第5章 模数和数模转换器

A/D转换器简写为ADC(Analog to Digital Converter),是一种能把模拟量转换为相应的数字量的电子器件。D/A转换器简写为DAC(Digital To Analog Converter),与A/D转换器相反,它能把数字量转换为相应的模拟量。在单片机控制系统中,经常需要用到A/D和D/A转换器。它们的功能及其在实时控制系统中的地位如图5-1所示。由图可见,被控实体的过程信号可以是电量(如电流、电压等),也可以是非电量(如温度、压力、传速等),其数值是随时间连续变化的。各种模拟量都可以通过变送器或传感器转换为相应的数字量送给单片机。单片机对过程信息进行运算和处理,把过程信息进行当地显示或打印等,同时将处理后的数字量送给D/A转换器,转换为相应的模拟量去对被控系统进行控制和调整,使系统处于最佳工作状态。

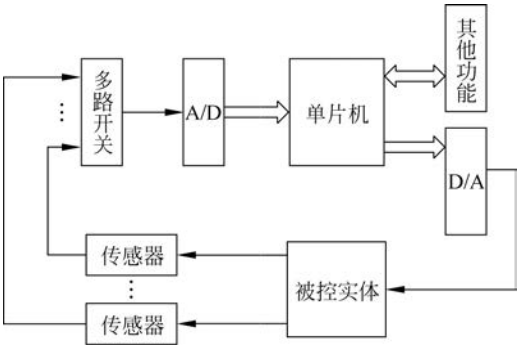


图5-1 单片机实时控制系统示意图

上述分析表明:A/D转换器在单片机控制系统中主要用于数据采集,向单片机提供被控对象的各种实时参数,以便单片机对被控对象进行监视和控制决策;D/A转换器用于模拟控制,通过机械或电气手段对被控对象进行调整和控制。因此,A/D、D/A转换器是架设在单片机和被控实体之间的桥梁,在单片机控制系统占有极其重要的地位。

C8051F020是混合信号型单片机,在片内集成了模数(A/D)和数模(D/A)转换电路。下面分别进行叙述。

5.1 模数转换器

A/D是将模拟量转换为数字量的器件。模拟量可以是电压、电流等电信号,也可以是声、光、压力、湿度、温度等随时间连续变化的非电的物理量。非电的模拟量可通过合适的传

感器(如光电传感器、压力传感器、温度传感器)转换为电信号。

C8051F020 片内包含一个 9 通道的 12 位的模数转换器 ADC0 和 8 通道 8 位的模数转换器 ADC1。

5.1.1 模数转换原理和性能指标

1. 转换原理

A/D 转换器的种类很多,根据转换原理可以分计数式、并行式、双积分式、逐次逼近式等。计数式 A/D 转换器结构简单,但转换速度也很慢,所以很少采用。并行 A/D 转换器的转换速度最快,但因结构复杂而造价较高,只用于那些转换速度极高的场合。双积分式 A/D 转换器抗干扰能力强,转换精度也很高,但速度不够理想,常用于数字式测量仪表中。计算机中广泛采用逐次逼近式 A/D 转换器作为接口电路,它的结构不太复杂,转换速度也较高。下面仅对逐次逼近式和双积分式 A/D 转换器的转换原理进行简单介绍。

1) 逐次逼近式 A/D 转换器

逐次逼近式 A/D 也称逐次比较法 A/D。它由结果寄存器、D/A、比较器和置位控制逻辑等部件组成,原理框图如图 5-2 所示。

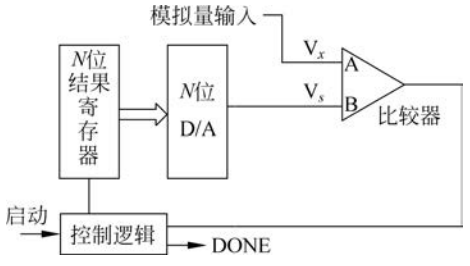


图 5-2 逐次逼近式 A/D 原理框图

这种 A/D 采用对分搜索法逐次比较、逐步逼近的原理来转换,整个转换过程是个“试探”过程。控制逻辑先置 1 结果寄存器最高位 D_{n-1} 然后经 D/A 转换得到一个占整个量程一半的模拟电压 V_s,比较器将此 V_s 和模拟输入电压 V_x 比较,若 $V_x > V_s$ 则保留此位 D_{n-1}(为 1),否则清 0D_{n-1} 位。然后控制逻辑置 1 结果寄存器次高位 D_{n-2},连同 D_{n-1} 一起送 D/A 转换,得到的 V_s 再和 V_x 比较,以决定 D_{n-2} 位保留为 1 还是清 0,以此类推。最后,控制逻辑置 1 结果寄存器最低位 D_0,然后将 D_{n-1}、D_{n-2}、……、D_0 一起送 D/A 转换。转换得到的结果 V_s 和 V_x 比较,决定 D_0 位保留为 1 还是清 0。

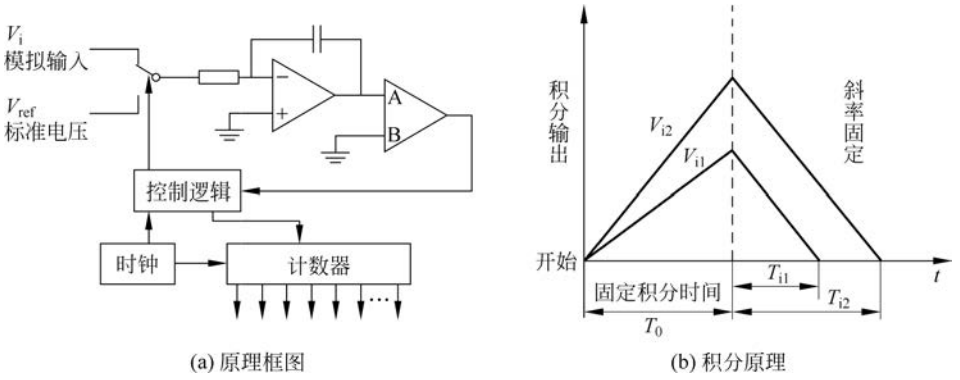
至此,结果寄存器的状态便是与输入的模拟量 V_x 对应的数字量。

2) 双积分式的 A/D 转换器

双积分式也称二重积分式,其实质是测量和比较两个积分的时间(它的工作原理见图 5-3):一个是对模拟输入电压积分的时间 T_0,此时间往往是固定的;另一个是以充电后的电压为初值,对参考电源 V_{ref} 反向积分,积分电容被放电至零所需的时间 T_i(V_{ref} 与 V_i 符号相反)。反向积分的斜率是固定的。模拟输入电压 V_i 与参考电压 V_{ref} 之比,等于上述两个时间之比。由于 V_{ref}、T_0 固定,而放电时间 T_i 可以测出,因而可计算出模拟输入电压的大小。

由于 T_0、V_{ref} 为已知的固定常数,因此反向积分时间 T_i 与输入模拟电压 V_i 在 T_0 时间内的平均值成正比。输入电压 V_i 愈高,V_A 愈大,T_i 就愈长。在 T_i 开始时刻,控制逻辑同时打开计数器的控制门开始计数,直到积分器恢复到零电平时,计数停止。则计数器所计出的数字即正比于输入电压 V_i 在 T_0 时间内的平均值,于是完成了一次 A/D 转换。

由于双积分型 A/D 转换是测量输入电压 V_i 在 T_0 时间内的平均值,因此对常态干扰(串模干扰)有很强的抑制作用,尤其对正负波形对称的干扰信号,抑制效果更好。



(a) 原理框图　　(b) 积分原理

图 5-3　双积分式 A/D 转换器工作原理图

双积分型的 A/D 转换器具有电路简单、抗干扰能力强、精度高等优点。但转换速度比较慢，常用的 A/D 转换芯片的转换时间为毫秒级。例如 12 位的积分型 A/D 芯片 ADCET12BC，其转换时间为 1ms。因此适用于模拟信号变化缓慢，采样速率要求较低，而对精度要求较高，或现场干扰较严重的场合。例如常在数字电压表中采用该芯片进行 A/D 采样。

2. 性能指标

衡量 A/D 性能的主要参数是：

1) 分辨率

分辨率(Resolution)是指输出的数字量变化一个相邻的值所对应的输入模拟量的变化值；取决于输出数字量的二进制位数。一个 n 位的 A/D 转换器所能分辨的最小输入模拟增量定义为满量程值的 2^{-n} 倍。例如，满量程为 10V 的 8 位 A/D 芯片的分辨率为 $10V\times 2^{-8}=39mV$；而 16 位的 A/D 是 $10V\times 2^{-16}=153\mu V$。

2) 满刻度误差

满刻度误差(Full Scale Error)也称增益误差，即输出全 1 时输入电压与理想输入量之差。

3) 转换速率

转换速率(Conversion Rate)是指完成一次 A/D 转换所需时间的倒数，是一个很重要的指标。A/D 转换器型号不同，转换速率差别很大。选用 A/D 转换器型号视应用需求而定，在被控系统的时间允许的情况下，应尽量选用便宜的逐次比较型 A/D 转换器。

4) 转换精度

A/D 转换器的转换精度(Conversion Accuracy)由模拟误差和数字误差组成。模拟误差是比较器、解码网络中的电阻值以及基准电压波动等引起的误差，数字误差主要包括丢失码误差和量化误差，前者属于非固定误差，由器件质量决定，后者和 A/D 输出数字量的位数有关，位数越多，误差越小。

5.1.2　C8051F020 的 ADC0 功能结构

C8051F020 的 ADC0 子系统就是一个 100ksps、12 位分辨率的逐次逼近寄存器型

ADC。ADC0 的最高转换速度为 100ksps，其转换时钟来源于系统时钟分频，分频值保存在寄存器 ADC0CF 的 ADCSC 位。C8051F020 的 ADC0 子系统功能框图如图 5-4 所示，它包括一个 9 通道的可编程模拟多路选择器（AMUX0）、一个可编程增益放大器（PGA0）和一个 100ksps、12 位分辨率的逐次逼近寄存器型 ADC，ADC 中集成了跟踪保持电路和可编程窗口检测器。AMUX0、PGA0、数据转换方式及窗口检测器都可用软件通过特殊功能寄存器来控制。ADC0 所使用的电压基准将在 5.3 节专门介绍。只有当 ADC0 控制寄存器中的 AD0EN 位被置 1 时 ADC0 子系统（ADC0、跟踪保持器和 PGA0）才被允许工作。当 AD0EN 位为 0 时，AD0C 子系统处于低功耗关断方式。

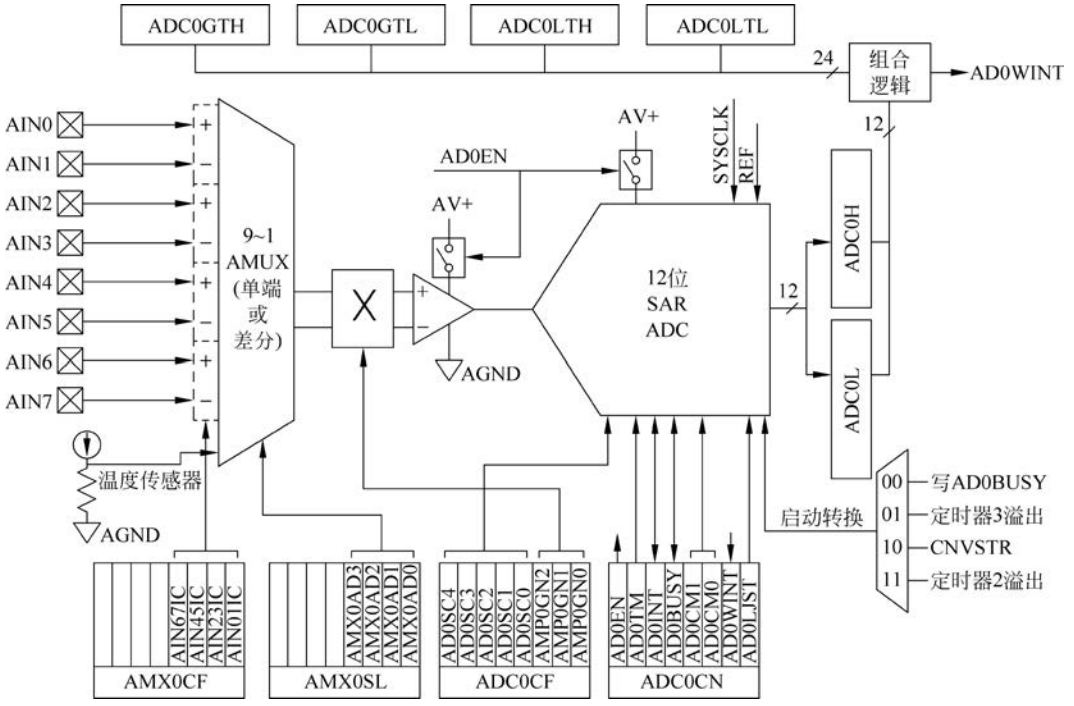


图 5-4　ADC0 子系统功能框图

从 ADC0 的功能框图可以看出，ADC0 的运行主要与图上标的 10 个 SFR 有关。8 个外部输入的模拟量可以通过配置寄存器 AMX0CF 设定为单端输入或双端输入；8 个外部输入的模拟量和一个内部温度传感器量通过通道选择寄存器 AMX0SL 设定在某一时刻通过多路选择器；从多路选择器出来的模拟量通过配置寄存器 ADC0CF 设定 ADC 转换速度和对模拟量的放大倍数；由控制寄存器 ADC0CN 对 ADC 进行模拟量转换的启动、启动方式、采样保持、转换结束、数字量格式等进行设定；12 位的转换好的数字量存放在数据字寄存器 ADC0H、ADC0L 中；ADC0 中提供了可编程窗口检测器，通过上下限寄存器 ADC0GTH、ADC0GTL、ADC0LTH、ADC0LTL 设定所需要的比较极限值。

