查询 ADC 采样完成标志位，若该位置位，表示当前 ADC 采样完成，此时，可以读取 ADC 采样结果。

4.1.2. 代码编写

1. 新建一个名称为“adc.c”的文件及其头文件“adc.h”并保存到工程的“Source”文件夹，并将“adc.c”加入到 Keil 工程中的“SOURCE”组。
2. 引用头文件

因为在“main.c”文件中使用了“adc.c”文件中的函数，所以需要引用下面的头文件“adc.h”。

代码清单：引用头文件

```
//引用 ADC 的头文件
#include    " adc.h"
```

3. ADC 初始化

ADC 初始化通常包含以下部分内容：

1) 通道选择：本例中使用 ADC 模拟通道输入 14 采样电位器抽头电压，ADC 模拟通道输入 14 对应的 I/O 是 P0.6，因此，需要选择 P0.6 作为模拟功能 ADC 使用。

2) 时钟频率和内部时序：

- ADC 通道选择时间设置为：占用 1 个 ADC 工作时钟。
- ADC 通道选择保持时间设置为：占用 2 个 ADC 工作时钟。
- ADC 模拟信号采样时间设置为：占用 32 个 ADC 工作时钟。
- ADC 工作时钟频率设置为：SYSclk/2/16。

3) ADC 采样结果数据格式设置为：右对齐，即 ADC_RES 存高 4 位，ADC_RESL 存低 8 位。
4) ADC 中断：本例使用查询方式，因此不开启 ADC 中断。
5) ADC 上电：ADC 上电可以在 ADC 初始化中完成，也可以在需要的时候再执行，执行 ADC 上电操作后，切记，需要延时不小于 1ms 的时间以保证 ADC 电源稳定。

为了方便程序调用，我们将 ADC 初始化封装为名称为“adc_config()”的函数，该函数代码如下：

代码清单：ADC 初始化函数

```
1.  /*******************************************************************************
2.  功能描述：初始化 ADC
3.  参    数：无
4.  返 回 值：无
5.  *******************************************************************************/
6.  void adc_config(void)
7.  {
8.      P0M1 |= 0x40;P0M0 &= ~0x40; //设置 P0.6 为输入
9.      ADC_CONTR |= 0x0E;          //选择 P0.6 作为模拟功能 ADC 使用
10.     ADC_CONTR &= 0xFE;          //选择 P0.6 作为模拟功能 ADC 使用
11.
12.     //内部时序配置
13.     P_SW2 |= 0x80;              //将 EAXFR 位置 1，允许访问扩展 RAM 区特殊功能寄存器(XFR)
14.     ADCTIM = 0x3f;              //设置 ADC 内部时序
15.     P_SW2 &= 0x7f;              //将 EAXFR 位置 0，禁止访问 XFR
16.
17.     ADCCFG |= 0x0F;             //ADC 工作时钟频率设置为 SYSclk/2/16
18.     ADC_CONTR &= 0xDF;          //清 AD 转换完成标志
19.
20.     EADC = 0;           //关闭 ADC 中断
21.     ADCCFG|=0x20;       //ADC 转换结果格式设置为右对齐，即 ADC_RES 存高 4 位，ADC_RESL 存低 8 位
22.     ADC_CONTR|=0x80;    //ADC 上电
23.     delay_ms(2);        //ADC 上电后，延时不小于 1ms 以保证 ADC 供电稳定
24. }
```

4. 启动 ADC 转换

ADC 初始化并上电完成后，即可启动 ADC 转换。ADC 每次转换都需要启动一次，也就是每启动一次 ADC 转换，ADC 就会执行一次转换，下次转换前需要再次启动。我们编写的启动 ADC 转换的函数代码如下。

代码清单：ADC 启动函数

```
1.  /*******************************************************************************
2.   * 描  述 : 启动 AD 转换
```

```
3.    * 入  参 : 无
4.    * 返回值 : 无
5.    ******************************************************************************/
6.   void adc_start(void)
7.   {
8.      ADC_CONTR|=0x40;            //启动 AD 转换
9.   }
```