在进行模拟量转换前设定好以上 SFR，CPU 就按设定好的模式在模拟量转换好时用指令读出数据寄存器中的数字量或在中断服务程序中读取数字量，然后再进行下一次的转换。ADC0 的电气特性见附录 C。

5.1.3 模拟多路选择器和 PGA

模拟多路选择器(Analog Multiplexer,AMUX)中的 8 个通道用于外部测量,而第 9 通道在内部被接到片内温度传感器。这 9 个模拟通道通过通道选择寄存器 AMX0SL 和配置寄存器 AMX0CF 进行选择和配置,可以将 AMUX 输入对编程为工作在差分或单端方式。这就允许用户对每个通道选择最佳的测量技术,甚至可以在测量过程中改变方式。在系统复位后 AMUX 的默认方式为单端输入。表 5-1 给出了每种配置下各通道的功能。

配置寄存器 AMX0CF 的格式如下:

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	复位值
—	—	—	—	AIN67IC	AIN45IC	AIN23IC	AIN01IC	00000000
位 7	位 6	位 5	位 4	位 3	位 2	位 1	位 0	SFR地址:0xBA

其中,各位的含义如下:

位 7～4——未使用。读=0000b;写=忽略。

位 3(AIN67IC)——AIN6、AIN7 输入对配置位。

0:AIN6 和 AIN7 为独立的单端输入。

1:AIN6 和 AIN7 为(分别为)+、−差分输入对。

位 2(AIN45IC)——AIN4、AIN5 输入对配置位。

0:AIN4 和 AIN5 为独立的单端输入。

1:AIN4、AIN5 为(分别为)+、−差分输入对。

位 1(AIN23IC)——AIN2、AIN3 输入对配置位。

0:AIN2 和 AIN3 为独立的单端输入。

1:AIN2、AIN3 为(分别为)+、−差分输入对。

位 0(AIN01IC)——AIN0、AIN1 输入对配置位。

0:AIN0 和 AIN1 为独立的单端输入。

1:AIN0、AIN1 为(分别为)+、−差分输入对。

注:对于被配置成差分输入的通道,ADC0 数据字格式为 2 的补码。

通道选择寄存器 AMX0SL 的格式如下:

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	复位值
—	—	—	—	AMX0AD3	AMX0AD2	AMX0AD1	AMX0AD0	00000000
位 7	位 6	位 5	位 4	位 3	位 2	位 1	位 0	SFR地址:0xBB

其中,各位的含义如下:

位 7～4——未使用。读=0000b;写=忽略。

位 3～0(AMX0AD3～0)——AMUX0 地址位。

0000～1111,根据表 5-1 选择 ADC 输入的通道。

表 5-1　模拟通道配置

		AMX0SL 位 3～0								
		0000	0001	0010	0011	0100	0101	0110	0111	1xxx
AMX0CF 位 3～0	0000	AIN0	AIN1	AIN2	AIN3	AIN4	AIN5	AIN6	AIN7	温度传感器
	0001	＋（AIN0） －（AIN1）		AIN2	AIN3	AIN4	AIN5	AIN6	AIN7	温度传感器
	0010	AIN0	AIN1	＋（AIN2） －（AIN3）		AIN4	AIN5	AIN6	AIN7	温度传感器
	0011	＋（AIN0） －（AIN1）		＋（AIN2） －（AIN3）		AIN4	AIN5	AIN6	AIN7	温度传感器
	0100	AIN0	AIN1	AIN2	AIN3	＋（AIN4） －（AIN5）		AIN6	AIN7	温度传感器
	0101	＋（AIN0） －（AIN1）		AIN2	AIN3	＋（AIN4） －（AIN5）		AIN6	AIN7	温度传感器
	0110	AIN0	AIN1	＋（AIN2） －（AIN3）		＋（AIN4） －（AIN5）		AIN6	AIN7	温度传感器
	0111	＋（AIN0） －（AIN1）		＋（AIN2） －（AIN3）		＋（AIN4） －（AIN5）		AIN6	AIN7	温度传感器
	1000	AIN0	AIN1	AIN2	AIN3	AIN4	AIN5	＋（AIN6） －（AIN7）		温度传感器
	1001	＋（AIN0） －（AIN1）		AIN2	AIN3	AIN4	AIN5	＋（AIN6） －（AIN7）		温度传感器
	1010	AIN0	AIN1	＋（AIN2） －（AIN3）		AIN4	AIN5	＋（AIN6） －（AIN7）		温度传感器
	1011	＋（AIN0） －（AIN1）		＋（AIN2） －（AIN3）		AIN4	AIN5	＋（AIN6） －（AIN7）		温度传感器
	1100	AIN0	AIN1	AIN2	AIN3	＋（AIN4） －（AIN5）		＋（AIN6） －（AIN7）		温度传感器
	1101	＋（AIN0） －（AIN1）		AIN2	AIN3	＋（AIN4） －（AIN5）		＋（AIN6） －（AIN7）		温度传感器
	1110	AIN0	AIN1	＋（AIN2） －（AIN3）		＋（AIN4） －（AIN5）		＋（AIN6） －（AIN7）		温度传感器
	1111	＋（AIN0） －（AIN1）		＋（AIN2） －（AIN3）		＋（AIN4） －（AIN5）		＋（AIN6） －（AIN7）		温度传感器

在表 5-1 中可看出，从多路选择器出来的哪一个通道和单端或差分输入由通道选择寄存器 AMX0SL 和配置寄存器 AMX0CF 进行选择和配置，表左边的垂直方向表示配置寄存器 AMX0CF 低 4 位值，指出各通道的单端还是差分输入，表上边的水平方向表示通道选择寄存器 AMX0SL 的低 4 位，选择 9 路输入的中的某一路。

PGA(Programmable Gain Amplifier)即可编程增益放大器，它对 AMUX 输出信号的放大倍数由 ADC0 配置寄存器 ADC0CF 中的 AMP0GN2～0 确定。PGA 增益可以用软件编程为 0.5、1、2、4、8 或 16，复位后的默认增益为 1。注意，PGA0 的增益对温度传感器也起作用。

配置寄存器 ADC0CF 的格式如下：

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	复位值
AD0SC4	AD0SC3	AD0SC2	AD0SC1	AD0SC0	AMP0GN2	AMP0GN1	AMP0GN0	11111000
位 7	位 6	位 5	位 4	位 3	位 2	位 1	位 0	SFR地址：0xBC

其中，各位的含义如下：

位 7～3(AD0SC4～0)——ADC0 SAR 转换时钟周期控制位。

SAR 转换时钟来源于系统时钟，由下面的方程给出：

$$AD0SC = \frac{SYSCLK}{CLK_{SAR0}} - 1$$

其中，AD0SC 表示 AD0SC4～0 中保持的数值，CLK_{SAR0} 表示所需要的 ADC0 SAR 时钟(注：ADC0 SAR 时钟应小于或等于 2.5MHz)。

位 2～0(AMP0GN2～0)——ADC0 内部放大器增益(PGA)。

000：增益＝1

001：增益＝2

010：增益＝4

011：增益＝8

10x：增益＝16

11x：增益＝0.5

5.1.4 ADC 的工作方式

1. 转换过程

ADC0 的转换过程由控制寄存器 ADC0CN 来设置和控制。

控制寄存器 ADC0CN 的格式如下：

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	复位值
AD0EN	AD0TM	AD0INT	AD0BUSY	AD0CM1	AD0CM0	AD0WINT	AD0LJST	00000000
位 7	位 6	位 5	位 4	位 3	位 2	位 1	位 0(可位寻址)	SFR地址：0xE8

其中，各位的含义如下：

位 7(AD0EN)——ADC0 使能位。

0：ADC0 禁止。ADC0 处于低耗停机状态。

1：ADC0 使能。ADC0 处于活动状态，并准备转换数据。

位 6(AD0TM)——ADC0 跟踪方式位。

0：当 ADC0 被使能时，除了转换期间之外一直处于跟踪方式。

1：由 AD0CM1～0 定义跟踪方式。

位 5(AD0INT)——ADC0 转换结束中断标志。该标志必须用软件清 0。

0：从最后一次将该位清 0 后，ADC0 还没有完成一次数据转换。

1：ADC0 完成了一次数据转换。

位 4(AD0BUSY)——ADC0 忙标志位。

读：

0：ADC0 转换结束或当前没有正在进行的数据转换。AD0INT 在 AD0BUSY 的下降

沿被置 1。

1：ADC0 正在进行转换。

写：

0：无作用。

1：若 AD0CM1～0＝00b，则启动 ADC0 转换。

位 3 和位 2(AD0CM1～0)——ADC0 转换启动方式选择位。

如果 AD0TM＝0，则：

00：向 AD0BUSY 写 1 启动 ADC0 转换。

01：定时器 3 溢出启动 ADC0 转换。

10：CNVSTR 上升沿启动 ADC0 转换。

11：定时器 2 溢出启动 ADC0 转换。

如果 AD0TM＝1，则：

00：向 AD0BUSY 写 1 时启动跟踪，持续 3 个 SAR 时钟，然后进行转换。

01：定时器 3 溢出启动跟踪，持续 3 个 SAR 时钟，然后进行转换。

10：只有当 CNVSTR 输入为逻辑低电平时 ADC0 跟踪，在 CNVSTR 的上升沿开始转换。

11：定时器 2 溢出启动跟踪，持续 3 个 SAR 时钟，然后进行转换。

位 1(AD0WINT)——ADC0 窗口比较中断标志。该位必须用软件清 0。

0：自该标志被清除后未发生过 ADC0 窗口比较匹配。

1：发生了 ADC0 窗口比较匹配。

位 0(AD0LJST)——ADC0 数据左对齐选择位。

0：ADC0H：ADC0L 寄存器数据右对齐。

1：ADC0H：ADC0L 寄存器数据左对齐。

C8051F020 单片机的 ADC0 有 4 种转换启动方式，由 ADC0CN 中的 ADC0 启动转换方式位(AD0CM1 和 AD0CM0)的状态决定。转换触发源有：

(1) 向 ADC0CN 的 AD0BUSY 位写 1；

(2) 定时器 3 溢出(即定时的连续转换)；

(3) 外部 ADC 转换启动信号的上升沿，CNVSTR；

(4) 定时器 2 溢出(即定时的连续转换)。

AD0BUSY 位在转换期间被置 1，转换结束后复 0。AD0BUSY 位的下降沿触发一个中断(当被允许时)并将中断标志 AD0INT(ADC0CN.5)置 1。转换数据被保存在 ADC 数据字的 MSB 和 LSB 寄存器：ADC0H 和 ADC0L。转换数据在寄存器对 ADC0H：ADC0L 中的存储方式可以是左对齐或右对齐的，由 ADC0CN 寄存器中 AD0LJST 位的编程状态决定。当通过向 AD0BUSY 写 1 启动数据转换时，应查询 AD0INT 位以确定转换何时结束(也可以使用 ADC0 中断)。建议的查询步骤如下：