5. 查询 ADC 转换是否完成

本例中，我们使用的查询的方式，因此在启动 ADC 采样之后，需要查询 ADC_CONTR 寄存器中的 ADC_FLAG 位（ADC 转换结束标志位），以此判断 ADC 是否转换完成，代码清单如下。

代码清单：查询 ADC 转换是否完成

```
1.   /******************************************************************************
2.    * 描  述 : 查询 ADC 转换是否完成
3.    * 入  参 : 无
4.    * 返回值 : ADC 转换完成-返回 true，否则，返回 false
5.    ******************************************************************************/
6.   bool adc_completed(void)
7.   {
8.      if((ADC_CONTR & 0x20) == 0x20)return true;
9.      else return false;
10.  }
```

6. 读取 ADC 采样值

查询到 ADC 转换完成后，即可读取 ADC 转换结果，这里需要注意读取结果时需要结合 ADC 初始化时配置的采样结果数据格式（左对齐或右对齐），从采样结果中读出正确的采样数据。

代码清单：读取 ADC 采样值

```
1.   /******************************************************************************
2.    * 描  述 : 读取 ADC 采样值
3.    * 入  参 : 无
4.    * 返回值 : 读取的 ADC 采样值
5.    ******************************************************************************/
6.   u16 get_adc_value(void)
7.   {
8.      u8 adc_H,adc_L;
9.
10.     ADC_CONTR &= 0xDF;          //将 ADC_FLAG 清 0
11.     ADC_CONTR &= 0xBF;          //关闭 AD 转换，ADC_START=0
12.     adc_H = ADC_RES & 0x0F;     //读取计数值
13.     adc_L = ADC_RESL;           //12 位 AD 结果的高 4 位放 ADC_RES 的低 4 位，低 8 位在 ADC_RESL
14.     ADC_CONTR|=0x40;            //启动 AD 转换，ADC_START=1
```

```
    return (adc_H<<8)+adc_L;  //返回读取的计数值
}
```

7. 主函数

主函数中调用 ADC 初始化完成 ADC 的初始化，之后在主循环中每 500ms 启动一次 ADC 转换，启动后一直查询 ADC 转换结束标志位 ADC_FLAG 的值，直到 ADC_FLAG 置位，即 ADC 转换完成。此后，读取 ADC 采样值并将其计算为电压值通过串口输出。

代码清单：主函数

```
/*******************************************************************************
功能描述：主函数
入口参数：无
返回值：int 类型
 *******************************************************************************/
int main(void)
{
    u16 adc_value;  //存放 ADC 采样值
    float voltage;  //存放 ADC 采样值计算后的电压值

    P2M1 &= 0xBF;   P2M0 &= 0xBF;     //设置 P2.6 为准双向口（LED1）
    P3M1 &= 0xFE;   P3M0 &= 0xFE;     //设置 P3.0 为准双向口(UART RxD)
    P3M1 &= 0xFD;   P3M0 |= 0x02;     //设置 P3.1 为推挽输出(UART TxD)

    uart1_init();       //串口 1 初始化
    adc_config();       //初始化 ADC

    while(1)
    {
       adc_start();         //启动 ADC 转换
       while(adc_completed() == false) ;    //等待 ADC 转换完成
       adc_value = get_adc_value();         //读取 ADC 采样值
       voltage   = (2.5*adc_value)/4096;    //将 ADC 采样值转换为电压（单位 V）
       printf("voltage:  %.2fV\r\n",voltage);//串口打印 ADC 采样电压
       delay_ms(500); //延时 500ms，方便在串口调试助手中观察实验数据
    }
}
```