(1) 写 0 到 AD0INT；

(2) 向 AD0BUSY 写 1；

(3) 查询并等待 AD0INT 变为 1；

(4) 处理 ADC0 数据。

2. 跟踪方式

ADC0CN 中的 AD0TM 位控制 ADC0 的跟踪保持方式。在默认状态下,除了转换期间外,ADC0 输入被连续跟踪。当 AD0TM 位为逻辑 1 时,ADC0 工作在低功耗跟踪保持方式。在该方式下,每次转换之前都有 3 个 SAR 时钟的跟踪周期(在启动转换信号有效之后)。当 CNVSTR 信号用于在低功耗跟踪保持方式启动转换时,ADC0 只在 CNVSTR 为低电平时跟踪;在 CNVSTR 的上升沿开始转换(见图 5-5)。当整个芯片处于低功耗待机或休眠方式时,跟踪可以被禁止(关断)。当 AMUX 或 PGA 的设置频繁改变时,低功耗跟踪保持方式也非常有用,可以保证建立时间需求得到满足。

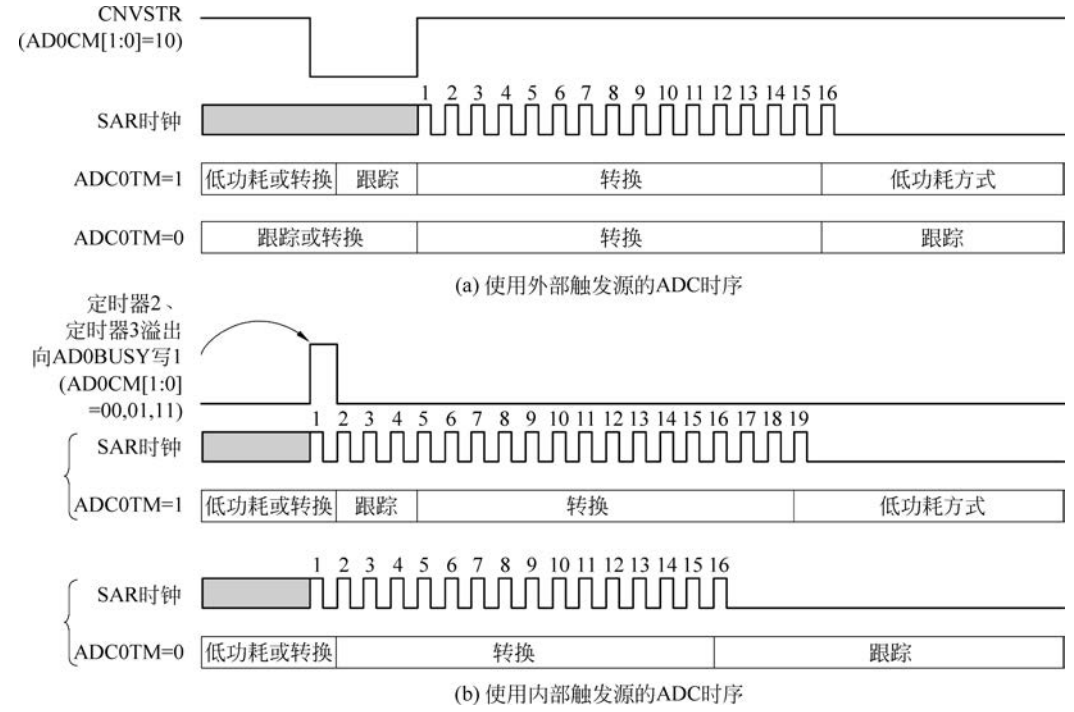


图 5-5　12 位 ADC 跟踪和转换时序

3. 建立时间要求

当 ADC0 输入配置发生改变时(AMUX 或 PGA 的选择发生变化),在进行一次精确的转换之前需要有一个最小的跟踪时间。该跟踪时间由 ADC0 模拟多路器的电阻、ADC0 采样电容、外部信号源阻抗及所要求的转换精度决定。图 5-6 给出了单端和差分方式下等效的 ADC0 输入电路。注意,这两种等效电路的时间常数完全相同。对于一个给定的建立精度(SA),所需要的 ADC0 建立时间可以用方程估算:

$$t = \ln\left[\frac{2^n}{\mathrm{SA}}\right] \times R_{\mathrm{TOTAL}} C_{\mathrm{SAMPLE}}$$

其中:

SA 是建立精度,用一个 LSB 的分数表示(例如,建立精度 0.25 对应 1/4 LSB);

t 为所需要的建立时间,以秒为单位;

R_{TOTAL} 为 ADC0 模拟多路器电阻与外部信号源电阻之和;

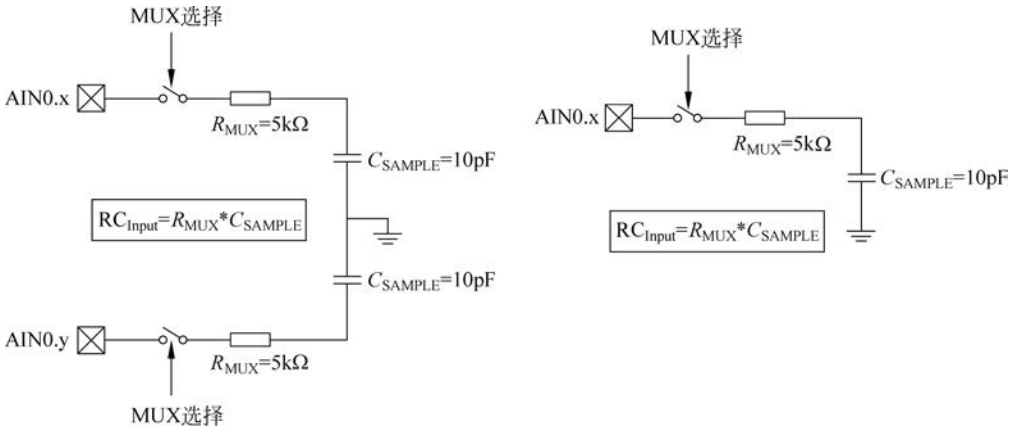


图 5-6　ADC0 等效输入电路

n 为 ADC0 的分辨率，用比特表示。

当测量温度传感器的输出时，RTOTAL 等于 RMUX。

注意：在低功耗跟踪方式，每次转换需要用 3 个 SAR 时钟跟踪。对于大多数应用，3 个 SAR 时钟可以满足跟踪需要。

4. 转换结果格式

12 位的转换好的数字量存放在数据字寄存器 ADC0H、ADC0L 中。

数据字寄存器 ADC0H 的格式如下：

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	复位值
								00000000
位 7	位 6	位 5	位 4	位 3	位 2	位 1	位 0	SFR地址：0xBF

其中，各位的含义如下：

位 7～0——ADC0 数据字高字节。

当 AD0LJST＝0：即寄存器数据右对齐，位 7～4 为位 3 的符号扩展位。位 3～0 是 12 位 ADC0 数据字的高 4 位。

当 AD0LJST＝1：即寄存器数据左对齐，位 7～0 为 12 位 ADC0 数据字的高 8 位。

数据字寄存器 ADC0L 的格式如下：

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	复位值
								00000000
位 7	位 6	位 5	位 4	位 3	位 2	位 1	位 0	SFR地址：0xBE

其中，各位的含义如下：

位 7～0——ADC0 数据字低字节。

当 AD0LJST＝0：即寄存器数据右对齐，位 7～0 为 12 位 ADC0 数据字的低 8 位。

当 AD0LJST＝1：即寄存器数据左对齐，位 7～4 为 12 位 ADC0 数据字的低 4 位。位 3～0 总是 0。

表 5-2 列出了输入信号与转换结果代码及不同的方式之间的对应关系。

表 5-2　ADC 数据字转换表

AIN0 为单端输入方式：

(AMX0CF＝0x00,AMX0SL＝0x00)

AIN0－AGND(伏)	ADC0H：ADC0L(AD0LJST＝0)	ADC0H：ADC0L(AD0LJST＝1)
VREF×(4095/4096)	0x0FFF	0xFFF0
VREF/2	0x0800	0x8000
VREF×(2047/4096)	0x07FF	0x7FF0
0	0x0000	0x0000

AIN0－AIN1 为差分输入对：

(AMX0CF＝0x01,AMX0SL＝0x00)

AIN0－AIN1(伏)	ADC0H：ADC0L(AD0LJST＝0)	ADC0H：ADC0L(AD0LJST＝1)
VREF×(2047/2048)	0x07FF	0x7FF0
VREF/2	0x0400	0x4000
VREF×(1/2048)	0x0001	0x0010
0	0x0000	0x0000
－VREF×(1/2048)	0xFFFF(－1d)	0xFFF0
－VREF/2	0xFC00(－1024d)	0xC000
－VREF	0xF800(－2048d)	0x8000

5.1.5　ADC0 可编程窗口检测器

ADC0 可编程窗口检测器提供一个中断，当 ADC0 转换值在 ADC0 下限(大于)寄存器 ADC0GTH：ADC0GTL 和 ADC0 上限(小于)寄存器 ADC0LTH：ADC0LTL 范围之内，并且中断开启时，引发相应中断。ADC0 可编程窗口检测器可应用于节能场合，例如让系统处于空闲方式，当 ADC0 输入信号(例如温度)在窗口检测器预设值的范围内时，引发中断，唤醒 CIP-51 进行相应的处理，既节能，又达到了监控的目的，同时也节省代码空间和 CPU 带宽。窗口检测器中断标志(ADC0CN 中的 AD0WINT 位)也可用于查询方式。

窗口设定值的高字节和低字节被装入 ADC0 下限(大于)和 ADC0 上限(小于)寄存器(ADC0GTH、ADC0GTL、ADC0LTH 和 ADC0LTL)。注意，窗口检测器标志既可以在测量数据位于用户编程的极限值以内时有效，也可以在测量数据位于用户编程的极限值以外时有效，这取决于 ADC0GTx 和 ADC0LTx 寄存器的编程值。

在默认情况下，ADC0GTH：ADC0GTL＝0xFFFF；ADC0LTH：ADC0LTL＝0x0000。所以使得电平在全范围内均不会引发监控中断。

在 ADC0LTH：ADC0LTL＞ADC0GTH：ADC0GTL 情况下，窗口检测中断条件为：

ADC0LTH:ADC0LTL ＜ADC0 转换值＜ADC0GTH:ADC0GTL

在 ADC0LTH：ADC0LTL＜ADC0GTH：ADC0GTL 情况下，窗口检测中断条件为：

ADC0 转换值＞ADC0GTH:ADC0GTL

或

ADC0 转换值＜ADC0LTH:ADC0LTL

图 5-7 中说明一个右对齐数据格式下，单端输入窗口检测器设置例子。

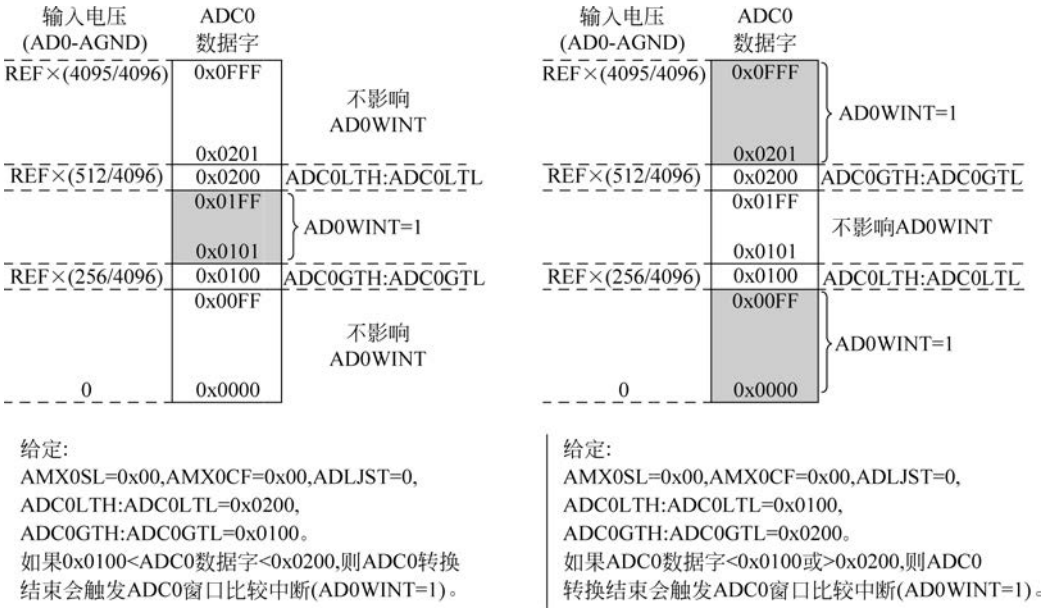


图 5-7　ADC0 右对齐的单端数据窗口中断示例

在图 5-7 的左边：

```
ADC0LTH:ADC0LTL = 0x0200,ADC0GTH:ADC0GTL = 0x0100
```

则当 0x0100＜ADC0 转换值＜0x0200 时，AD0WINT 自动置 1。若中断开启，则引发相应的中断(中断号为 8，与 ADC0 不是同一个中断)。AD0WINT 需要软件清 0。

在图 5-7 的右边：

```
ADC0LTH: ADC0LTL = 0x0100,ADC0GTH: ADC0GTL = 0x0200
```

右边的与左边的范围刚好相反。则当 ADC0 转换值＞0x0200 或 ADC0 转换值＜0x0100 时，窗口检测条件，产生 AD0WINT 自动置 1。

图 5-8 中说明一个右对齐数据格式下，差动输入窗口检测器设置例子。在差动模式下，右对齐数据格式与 int 相当，0x8000～0xFFFF 表示负数，0xFFFF 为－1，0xF800 为转换最小值(即负电平最大值)。

在图 5-8 的左边：

```
ADC0LTH: ADC0LTL = 0x0100,ADC0GTH:ADC0GTL = 0xFFFF
```

则当 0xFFFF(即－1)＜ADC0 转换值＜0x0100 时，AD0WINT 自动置 1。

在图 5-8 的右边：

```
ADC0LTH: ADC0LTL = 0xFFFF,ADC0GTH: ADC0GTL = 0x0100
```

右边的与左边的范围刚好相反。则当 ADC0 转换值＞0x0100 或 ADC0 转换值＜0xFFFF

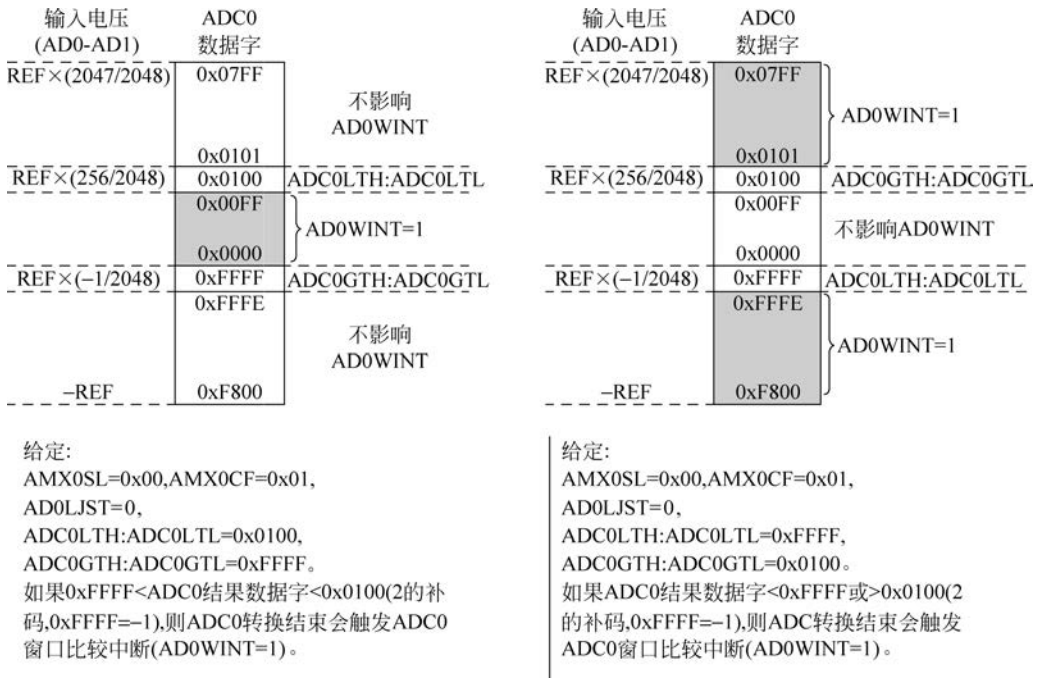


图 5-8　ADC0 右对齐的差分数据窗口中断示例

时,窗口检测条件,产生 AD0WINT 自动置 1。

至于左对齐数据格式相当于将转换值乘以 16(左移 4 位),其他有关窗口检测的规则与右对齐相似 ADC0 左对齐的单端数据输入的窗口检测中断示例见图 5-9,差分数据输入的窗口检测中断示例见图 5-10,请读者自行分析。

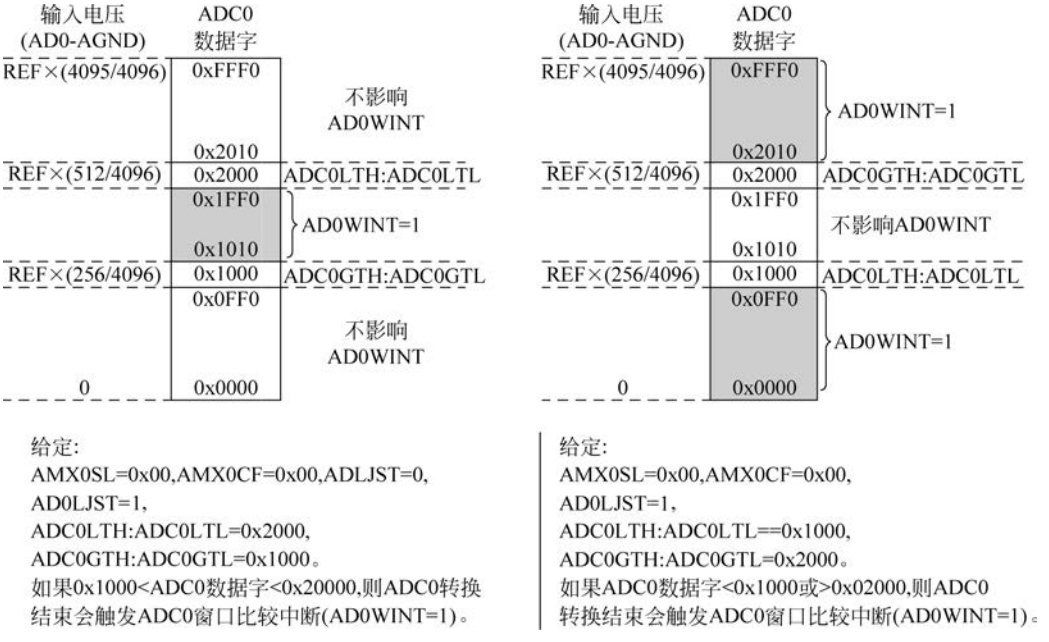
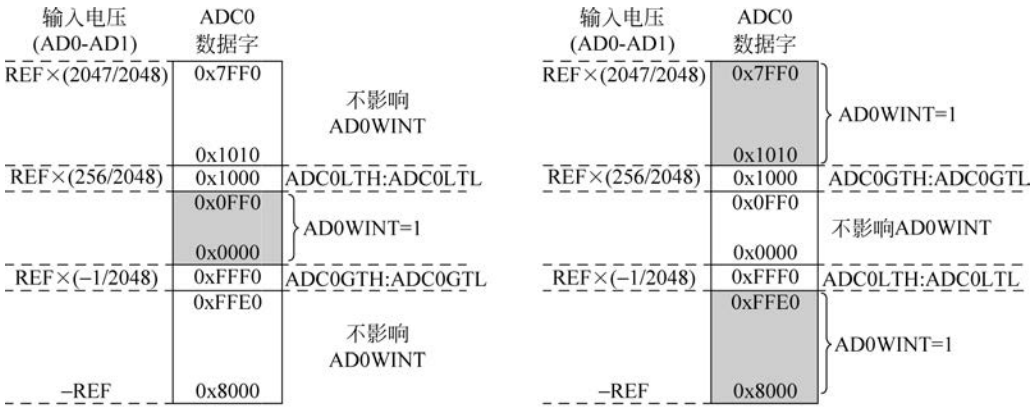


图 5-9　ADC0 左对齐的单端数据窗口中断示例



给定:
AMX0SL=0x00,AMX0CF=0x01,
AD0LJST=1,
ADC0LTH:ADC0LTL=0x1000,
ADC0GTH:ADC0GTL=0xFFF0。
ADC0转换结束,如果0xFFF0<ADC0结果数据字<0x1000(2的补码),则会触发ADC0窗口比较中断(AD0WINT=1)。

给定:
AMX0SL=0x00,AMX0CF=0x01,
AD0LJST=1,
ADC0LTH:ADC0LTL=0xFFF0,
ADC0GTH:ADC0GTL=0x1000。
ADC0转换结束,如果数据字<0xFFF0或>0x1000(2的补码),则会触发ADC0窗口比较中断(AD0WINT=1)。

图 5-10　ADC0 左对齐的差分数据窗口中断示例

5.1.6　ADC1(8 位 ADC)

C8051F020 还有一个 ADC1 子系统,包括一个 8 通道的可配置模拟多路开关(AMUX1)、一个可编程增益放大器(PGA1)和一个 500ksps、8 位分辨率的逐次逼近寄存器型 ADC,该 ADC 中集成了跟踪保持电路。ADC1 的原理框图如图 5-11 所示。AMUX1、PGA1 及数据

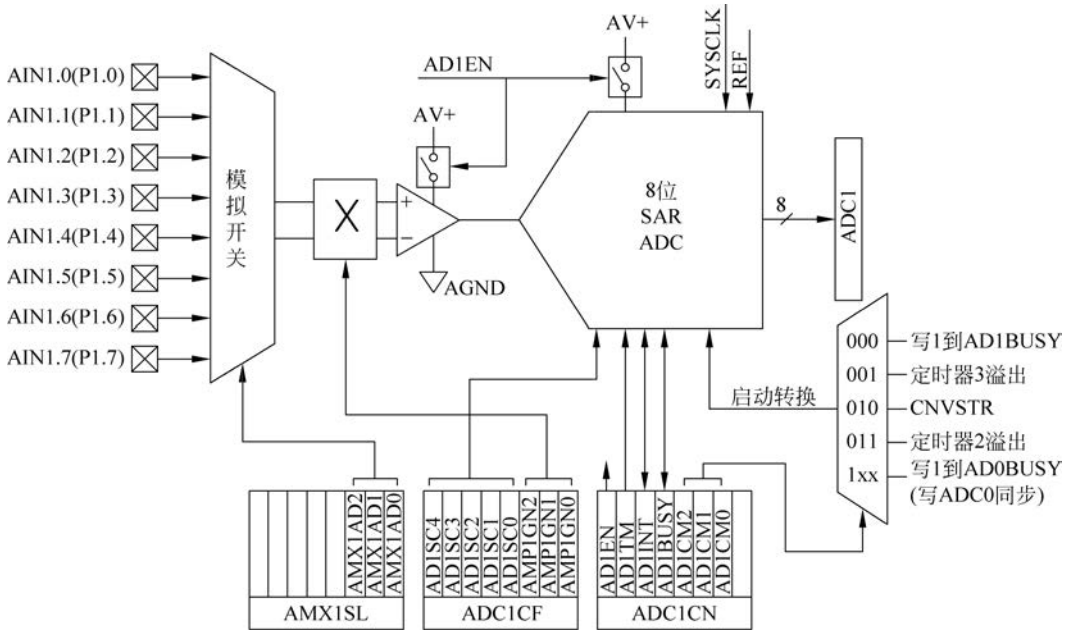


图 5-11　ADC1 原理框图

转换方式都可用软件通过特殊功能寄存器来配置。与 ADC1 工作有关的 SFR 有 ADC1 配置寄存器 ADC1CF、AMUX 配置寄存器 AMX1SL、ADC1 控制寄存器 ADC1CN、ADC1 数据寄存器 ADC1。只有当 ADC1 控制寄存器 ADC1CN 中的 AD1EN 位被置 1 时 ADC1 子系统(8 位 ADC、跟踪保持器和 PGA)才被使能。当 AD1EN 位为 0 时,ADC1 子系统处于低功耗关断方式。

1. 模拟多路开关和 PGA

ADC1 有 8 个通道用于测量,用寄存器 AMX1SL 选择通道。PGA 对 AMUX 输出信号的放大倍数由 ADC1 配置寄存器 ADC1CF 中的 AMP1GN1～0 确定。PGA 增益可以用软件编程为 0.5、1、2、4。复位时的默认增益为 0.5。

注意:AIN1 引脚也作为端口 1 的 I/O 引脚,当用作 ADC1 输入时必须被配置为模拟输入。为了将 AIN1 的某个引脚配置为模拟输入,要将寄存器 P1MDIN 中的对应位设置为 0。被选作模拟输入的端口 1 引脚不进入数字 I/O 交叉开关。