4.1.3. 硬件连接

本实验需要使用 ADC 模拟通道输入 14（即引脚 P0.6）采样电位器抽头电压，因此需要将 P06 引脚和电位器电路通过跳线帽连接，如下图所示。

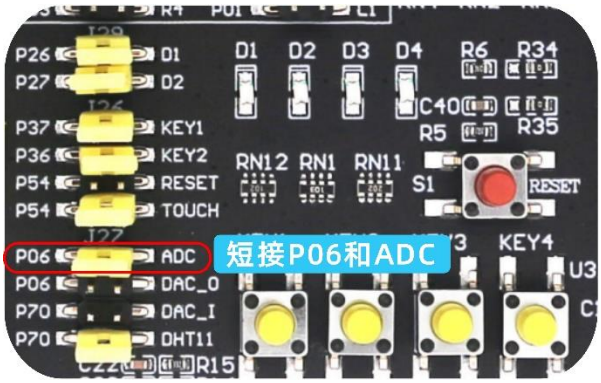


图 2：跳线帽短接

4.1.4. 实验步骤

1) 解压“…\第 3 部分：配套例程源码”目录下的压缩文件“实验 2-11-1：ADC 采样电位器电压（查询方式）”，将解压后得到的文件夹拷贝到合适的目录，如“D\STC8”（这样做的目的是为了防止中文路径或者工程存放的路径过深导致打开工程出现问题）。
2) 双击“…\adc\project”目录下的工程文件“adc.uvproj”。
3) 点击编译按钮编译工程，编译成功后生成的 HEX 文件“adc.hex”位于工程的“…\adc\Project\Object”目录下。
4) 打开 STC-ISP 软件下载程序，下载使用内部 IRC 时钟，IRC 频率选择：24MHz。
5) 电脑上打开串口调试助手，选择开发板对应的串口号，将波特率设置为 9600bps。程序运行后，在串口接收窗口可以看到开发板上报的 ADC 采样的电压值，如下图所示。

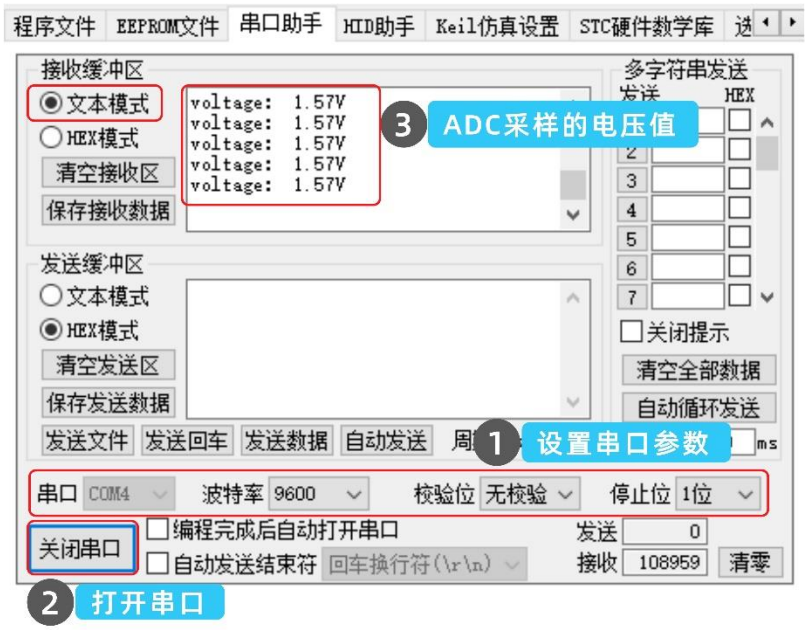


图 3：串口调试助手收发数据

6) 旋转电位器改变电位器抽头电压，观察串口接收的数据，可以看到电压值的变化。

4.2. ADC 采样电位器电压（中断方式）

✧ 注：本节的实验是在“实验 2-11-1：ADC 采样电位器电压（查询方式）”的基础上修改，本节对应的实验源码是：“实验 2-11-2：ADC 采样电位器电压（中断方式）”。

4.2.1. 实验内容

本实验所实现的功能和“实验 13-1”完全一样，不同的地方在于本例中 ADC 采样使用了中断的方式。即程序中开启 ADC 中断，当启动 ADC 采样后，无需查询 ADC 转换完成标志位，ADC 转换完成后会进入 ADC 中断服务函数，此时，我们即可读取 ADC 采样结果。

由此可见，中断方式的效率要高于查询方式，因为 CPU 在启动 ADC 采样后，可以继续去做其他事情，而无需像查询方式一样去反复查询 ADC 采样是否完成。当 ADC 采样完成后，会触发中断，以此通知 CPU，ADC 采样已经完成。