配置寄存器 ADC1CF 的格式如下:

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	复位值
AD1SC4	AD1SC3	AD1SC2	AD1SC1	AD1SC0	—	AMP1GN1	AMP1GN0	11111000
位 7	位 6	位 5	位 4	位 3	位 2	位 1	位 0	SFR地址:0xAB

其中,各位的含义如下:

位 7～3:(AD1SC4～0)——ADC1 SAR 转换时钟周期控制位。

SAR 转换时钟频率由下面的方程计算:

$$\mathrm{AD1SC}=\frac{\mathrm{SYSCLK}}{\mathrm{CLK_{SAR1}}}-1$$

其中:SYSCLK 为系统时钟,AD1SC 为 AD1SC4～0 中的 5 位数值(注意,ADC1SAR 转换时钟应小于或等于 6MHz)。

位 2——未使用。读=0b;写=忽略。

位 1 和位 0(AMP1GN1～0)——ADC1 内部放大器增益(PGA)。

00:增益=0.5

01:增益=1

10:增益=2

11:增益=4

AMUX 配置寄存器 AMX1SL 的格式为:

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	复位值
—	—	—	—	—	AMX1AD2	AMX1AD1	AMX1AD0	00000000
位 7	位 6	位 5	位 4	位 3	位 2	位 1	位 0	SFR地址:0xAC

其中,各位的含义如下:

位 7～3——未使用。读=00000b;写=忽略。

位 2～0——(AMX1AD2～0)——AMX1 地址位。

000～111:ADC1 输入选择如下:

000——选择 AIN1.0

001——选择 AIN1.1
010——选择 AIN1.2
011——选择 AIN1.3
100——选择 AIN1.4
101——选择 AIN1.5
110——选择 AIN1.6
111——选择 AIN1.7

2. ADC1 的工作方式

ADC1 的最高转换速度为 500ksps。ADC1 的转换时钟(SAR1 时钟)来源于系统时钟分频。由 ADC1CF 寄存器的 AD1SC 位决定(系统时钟/(AD1SC+1),0≤AD1SC≤31)。ADC1 转换时钟频率最大为 6MHz。ADC1 的转换过程主要由控制寄存器 ADC1CN 的设置来控制。ADC1 控制寄存器 ADC1CN 的格式为:

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	复位值
AD1EN	AD1TM	AD1INT	AD1BUSY	AD1CM2	AD1CM1	AD1CM0	—	00000000
位 7	位 6	位 5	位 4	位 3	位 2	位 1	位 0	SFR地址: 0xAC

其中,各位的含义如下:

位 7(AD1EN)——ADC1 使能位。

0:ADC1 禁止。ADC1 处于低功耗停机状态。

1:ADC1 使能。ADC1 处于活动状态,并准备转换数据。

位 6(AD1TM)——ADC1 跟踪方式位。

0:一般跟踪方式。当 ADC1 被使能时,除了转换期间之外一直处于跟踪方式。

1:低功耗跟踪方式。由 AD1CM2~0 定义跟踪方式。

位 5(AD1INT)——ADC1 转换结束中断标志,该标志必须用软件清 0。

0:从最后一次将该位清 0 后,ADC1 还没有完成一次数据转换。

1:ADC1 完成一次数据转换。

位 4(AD1BUSY)——ADC1 忙标志位。

读:

0:ADC1 转换结束或不在进行数据转换。AD1INT 在 AD1BUSY 的下降沿被置 1。

1:ADC1 正在进行转换。

写:

0:无效。

1:若 AD1CM2~0=000b,则启动 ADC1 转换。

位 3~1(AD1CM2~0)——ADC1 转换启动方式选择。

AD1TM=0:

000——向 AD1BUSY 写 1 启动 ADC1 转换。

001——定时器 3 溢出启动 ADC1 转换。

010——CNVSTR 上升沿启动 ADC1 转换。

011——定时器 2 溢出启动 ADC1 转换。

1xx——向 AD0BUSY 写 1 启动 ADC1 转换(与 ADC0 软件命令转换同步)。

AD1TM=1:

000——向 AD1BUSY 写 1 时启动跟踪并持续 3 个 SAR1 时钟,然后进行转换。

001——定时器 3 溢出启动跟踪并持续 3 个 SAR1 时钟,然后进行转换。

010——只有当 CNVSTR 输入为逻辑低电平时才启动 ADC1 跟踪,在 CNVSTR 上升沿开始转换。

011——定时器 2 溢出启动跟踪并持续 3 个 SAR1 时钟,然后进行转换。

1xx——向 AD0BUSY 写 1 启动跟踪并持续 3 个 SAR1 时钟,然后进行转换。

位 0——未使用。读=0b;写=忽略。

ADC1 有 5 种 A/D 转换启动方式,由 ADC1CN 中的 ADC1 启动转换方式位(AD1CM2~0)的编程状态决定。转换启动源有:

(1) 向 ADC1CN 的 AD1BUSY 位写 1;

(2) 定时器 3 溢出(即定时的连续转换);

(3) 外部 ADC 转换启动信号 CNVSTR 的上升沿;

(4) 定时器 2 溢出(即定时的连续转换)。

(5) 向 ADC0CN 的 AD0BUSY 位写 1(用一个软件命令启动 ADC1 和 ADC0)。

AD1BUSY 位在转换期间被置 1,转换结束后清 0。AD1BUSY 位的下降沿触发一个中断(当被允许时)并将 ADC1CN 中的中断标志置 1。转换结果保存在 ADC1 的数据字 ADC1 中。当采用向 AD1BUSY 位写 1 这一启动方式时,建议通过查询 AD1INT 来确定转换何时完成。查询步骤同 ADC0。

3. 跟踪方式

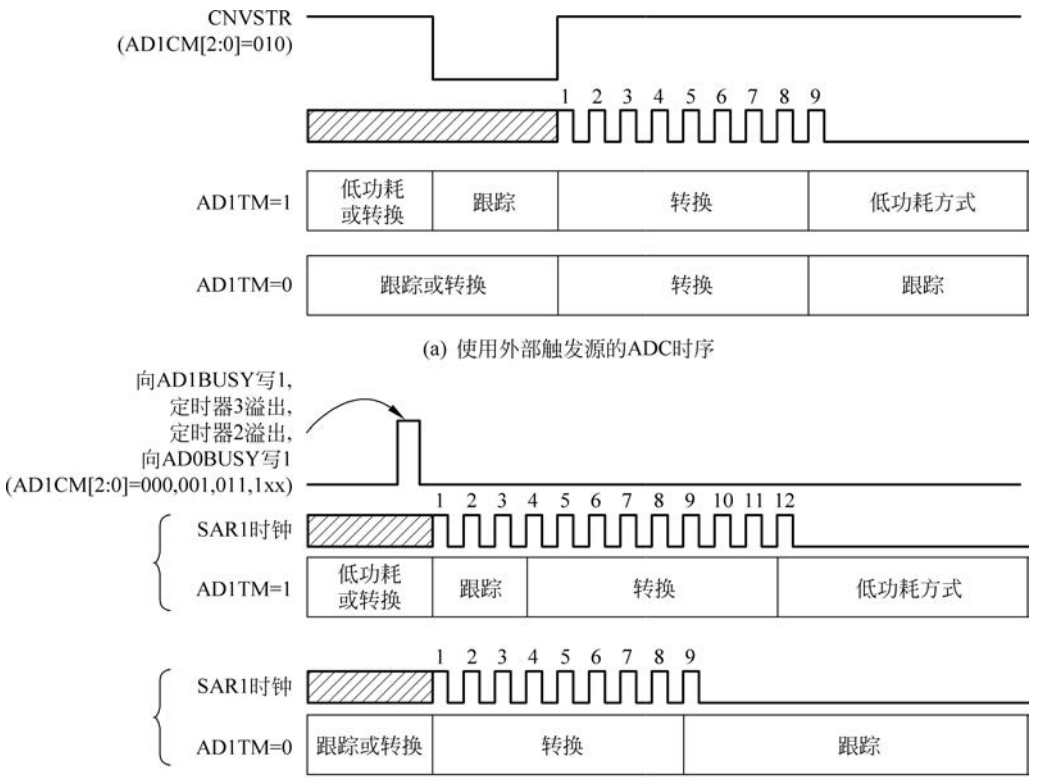
ADC1CN 中的 AD1TM 位控制 ADC1 的跟踪保持方式。在默认状态下,ADC1 输入被连续跟踪(转换期间除外)。当 AD1TM 位被设置为逻辑 1 时,ADC1 工作在低功耗跟踪方式。在该方式下,每次转换之前都要有 3 个 SAR 时钟的跟踪周期(在启动转换信号之后)。当在低功耗跟踪方式下用 CNVSTR 信号作为转换启动源时,只在 CNVSTR 为低电平时跟踪,从 CNVSTR 的上升沿开始转换。当整个芯片处于低功耗停机或休眠方式时,跟踪被禁止。由于需要有建立时间,因此低功耗跟踪保持方式在需要频繁改变 AMUX 和 PGA 的场合也是非常有用的。ADC1 跟踪和转换时序的举例见图 5-12。

5.1.7 模数转换举例

1. 片内温度传感器数据采集

C8051F020 的 ADC0 中有一个片内温度传感器,在图 5-1 中已标出。温度传感器产生一个与器件内部温度成正比的电压,该电压作为一个单端输入提供给 ADC(模数转换器)的多路选择器。当选择温度传感器作为 ADC 的输入并且 ADC 启动一次转换后,可以通过简单的数学运算将 ADC 的输出结果转换为用度数表示的温度。温度转换器的典型应用有系统环境检测、系统过热测试和在基于热电偶的应用中测量冷端温度。

为了能使用温度传感器,它首先必须被允许,ADC 及其相关的偏置电路也必须被允许。ADC 可以使用内部电压基准,也可以使用外部电压基准。本例子使用内部电压基准。ADC 转换的结果代码可以选择为左对齐或右对齐。本例子使用左对齐,这样可使代码的权值与



(a) 使用外部触发源的ADC时序

(b) 使用内部触发源的ADC时序

图 5-12　ADC1 跟踪和转换时序举例

ADC 的位数(12 或 10)无关。有关 ADC 转换步骤为：

(1) 通过将 TEMPE(REF0CN.2)设置为 1 来允许温度传感器工作。模拟偏置发生器和内部电压基准的允许位位于 REF0CN 中(分别为 REF0CN.1 和 REF0CN.0)。例如：

```
mov     REF0CN,#07 h          ;允许温度传感器、模拟偏置发生器和电压基准
```

(2) 选择温度传感器作为 ADC 的输入。这可以通过写 AMX0SL 来完成，例如：

```
mov     AMX0SL,#08 h          ;选择温度传感器作为 ADC 输入
```

AM0CF 的取值如何以及 AMUX 配置寄存器选择 ADC 是单端输入还是差分输入并不影响温度传感器工作。

(3) 设置位于 ADC0CF 中的 ADCSAR 时钟分频系数，特别是 ADC 转换时钟的周期至少应为 500ns。

(4) 选择 ADC 的增益。在单端方式下，ADC 能够接受的最大直流输入电压等于 VREF。如果使用内部电压基准，则该值大约为 2.4V。温度传感器所能产生的最大电压值稍大于 1V。因此，可以安全地将 ADC 的增益设置为 2，以提高温度分辨率。设置 ADC 增益的配置位在 ADC0CF 中。所以有：

```
mov     ADC0CF, #41h          ;设置 ADC 的时钟为 SYSCLK/8、ADC 增益为 2
```

其余的 ADC 配置位在 ADC0CN 中。这是一个可以位寻址的特殊功能寄存器。可以选择任何一种有效的转换启动源：定时器 2 或定时器 3 溢出、向 AD0BUSY 写 1 或使用外部 CNVSTR。后面的软件示例使用定时器 3 溢出作为转换启动源。这里采用向 AD0BUSY 写 1 的方式。

(5) 通过写入下面的控制字，将 ADC 配置为低功耗跟踪方式，采用向 AD0BUSY 写 1 作为转换启动信号，输出数据采用左对齐格式：

```
mov     ADC0CN, #C1H          ;允许 ADC; 允许低功耗跟踪方式
                              ;清除转换完成中断
                              ;选择 AD0BUSY 作为转换启动源
                              ;清除窗口比较中断
                              ;设置输出数据格式为左对齐
```

(6) 至此，可以通过将 AD0BUSY 写 1 来启动一次转换：

```
setb    AD0BUSY               ;启动转换
```

(7) 用查询或中断方式(ADC0CN 的 AD0INT 位)等待转换完成，ADC 输出寄存器(即 ADC0H 和 ADC0L 中的 16 位数值)中的值就是与器件内部的绝对温度成正比的代码。下面说明如何通过这一代码得到温度的摄氏度数值。

温度传感器产生一个与器件内部绝对温度成正比的电压输出。温度传感器的传输特性如图 5-13 所示。式(5-1)的方程给出这一电压与温度的摄氏度数值之间的关系：

$$V_{temp} = (2.86\text{mV}/℃) \times \text{Temp} + 776\text{mV} \tag{5-1}$$

其中：

V_{temp}——温度传感器的输出电压；

Temp——器件内部的摄氏温度值。

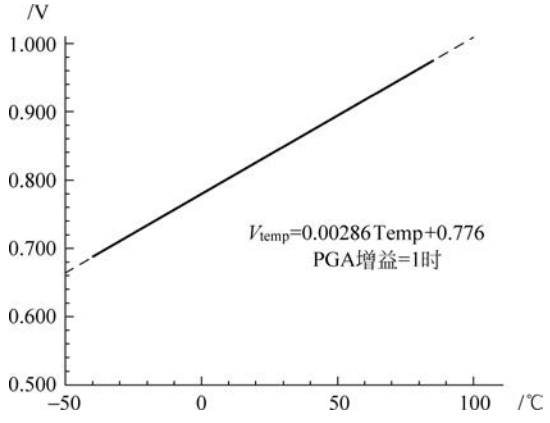


图 5-13　温度传感器的传输特性

温度传感器的电压不能直接在器件外部测量，它出现在 ADC 多路选择器的输入端，允许 ADC 测量该电压值并产生一个与电压值成正比的输出代码。ADC 在左对齐、单端方式下产生的输出代码与输入电压成正比，见式(5-2)。

$$\text{CODE} = V_{in} \times \frac{\text{Gain}}{V_{REF}} \times 2^{16} \tag{5-2}$$

其中：

CODE——左对齐的 ADC 输出代码；

Gain——PGA 的增益；

V_{REF}——电压基准的电压值，如果使用内部 V_{REF}，则大约为 2.43V。

把式(5-1)代入式(5-2)，并假设 Gain＝2 和 V_{REF}＝2.43V，解方程得到输出温度值为式(5-3)：

$$\text{Temp}=\frac{(\text{CODE}-41857)}{154} \tag{5-3}$$

其中：

Temp——温度的摄氏度数值；

CODE——左对齐的 ADC 输出代码。

温度传感器测量的是器件的内部温度。如果希望测量环境温度，则必须考虑器件的自热效应。由于器件功率消耗的原因，测量值很可能比环境温度值高几度，为了得到环境温度，应从结果中减去因自热产生的温度增加值。这一温度值可以通过计算或测量得到。

有很多因素影响器件的自热效应。其中最主要的是电源电压、工作频率、封装的热耗散特性、器件在 PCB 中的安装方式以及封装外壳周围的空气流通情况。温度增加值可以通过将器件的功率消耗乘以封装的热耗散常数(通常称为 θ_{JA})来计算。在用这一常数时假定采用标准的 PCB 安装方式，所有的引脚都焊到电路板上，封装周围没有气流通过。

例如，一个工作在 11.0592MHz、采用 3.3V 电源电压的 C8051F005 单片机，其功率消耗大致为 35mW。对于 64 引脚的 TQFP 封装，其 θ_{JA} 值是 39.5℃/W。这等价于 $39.5\times 35e^{-3}$ 的自热温度值，大约相当于 1.4℃。

因自热而导致的温度增加可以用几种方法测量。一种方法是在器件上电之后立即启动一次转换，得到一个“冷”温度值；然后再工作大约 1min 之后再测量一次，得到一个“热”温度值。这两个测量值的差就是因自热而产生的温度增加值。

另一种方法是让器件从一个低的 SYSCLK 频率开始工作，进行一次温度测量，然后再让器件工作在较高频率进行一次温度测量，取两者之差。在时钟频率更低时自热值是可忽略的，因为此时器件的功耗很低。

片内温度传感器的测量方法可以采用查询法或中断法。

(1) 查询法程序。

```
//此程序示范了 ADC0 的查询操作模式,ADC0 配置为写 AD0BUSY 作为转换的开始信号,测
//量片内温度传感器,温度传感器的输出转换为摄氏度由 UART0 传输出去。可以通过 PC
//超级终端来观察温度采样值。超级终端使用方法为：以 Windows 2000 系统为例,选择
//"开始"→"程序"→"附件"→"通信"→"超级终端"命令进入超级终端(HyperTerminal)
//应用程序界面,新建一个通信终端,取名为 temp。单击"确定"按钮。选择终端的连接的
//串口(如串行口 1)
//设置对应于单片通信机程序的通信的格式和协议(如波特率、每字节的位数等)
//假设在 XTAL1 和 XTAL2 之间连接 22.1184MHz 晶体
//系统时钟频率存储在全局常量 SYSCLK,目标器件 UART 波特率存储在全局常量 BAUDRATE
//目标器件：C8051F020
//链接工具：KEIL C51 6.03 / KEIL EVAL C51
//-----------------------------------------------
```

```
//包含文件
//-------------------------------------------
#include <c8051f020.h>                    //SFR 声明
#include <stdio.h>
//-------------------------------------------
//C8051F02X 的 16 位 SFR 定义
//-------------------------------------------
sfr16 DP = 0x82;                          //数据指针
sfr16 TMR3RL = 0x92;                      //定时器 3 重装值
sfr16 TMR3 = 0x94;                        //定时器 3 计数器
sfr16 ADC0 = 0xbe;                        //ADC0 数据
sfr16 ADC0GT = 0xc4;                      //ADC0 大于窗口
sfr16 ADC0LT = 0xc6;                      //ADC0 小于窗口
sfr16 RCAP2 = 0xca;                       //定时器 2 捕捉/重装
sfr16 T2 = 0xcc;                          //定时器 2
sfr16 RCAP4 = 0xe4;                       //定时器 4 捕捉/重装
sfr16 T4 = 0xf4;                          //定时器 4
sfr16 DAC0 = 0xd2;                        //DAC0 数据
sfr16 DAC1 = 0xd5;                        //DAC1 数据
//-------------------------------------------
//全局常量
//-------------------------------------------
#define SYSCLK 22118400                   //系统时钟频率(Hz)
#define BAUDRATE 9600                     //UART 波特率(b/s)
//-------------------------------------------
//函数原型
//-------------------------------------------
void SYSCLK_Init(void);
void PORT_Init(void);
void UART0_Init(void);
void ADC0_Init(void);
//-------------------------------------------
//主程序
//-------------------------------------------
void main(void) {
    long temperature;                     //温度百分之一的精度
    int temp_int, temp_frac;              //温度的整数和小数部分
    WDTCN = 0xde;                         //禁止看门狗定时器
    WDTCN = 0xad;
    SYSCLK_Init();                        //初始化振荡器
    PORT_Init();                          //初始化数据交叉开关和通用 I/O 端口
    UART0_Init();                         //初始化 UART0
    ADC0_Init();                          //初始化和使能 ADC
    while (1) {
        AD0INT = 0;                       //清除转换结束标记
        AD0BUSY = 1;                      //开始转换
        while (AD0INT == 0);              //等待转换结束
        temperature = ADC0;               //读 ADC0 数据
        //计算温度精度为百分之一度
        temperature = temperature - 41857;    //减去偏移量,使之对应 0℃ 的值
        temperature = (temperature * 100L) / 154;       //计算出对应的温度值(2.86mV/℃ )
```

```
        temp_int = temperature / 100;          //得到温度值的整数部分
        temp_frac = temperature - (temp_int * 100);          //得到温度值的小数部分
        printf("Temperature is %+02d.%02d\n", temp_int, temp_frac);    //从串口输出
}
}
//-----------------------------------------------
//系统时钟初始化
//-----------------------------------------------
//此程序初始化系统时钟使用 22.1184MHz 晶体作为时钟源
void SYSCLK_Init(void)
    {
    int i;                          //延时计数器
    OSCXCN = 0x67;                  //启动外部振荡器 22.1184MHz 晶体
    for (i=0; i<256; i++);          //等待振荡器启动 (>1ms)
    while (!(OSCXCN & 0x80));       //等待晶体振荡器稳定
    OSCICN = 0x88;                  //选择外部振荡器作为系统时钟源并使能丢失时钟检测器
}
//-----------------------------------------------
//I/O 端口初始化
//-----------------------------------------------
//配置数据交叉开关和通用 I/O 端口
void PORT_Init(void)
    {
    XBR0 = 0x04;                    //使能 UART0
    XBR1 = 0x00;
    XBR2 = 0x40;                    //使能数据交叉开关和弱上拉
    P0MDOUT |= 0x01;                //允许 TX0 为推挽输出
    }
//-----------------------------------------------
//UART0 初始化
//-----------------------------------------------
//配置 UART0 使用定时器 1 产生波特率
void UART0_Init(void)
    {
    SCON0 = 0x50;                   //SCON0: 模式 1,8 位 UART,允许 RX
    TMOD = 0x20;                    //TMOD: 1 定时器, 模式 2, 8 位重装
    TH1 = -(SYSCLK/BAUDRATE/16);    //按波特率设置定时器 1 重装值
    TR1 = 1;                        //启动定时器 1
    CKCON |= 0x10;                  //定时器 1 使用系统时钟为时基
    PCON |= 0x80;                   //SMOD=1
    TI0 = 1;                        //表示就绪
}
//-----------------------------------------------
//ADC0 初始化
//-----------------------------------------------
//配置 ADC0 使用 AD0BUSY 作为转换源, 使用左对齐输出模式
//使用正常跟踪模式, 测量片内温度传感器输出
//禁止 ADC0 转换结束中断和 ADC0 窗口比较器中断
void ADC0_Init(void)
    {
    ADC0CN = 0x81;                  //ADC0 使能;正常跟踪模式
```

```
                                        //当写 AD0BUSY 时 ADC0 转换开始,ADC0 数据左对齐
    REF0CN = 0x07;                      //使能温度传感器片内 VREF 和 VREF 输出缓冲器
    AMX0SL = 0x0f;                      //选择温度传感器作为 ADC 多路模拟转换器输出
    ADC0CF = (SYSCLK/2500000) << 3;     //ADC 转换时钟 = 2.5MHz
    ADC0CF |= 0x01;                     //PGA 增益 = 2
    EIE2 &= ~0x02;                      //禁止 ADC0 EOC 中断
    EIE1 &= ~0x04;                      //禁止 ADC0 窗口比较器中断
    }
//-----------------------------------------------
```