4.2.2. 代码编写

本例中使用了 ADC 中断，因此，ADC 初始化函数中需要开启 ADC 中断，代码清单如下所示。

代码清单：ADC 初始化函数

```
1.  /*************************************************************************************************
2.  功能描述：初始化 ADC
3.  参    数：无
4.  返 回 值：无
5.  *************************************************************************************************/
6.  void adc_config(void)
7.  {
8.      //省略了无关的代码
9.      EADC = 1;           //使能 ADC 中断，注意：开启 ADC 中断的情况下，还需要开启总中断“EA=1”，中断才能起作用
10.     //省略了无关的代码
11. }
```

当 ADC 采样完成后，会进入中断服务函数，我们在 ADC 中断服务函数中读取 ADC 采样值即可。读出的采样值计算为电压值后，同样使用串口输出，代码清单如下。

代码清单：ADC 中断服务函数

```
1.  /*************************************************************************************
2.   * 描   述 : ADC 中断服务函数
3.   * 入   参 : 无
4.   * 返回值 : 无
5.   *************************************************************************************/
6.  void adc_isr() interrupt 5
7.  {
8.      u8 adc_H,adc_L;
9.      u16 adc_value;  //存放 ADC 采样值
10.     float voltage;  //存放 ADC 采样值计算后的电压值
```

```
11.
12.     ADC_CONTR &= ~0x20;                         //清零 ADC 中断标志
13.     adc_H = ADC_RES & 0x0F;                     //读取计数值
14.     adc_L = ADC_RESL;           //12 位 AD 结果的高 4 位放 ADC_RES 的低 4 位，低 8 位在 ADC_RESL
15.     adc_value = (adc_H<<8)+adc_L;               //读取 ADC 采样值
16.     voltage   = (2.5*adc_value)/4096;           //将 ADC 采样值转换为电压（单位 V）
17.     printf("voltage:  %.1fV\r\n",voltage);    //串口打印 ADC 采样电压
18. }
```

主函数中开启总中断，之后，在主循环中每 500ms 启动一次 ADC 采样，即每 500ms 采样一次电位器抽头电压，代码清单如下。

代码清单：主函数

```
1.  /*************************************************************************
2.  功能描述：主函数
3.  入口参数：无
4.  返回值：int 类型
5.   *************************************************************************/
6.  int main(void)
7.  {
8.     //省略了无关的代码
9.     adc_config();       //初始化 ADC
10.    EA = 1;             //使能总中断
11.
12.    while(1)
13.    {
14.       adc_start();    //启动 ADC 转换
15.       delay_ms(500); //延时 500ms，方便在串口调试助手中观察实验数据
16.    }
17. }
```

4.2.3. 硬件连接

同“实验 2-11-1：ADC 采样电位器电压（查询方式）”。

4.2.4. 实验步骤

同“实验 2-11-1：ADC 采样电位器电压（查询方式）”。

4.3. ADC 多通道采样

STC8A8K64D4 单片机的 ADC 有 16 个通道，但是 ADC 转换器只有一个，因此，ADC 模块每次只能对一个通道进行转换，而不能同时对多个通道进行转换。但是在实际应用时，我们经常会用到多个 ADC 通道采样电压值，这时，我们可以使用轮询的方式对各个 ADC

通道进行采样，具体的实现流程如下图所示，即对当前 ADC 通道采样完成后，切换到下一个 ADC 通道，之后启动 ADC 转换并在转换完成后读取采样结果，如此轮询对使用的各个 ADC 通道采样，从而实现 ADC 多通道采样。

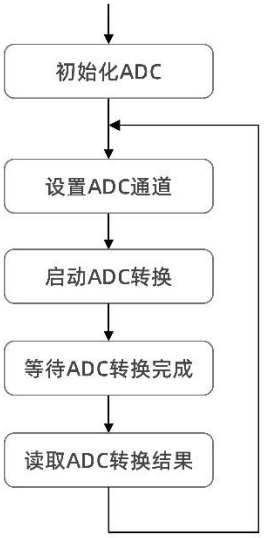


图 4：ADC 多通道采样流程

✧ 注：本节的实验是在“实验 2-11-1：ADC 采样电位器电压（查询方式）”的基础上修改，本节对应的实验源码是：“实验 2-11-3：ADC 多通道采样（2 个通道）”。