(2) 中断法程序。

```
//此程序是 ADC0 应用例程在中断模式使用定时器 3 溢出作为转换开始信号,测量片内温度传感
//器输出
//ADC0 结果经简单的均值滤波处理,均值滤波计数值由常量 INT_DEC 给出
//ADC 结果经计算得出温度从 UART0 传输
//假设在 XTAL1 和 XTAL2 之间连接 22.1184MHz 晶体
//系统时钟频率存储在全局常量 SYSCLK,目标 UART 波特率存储在全局常量 BAUDRATE
//ADC0 采样率存储在全局常量 SAMPLERATE0
//目标器件: C8051F020
//链接工具: KEIL C51 6.03 / KEIL EVAL C51
//-----------------------------------------------
//包含文件
//-----------------------------------------------
#include <c8051f020.h>              //SFR 声明
#include <stdio.h>
//-----------------------------------------------
//C8051F02X 的 16 位 SFR 定义
//-----------------------------------------------
sfr16 DP = 0x82;                    //数据指针
sfr16 TMR3RL = 0x92;                //定时器 3 重装值
sfr16 TMR3 = 0x94;                  //定时器 3 计数器
sfr16 ADC0 = 0xbe;                  //ADC0 数据
sfr16 ADC0GT = 0xc4;                //ADC0 大于窗口
sfr16 ADC0LT = 0xc6;                //ADC0 小于窗口
sfr16 RCAP2 = 0xca;                 //定时器 2 捕捉/重装
sfr16 T2 = 0xcc;                    //定时器 2
sfr16 RCAP4 = 0xe4;                 //定时器 4 捕捉/重装
sfr16 T4 = 0xf4;                    //定时器 4
sfr16 DAC0 = 0xd2;                  //DAC0 数据
sfr16 DAC1 = 0xd5;                  //DAC1 数据
//-----------------------------------------------
//全局常量
//-----------------------------------------------
#define SYSCLK 22118400             //系统时钟频率(Hz)
#define BAUDRATE 9600               //UART 波特率(b/s)
#define SAMPLERATE0 50000           //ADC0 采样频率(Hz)
#define INT_DEC 256                 //均值滤波计数值
//-----------------------------------------------
//函数原型
//-----------------------------------------------
```

```
void SYSCLK_Init(void);
void PORT_Init(void);
void UART0_Init(void);
void ADC0_Init(void);
void Timer3_Init(int counts);
void ADC0_ISR(void);
//-----------------------------------------
//全局变量
//-----------------------------------------
long result;                          //放置经数字滤波后的结果
//-----------------------------------------
//主程序
//-----------------------------------------
void main(void) {
long temperature;                     //精度为百分之一的温度(℃)
int temp_int, temp_frac;              //温度的整数和小数部分
WDTCN = 0xde;                         //禁止看门狗定时器
WDTCN = 0xad;
SYSCLK_Init();                        //初始化振荡器
PORT_Init();                          //初始化数据交叉开关和通用 I/O 端口
UART0_Init();                         //初始化 UART0
Timer3_Init(SYSCLK/SAMPLERATE0);      //初始化定时器 3 溢出为采样速率
ADC0_Init();                          //初始化 ADC
AD0EN = 1;                            //使能 ADC
EA = 1;                               //使能所有中断
while (1) {
    EA = 0;                           //禁止中断
    temperature = result;
    EA = 1;                           //重使能中断
    //计算温度百分之一精度
    temperature = temperature - 41758;
    temperature = (temperature * 100L) / 154;
    temp_int = temperature / 100;
    temp_frac = temperature - (temp_int * 100);
    printf ("Temperature is %+02d.%02d\n", temp_int, temp_frac);
}
}
//-----------------------------------------
//系统时钟初始化
//-----------------------------------------
//此程序初始化系统时钟使用 22.1184MHz 晶体作为系统时钟源
void SYSCLK_Init(void)
    {
    int i;                            //延时计数器
    OSCXCN = 0x67;                    //启动外部振荡器 22.1184MHz 晶体
    for (i=0; i<256; i++);            //等待振荡器启动(>1ms)
    while (!(OSCXCN & 0x80));         //等待晶体振荡器稳定
    OSCICN = 0x88;                    //选择外部振荡器作为系统时钟源并允许丢失时钟检测器
    }
//-----------------------------------------
//I/O 端口初始化
```

```
//-------------------------------------------
//配置数据交叉开关和通用 I/O 端口
void PORT_Init(void)
    {
    XBR0 = 0x04;                    //使能 UART0
    XBR1 = 0x00;
    XBR2 = 0x40;                    //使能数据交叉开关和弱上拉
    P0MDOUT |= 0x01;                //使能 TX0 推挽输出
    }
//-------------------------------------------
//UART0 初始化
//-------------------------------------------
//配置 UART0 使用定时器 1 作为波特率发生器
void UART0_Init(void)
    {
    SCON0 = 0x50;                   //SCON0: 模式 1, 8 位 UART, 使能 RX
    TMOD = 0x20;                    //TMOD: 定时器 1, 模式 2, 8 位重装
    TH1 = -(SYSCLK/BAUDRATE/16);    //按波特率设置 T1 重装值
    TR1 = 1;                        //启动定时器 1
    CKCON |= 0x10;                  //定时器 1 使用系统时钟作为时基
    PCON |= 0x80;                   //SMOD00 = 1
    TI0 = 1;                        //表示 TX0 就绪
    }
//-------------------------------------------
//ADC0 初始化
//-------------------------------------------
//配置 ADC0 使用定时器 3 溢出作为转换源, 转换结束产生中断
//使用左对齐输出模式允许 ADC 转换结束中断不使用时禁止 ADC
void ADC0_Init(void)
    {
    ADC0CN = 0x05;                  //ADC0 禁止; 正常跟踪 mode; 定时器 3 溢出 ADC0 转换开始
                                    //ADC0 数据是左对齐
    REF0CN = 0x07;                  //允许温度传感器片内 VREF 和 VREF 输出缓冲器
    AMX0SL = 0x0f;                  //选择温度传感器作为 ADC 多路模拟转换输出
    ADC0CF = (SYSCLK/2500000) << 3; //ADC 转换时钟 = 2.5MHz
    ADC0CF |= 0x01;                 //PGA 增益 = 2
    EIE2 |= 0x02;                   //允许 ADC 中断
    }
//-------------------------------------------
//定时器 3 初始化
//-------------------------------------------
//配置定时器 3,自动重装间隔由 counts 指定,不产生中断,使用系统时钟为时基
void Timer3_Init(int counts)
    {
    TMR3CN = 0x02;                  //停止定时器 3; 清除 TF3;
                                    //使用系统时钟为时基
    TMR3RL = -counts;               //初始化重装值
    TMR3 = 0xffff;                  //设置为立即重装
    EIE2 &= ~0x01;                  //禁止定时器 3 中断
    TMR3CN |= 0x04;                 //启动定时器 3
}
```

```
//-----------------------------------------------
//ADC0 中断服务程序
//-----------------------------------------------
//得到 ADC0 采样值，将它加到运行总数<accumulator>中
//数字滤波计数器 <int_dec>减 1,当<int_dec>为 0 时，在全局变量<result>放置经数字滤波后
//的结果
void ADC0_ISR(void) interrupt 15
    {
    static unsigned int_dec = INT_DEC; //数字滤波计数器
                                       //当 int_dec = 0 时重设新值
    static long accumulator = 0L;
    AD0INT = 0;                        //清除 ADC 转换结束标志
    accumulator += ADC0;               //读 ADC 值并加到运行总数中
    int_dec--;                         //更新数字滤波计数器
    if (int_dec == 0) {                //如果为 0 记入结果
    int_dec = INT_DEC;                 //重设计数器
    result = accumulator >> 8;         //除以 256,求平均值(数字滤波)
    accumulator = 0L;                  //复位 accumulator
    }
    }
```

2. 多通道数据采集

```
//此程序为 ADC0 的应用例程在中断模式使用定时器 3 溢出作为开始转换信号
//测量 AIN0 到 AIN7 的电压和温度传感器
//转换结果经过计算所得电压从 UART0 传输
//假设在 XTAL1 和 XTAL2 之间接 22.1184MHz 晶体
//系统时钟频率存储在全局常量 SYSCLK,目标 UART 波特率存储在全局常量 BAUDRATE
//ADC0 采样频率存储在全局常量 SAMPLERATE0,电压参考值存储在 VREF0
//目标器件: C8051F020
//链接工具: KEIL C51 6.03 / KEIL EVAL C51
//-----------------------------------------------
//包含文件
//-----------------------------------------------
#include <c8051f020.h>             //SFR 声明
#include <stdio.h>
//-----------------------------------------------
//C8051F02X 的 16 位 SFR 定义
//-----------------------------------------------
sfr16 DP = 0x82;                   //数据指针
sfr16 TMR3RL = 0x92;               //定时器 3 重装值
sfr16 TMR3 = 0x94;                 //定时器 3 计数器
sfr16 ADC0 = 0xbe;                 //ADC0 数据
sfr16 ADC0GT = 0xc4;               //ADC0 大于窗口
sfr16 ADC0LT = 0xc6;               //ADC0 小于窗口
sfr16 RCAP2 = 0xca;                //定时器 2 捕捉/重装
sfr16 T2 = 0xcc;                   //定时器 2
sfr16 RCAP4 = 0xe4;                //定时器 4 捕捉/重装
sfr16 T4 = 0xf4;                   //定时器 4
sfr16 DAC0 = 0xd2;                 //DAC0 数据
sfr16 DAC1 = 0xd5;                 //DAC1 数据
//-----------------------------------------------
//全局常量
```

```
//-------------------------------------------
#define SYSCLK 22118400            //系统时钟频率(Hz)
#define BAUDRATE 9600              //UART 波特率(b/s)
#define SAMPLERATE0 50000          //ADC0 采样频率(Hz)
#define VREF0 2430                 //VREF 参考电平(mV)
//-------------------------------------------
//函数原型
//-------------------------------------------
void SYSCLK_Init(void);
void PORT_Init(void);
void UART0_Init(void);
void ADC0_Init(void);
void Timer3_Init(int counts);
void ADC0_ISR(void);
//-------------------------------------------
//全局变量
//-------------------------------------------
long result[9];                    //AIN0 - 7 和温度传感器输出结果
//-------------------------------------------
//主程序
//-------------------------------------------
void main(void) {
long voltage;                      //电压以 mV 为单位
int i;                             //循环计数器
WDTCN = 0xde;                      //禁止看门狗定时器
WDTCN = 0xad;
SYSCLK_Init();                     //初始化振荡器
PORT_Init();                       //初始化数据交叉开关和通用 I/O
UART0_Init();                      //初始化 UART0
Timer3_Init(SYSCLK/SAMPLERATE0);   //初始化定时器 3 溢出作为采样率
ADC0_Init();                       //初始化 ADC
AD0EN = 1;                         //允许 ADC
EA = 1;                            //允许所有中断
while (1) {
for (i = 0; i < 9; i++) {
EA = 0;                            //禁止中断
voltage = result[i];               //从全局变量取得 ADC 值
EA = 1;                            //重新使能中断
                                   //计算电压(mV)
voltage = voltage * VREF0;
voltage = voltage >> 16;
printf("Channel '%d' voltage is %ldmV\n", i, voltage);
}
}
}
//-------------------------------------------
//系统时钟初始化
//-------------------------------------------
//此程序初始化系统时钟使用 22.1184MHz 晶体作为系统时钟
void SYSCLK_Init(void)
{
    int i;                         //延时计数器
    OSCXCN = 0x67;                 //启动外部振荡器 22.1184MHz 晶体
    for (i = 0; i < 256; i++);     //等待振荡器启动 (> 1ms)
```

```
    while (!(OSCXCN & 0x80));          //等待晶体振荡器稳定
    OSCICN = 0x88;                     //选择外部振荡器作为系统时钟源并允许丢失时钟检测器
    }
//-----------------------------------------------
//I/O 端口初始化
//-----------------------------------------------
//配置数据交叉开关和通用 I/O 端口
void PORT_Init(void)
    {
    XBR0 = 0x04;                       //使能 UART0
    XBR1 = 0x00;
    XBR2 = 0x40;                       //使能数据交叉开关和弱上拉

    }
//-----------------------------------------------
//UART0 初始化
//-----------------------------------------------
//配置 UART0 使用定时 1 为波特率发生器
void UART0_Init(void)
    {
    SCON0 = 0x50;                      //SCON0: 模式 1, 8 位 UART, 允许 RX
    TMOD = 0x20;                       //TMOD: 定时器 1, 模式 2, 8 位重装
    TH1 = -(SYSCLK/BAUDRATE/16);       //按波特率设置定时器 1 重装值
    TR1 = 1;                           //启动定时器 1
    CKCON |= 0x10;                     //定时器 1 使用系统时钟为时基
    PCON |= 0x80;                      //SMOD00 = 1
    TI0 = 1;                           //表示 TX0 就绪
    }
//-----------------------------------------------
//ADC0 初始化
//-----------------------------------------------
//配置 ADC0 使用定时器 3 溢出作为转换源, 转换结束产生中断使用左对齐输出模式
//使能 ADC 转换结束中断禁止 ADC
//注意: 使能低功率跟踪模式保证当改变通道时的跟踪次数最少
void ADC0_Init(void)
    {
    ADC0CN = 0x45;                     //ADC0 禁止; 低功率跟踪模式
                                       //当定时器 3 溢出时 ADC0 转换开始; ADC0 数据左对齐
    REF0CN = 0x07;                     //使能温度传感器片内 VREF 和 VREF 输出缓冲器
    AMX0SL = 0x00;                     //选择 AIN0 为 ADC 多路模拟输出
    ADC0CF = (SYSCLK/2500000) << 3;    //ADC 转换时钟 = 2.5MHz
    ADC0CF &= ~0x07;                   //PGA 增益 = 1
    EIE2 |= 0x02;                      //允许 ADC 中断
    }
//-----------------------------------------------
//定时器 3 初始化
//-----------------------------------------------
//配置定时器 3 自动重装载时间间隔由<counts>指定(不产生中断)
//使用系统时钟作为时基
void Timer3_Init(int counts)
    {
    TMR3CN = 0x02;                     //停止定时器 3; 清除 TF3;
                                       //使用系统时钟作为时基
    TMR3RL = -counts;                  //初始化重装值
```

```
    TMR3 = 0xffff;                  //设置为立即重装
    EIE2 &= ~0x01;                  //禁止定时器 3 中断
    TMR3CN |= 0x04;                 //启动定时器 3
    }
//-----------------------------------------------
//ADC0 中断服务程序
//-----------------------------------------------
//ADC0 转换结束中断服务程序
//读取 ADC0 采样值并存储在全局数组 <result>
//同时选择下一个通道转换
void ADC0_ISR(void) interrupt 15
    {
    static unsigned char channel = 0; //ADC 多路模拟通道(0-8)
    AD0INT = 0;                     //清除 ADC 转换结束标志
    result[channel] = ADC0;         //读 ADC 值
    channel++;                      //改变通道
    if (channel == 9) {
    channel = 0;
    }
    AMX0SL = channel;               //设置多路模拟转换器到下一个通道
}
```