4.3.1. 实验内容

本实验对 2 个 ADC 通道进行采样：ADC 模拟通道输入 2（P1.2）和 ADC 模拟通道输入 14（P0.6）。由于开发板上只有一个电位器，为了实验方便，两个通道做如下处理。

- ADC 通道 14 对应的 P0.6：连接到电位器抽头，在实验时，可以通过旋转电位器改变电位器抽头电压，从而可以观察到 ADC 采样的电压值的变化。
- ADC 通道 2 对应的 P1.2：用杜邦线连接到 GND，ADC 采样的电压值应接近于 0V。

程序中使用轮询的方式对 2 个 ADC 通道进行采样，采样的结果计算为电压值后通过串口输出，在串口调试助手里面即可观察到 ADC 采样的电压值。

4.3.2. 代码编写

本例中使用了 2 个 ADC 通道，因此需要修改 ADC 初始化函数，将两个 ADC 通道的 I/O 设置为高阻输入模式。另外，为了方便轮询 ADC 通道，在 ADC 初始化函数中不设置 ADC 通道，而是编写了一个专门用于设置 ADC 通道的函数“adc_set_channel()”用于设置 ADC 通道。修改后的 ADC 初始化函数代码清单如下。

代码清单：ADC 初始化函数

```
1.  /*********************************************************************************
2.  功能描述：初始化 ADC
3.  参    数：无
4.  返 回 值：无
```

```
5.  ******************************************************************************/
6.  void adc_config(void)
7.  {
8.      P0M1 |= 0x40;P0M0 &= ~0x40; //设置 P0.6 为高阻输入
9.      P1M1 |= 0x04;P1M0 &= ~0x04; //设置 P1.2 为高阻输入
10.
11.     //内部时序配置
12.     P_SW2 |= 0x80;              //将 EAXFR 位置 1，允许访问扩展 RAM 区特殊功能寄存器(XFR)
13.     ADCTIM = 0x3f;              //设置 ADC 内部时序
14.     P_SW2 &= 0x7f;              //将 EAXFR 位置 0，禁止访问 XFR
15.
16.     ADCCFG |= 0x0F;             //ADC 工作时钟频率设置为 SYSclk/2/16
17.     ADC_CONTR &= 0xDF;          //清 AD 转换完成标志
18.
19.     EADC = 0;           //关闭 ADC 中断
20.     ADCCFG|=0x20;       //ADC 转换结果格式设置为右对齐，即 ADC_RES 存高 4 位，ADC_RESL 存低 8 位
21.     ADC_CONTR|=0x80;    //ADC 上电
22.     delay_ms(2);        //ADC 上电后，延时不小于 1ms 以保证 ADC 供电稳定
23. }
```

用于设置 ADC 通道的函数“adc_set_channel()”的代码清单如下。

代码清单：ADC 通道设置函数

```
1.  /******************************************************************************
2.  功能描述：设置 ADC 通道。ADC 共有 16 个通道，对应的通道号为 0~15，其中，通道 15 是没有输入引脚的，
3.            他只能用于检测内部参考信号源
4.  参     数：ch[in]：adc 通道，取值范围为 0~15
5.  返 回 值：无
6.  ******************************************************************************/
7.  void adc_set_channel(u8 ch)
8.  {
9.      ADC_CONTR &= ~0x0F;       //先清零 ADC 通道选择位
10.     ADC_CONTR |= ch & 0x0F;   //再设置 ADC 通道
11. }
```

使用函数“adc_set_channel()”选择 ADC 通道后，即可执行 ADC 采样。这里，我们编写了一个阻塞式的 ADC 采样函数，将通道切换、启动 ADC 采样和数据读取封装在该函数中。代码清单如下，首先设置 ADC 通道，接着启动 ADC 采样并一直查询 ADC 转换完成标志直到 ADC 转换完成标志置位，之后读出采样结果将之计算为电压并通过串口输出，

代码清单：ADC 采样函数

```
1.  /******************************************************************************
2.   * 描   述 : 对给定的 ADC 通道执行一次 ADC 转换，这里使用的是阻塞的方式，即启动转换后一直等待直到转换完成
3.   * 入   参 : ch[in]：adc 通道，取值范围为 0~15
4.   * 返回值 : 无
5.   ******************************************************************************/
```

```
void adc_block_sample(u8 ch)
{
    u16 adc_value;  //存放 ADC 采样值
    float voltage;  //存放 ADC 采样值计算后的电压值

    adc_set_channel(ch);                    //设置 ADC 通道
    adc_start();                            //启动 ADC 转换
    while(adc_completed() == false);        //等待 ADC 转换完成
    adc_value = get_adc_value();            //读取 ADC 采样值
    voltage   = (2.5*adc_value)/4096;       //将 ADC 采样值转换为电压（单位 V）
    printf("voltage:  %.2fV\r\n",voltage);//串口打印 ADC 采样电压
}
```

主函数中，先调用 ADC 初始化函数“adc_config()”初始化 ADC，之后在主循环里面轮询对两个 ADC 通道进行采样，每次采样后演示 500ms，这样做是为了方便观察实验数据。

对于多通道采样，需要特别注意的是：如果采样的频率比较快，则需要对 ADC 内部进行放电，即启动采样时先将 ADC 通道对应的 I/O 设置为开漏模式进行放电，接着再将其配置为高阻输入执行 ADC 采样。

代码清单：主函数

```
/*************************************************************************
功能描述：主函数
入口参数：无
返回值：int 类型
 *************************************************************************/
int main(void)
{
    P2M1 &= 0xBF;   P2M0 &= 0xBF;     //设置 P2.6 为准双向口（LED1）
    P3M1 &= 0xFE;   P3M0 &= 0xFE;     //设置 P3.0 为准双向口(UART RxD)
    P3M1 &= 0xFD;   P3M0 |= 0x02;     //设置 P3.1 为推挽输出(UART TxD)

    uart1_init();      //串口 1 初始化
    adc_config();      //初始化 ADC

    while(1)
    {
       /*--如果 ADC 采样的速度比较快，加上下面的两条语句，可以提高采样精度--*/
       //P0M1 |= 0x40;P0M0 |= 0x40;  //设置 P0.6 为开漏输入，目的是给 ADC 内部放电
       //P0M1 |= 0x40;P0M0 &= ~0x40; //设置 P0.6 为高阻输入
       adc_block_sample(12);          //对 ADC 通道 12 采样一次
       delay_ms(500);//延时 500ms，方便在串口调试助手中观察实验数据

       /*--如果 ADC 采样的速度比较快，加上下面的两条语句，可以提高采样精度--*/
       //P1M1 |= 0x04;P1M0 |= 0x04;  //设置 P1.2 为开漏输入，目的是给 ADC 内部放电
```

```
        //P1M1 |= 0x04;P1M0 &= ~0x04; //设置 P1.2 为高阻输入
        adc_block_sample(2);          //对 ADC 通道 2 采样一次
        delay_ms(500); //延时 500ms，方便在串口调试助手中观察实验数据
    }
}
```

4.3.3. 硬件连接

本实验需要使用 ADC 模拟输入通道 14（即引脚 P0.6）采样电位器抽头电压和 ADC 模拟输入通道 2（P1.2）采样 GND 电压，因此需要将 J27 端子的 P06 和电位器电路（ADC）通过跳线帽连接，并用杜邦线将 J9 端子的 P12 连接到 J18 端子的 GND，如下图所示

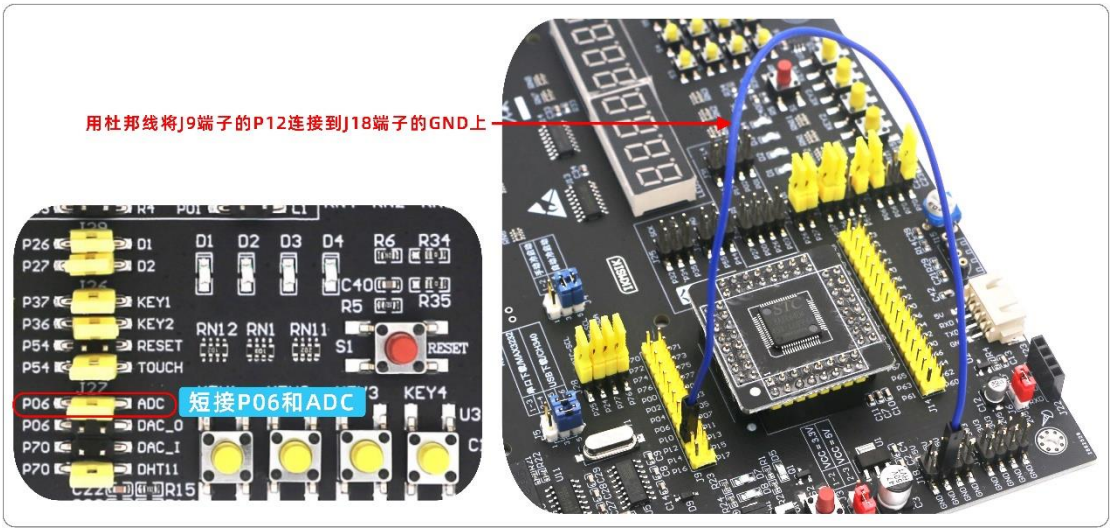


图 5：硬件连接

4.3.4. 实验步骤

1) 解压“…\第 3 部分：配套例程源码”目录下的压缩文件“实验 2-11-3：ADC 多通道采样（2 个通道）”，将解压后得到的文件夹拷贝到合适的目录，如“D\STC8”（这样做的目的是为了防止中文路径或者工程存放的路径过深导致打开工程出现问题）。
2) 双击“…\adc_muti_channel\project”目录下的工程文件“adc.uvproj”。
3) 点击编译按钮编译工程，编译成功后生成的 HEX 文件“adc.hex”位于工程的“…\adc\Project\Object”目录下。
4) 打开 STC-ISP 软件下载程序，下载使用内部 IRC 时钟，IRC 频率选择：24MHz。
5) 电脑上打开串口调试助手，选择开发板对应的串口号，将波特率设置为 9600bps。程序运行后，在串口接收窗口可以看到开发板上报的 ADC 采样的电压值，如下图所示。

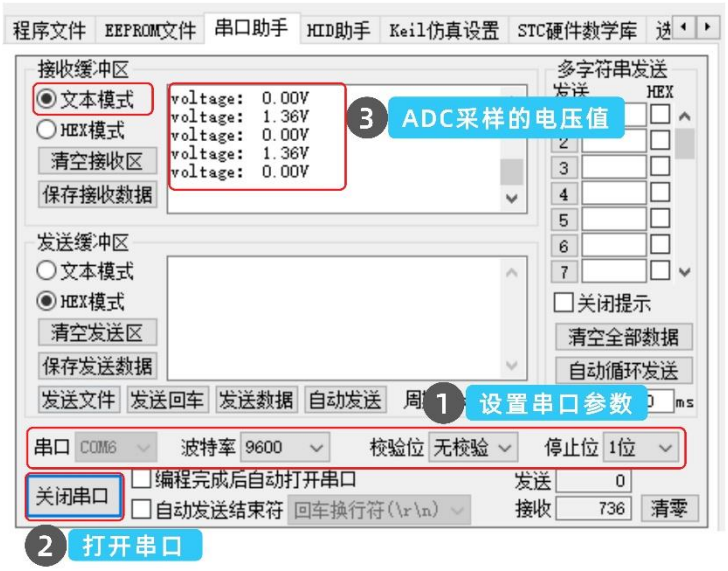


图 6：串口调试助手收发数据

6) 旋转电位器改变电位器抽头电压，观察串口接收的数据，可以看到电压值的变化（其中一路电压的值一直为 0，是因为实验时将通道 2 的 I/O P1.2 用杜邦线连接到了 GND）。