5.2 数模转换器

5.2.1 数模转换原理及性能指标

1. 转换原理

D/A 转换器的原理很简单,可以总结为“按权展开,然后相加”几个字。即将要转换的数字量中每一位都按其权值分别转换为模拟量,并通过运算放大器求和相加,因此 D/A 转换器内部必须有一个解码网络,以实现按权值分别进行 D/A 转换。

解码网络通常有两种:二进制加权电阻网络和 T 型电阻网络。在二进制加权电阻网络中,每位二进制位的 D/A 转换是通过相应位加权电阻实现的,这必然导致加权电阻阻值差别极大,尤其在 D/A 转换器位数较大时更不能容忍。例如,若某 D/A 转换器有 12 位,则最高位加权电阻为 10kΩ 时的最低位加权电阻应当是 $10k\Omega\times2^{11}=20M\Omega$。这么大的电阻在 VLSI 技术中很难制造出来,即便制造出来,其精度也很难符合要求。因此现代 D/A 转换器几乎毫不例外地采用 T 型电阻网络进行解码活动。

为了说明原理,现以 4 位 D/A 转换器为例介绍,它的原理框图如图 5-14 所示。在图中的虚线框内是 T 型电阻网络(桥上电阻为 R,桥臂电阻为 $2R$);OA 为运算放大器,A 点为虚拟地(接近 0V);V_{REF} 为参考电压,由稳压电源提供;$S_3\sim S_0$ 为电子开关,受 4 位 DAC 寄存器中 $b_3b_2b_1b_0$ 的控制。为了分析问题,设 $b_3b_2b_1b_0$ 全为 1,故 $S_3S_2S_1S_0$ 全部和 1 端相连。由于 A 点为虚地,B 点到 A 点和 B 点到地线的电阻一样都为 $2R$,即 $I_0=I_{L0}$,根据基尔霍夫电流定律:$I_{L1}=I_{L0}+I_0$,分析 C 点到 A 点的电阻及到地线的电阻可得 $I_1=I_{L1}$,则 $I_0=1/2I_1$,同理可推得 $I_1=1/2I_2$、$I_2=1/2I_3$,所以可得如下关系:

$$I_3=\frac{V_{REF}}{2R}=2^3\times\frac{V_{REF}}{2^4\times R}$$

$$I_2=\frac{I_3}{2}=2^2\times\frac{V_{REF}}{2^4\times R}$$

$$I_1=\frac{I_2}{2}=2^1\times\frac{V_{REF}}{2^4\times R}$$

$$I_0=\frac{I_1}{2}=2^0\times\frac{V_{REF}}{2^4\times R}$$

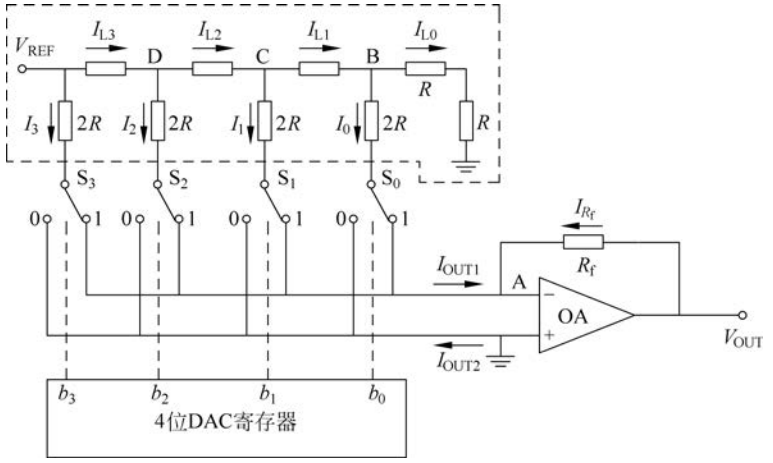


图 5-14　T 型电阻网络 D/A 转换原理框图

事实上，S_3～S_0 的状态是受 $b_3b_2b_1b_0$ 控制的，并不一定是全 1，所以流入 A 点的电流应该是：

$$I_{OUT1}=b_3I_3+b_2I_2+b_1I_1+b_0I_0=(b_32^3+b_22^2+b_12^1+b_02^0)\frac{V_{REF}}{2^4R}\qquad(5\text{-}4)$$

选取 $R_f=R$，并考虑 A 点为虚地，则有 $I_{R_f}=-I_{out1}$。

因此，可以得到式(5-5)：

$$V_{OUT}=I_{R_f}R_f=-(b_32^3+b_22^2+b_12^1+b_02^0)\frac{V_{REF}}{2^4R}R_f=-B\frac{V_{REF}}{16}\qquad(5\text{-}5)$$

对于 n 位 T 型电阻网络，式(5-5)可变为：

$$V_{OUT}=-(b_{n-1}2^{n-1}+b_{n-2}2^{n-2}+\cdots+b_12^1+b_02^0)\frac{V_{REF}}{2^nR}R_f=-B\frac{V_{REF}}{2^n}\qquad(5\text{-}6)$$

式中 B 为一个二进制数，T 型电阻网络的 D/A 转换输出电压量绝对值与该二进制数的大小成正比。

2. 性能指标

D/A 性能指标是衡量芯片质量的重要参数，主要的性能指标有 4 条。

1) 分辨率

分辨率(Resolution)是指 D/A 转换器能分辨的最小输出模拟增量，取决于输入数字量的二进制位数。一个 n 位的 D/A 转换器所能分辨的最小电压增量定义为满量程值的 2^{-n} 倍。例如，满量程为 10V 的 8 位 D/A 芯片的分辨率为 $10V\times2^{-8}=39mV$；而 16 位的 D/A 芯片的分辨率为 $10V\times2^{-16}=153\mu V$。

2) 转换精度

转换精度(Conversion Accuracy)与分辨率是两个不同的概念。转换精度是指满量程时

D/A 的实际模拟输出值和理论值的接近程度。对 T 型电阻网络的 D/A 转换器,其转换精度与参考电压 V_{REF}、电阻值和电子开关的误差有关。例如,满量程时理论输出值为 10V,实际输出值是在 9.99 到 10.01 之间,则其转换精度为±10mV。通常 D/A 转换器的转换精度为分辨率的一半,即为 LSB/2。LSB(Least Significant Bit)是最低有效位,指最低 1 位数字变化引起输出电压幅度的变化量。

3) 偏移量误差

偏移量误差(Offset Error)是指输入数字量为零时,输出模拟量对零的偏移值。这种误差通常可以通过 D/A 转换器的外接 V_{REF} 和电位器加以调整。

4) 线性度

线性度(Linearity)是指 D/A 转换器的实际转换特性曲线和理想直线之间的最大偏差。通常线性度不应超出±1/2LSB。

除此以外,指标还有转换速度、温度灵敏度等,通常这些参数都很小,一般不予考虑。

5.2.2 C8051F020 的 DAC 功能

C8051F020 单片机有两个片内 12 位电压方式 DAC。每个 DAC 的输出摆幅均为 0V 到(VREF-1LSB),对应的输入码范围是 0x000～0xFFF。可以用对应的控制寄存器 DAC0CN 和 DAC1CN 使能/禁止 DAC0 和 DAC1。在被禁止时,DAC 的输出保持在高阻状态,DAC 的供电电流降到 1μA 或更小。DAC 的功能框图如图 5-15 所示。每个 DAC 的电压基准在 VREFD 引脚提供。如果使用内部电压基准,为了使 DAC 输出有效,该基准必须被使能。有关配置 DAC 电压基准的详细信息将在 5.3 节介绍。

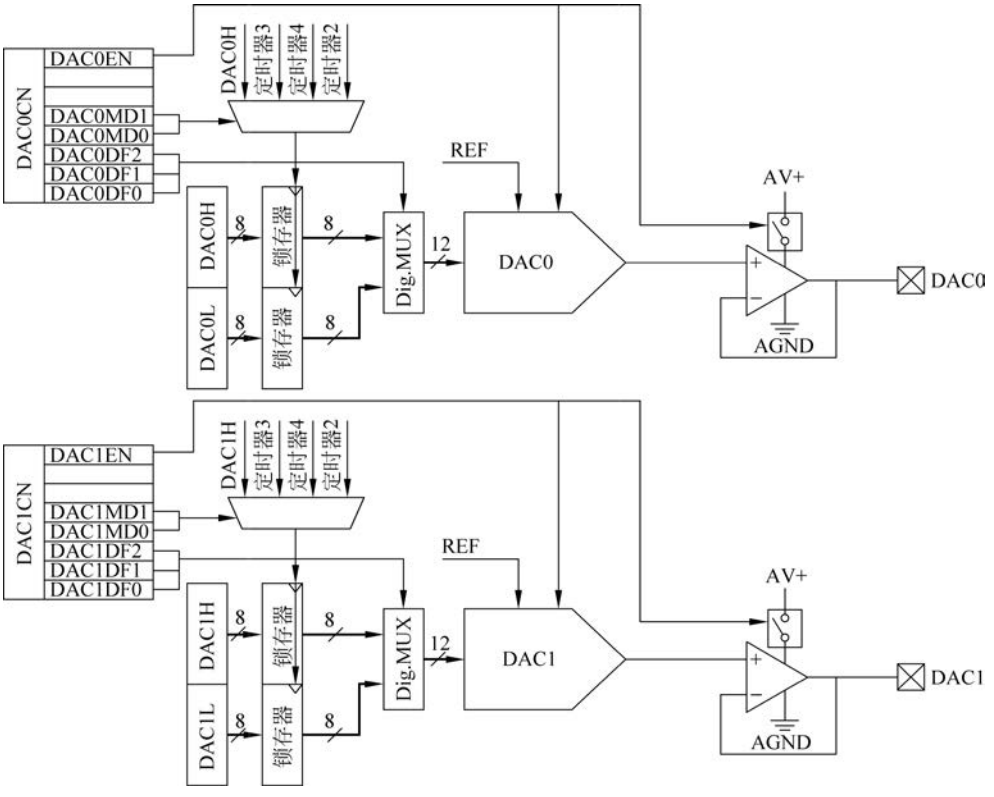


图 5-15 DAC 功能框图

控制 DAC 工作的主要是控制寄存器 DAC0CN 和 DAC1CN，两个 SFR 分别控制 DAC0 和 DAC1，以 DAC0CN 为例来说明。控制寄存器 DAC0CN 的格式为：

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	复位值
DAC0EN	—	—	DAC0MD1	DAC0MD0	DAC0DF2	DAC0DF1	DAC0DF0	00000000
位 7	位 6	位 5	位 4	位 3	位 2	位 1	位 0	SFR地址：0xD4

其中，各位的含义如下：

位 7(DAC0EN)——DAC0 使能位。

0：DAC0 禁止。DAC0 输出引脚被禁止，DAC0 处于低功耗关断方式。

1：DAC0 使能。DAC0 正常输出；DAC0 处于工作状态。

位 6 和位 5——未用。读＝0000b；写＝忽略。

位 4 和位 3(DAC0MD1－0)——DAC0 方式位。

00：DAC 输出更新发生在写 DAC0H 时。

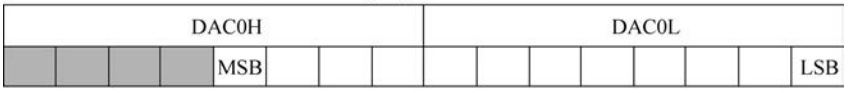
01：DAC 输出更新发生在定时器 3 溢出时。

10：DAC 输出更新发生在定时器 4 溢出时。

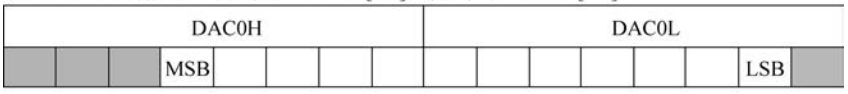
11：DAC 输出更新发生在定时器 2 溢出时。

位 2～0(DAC0DF2～0)：DAC0 数据格式位。

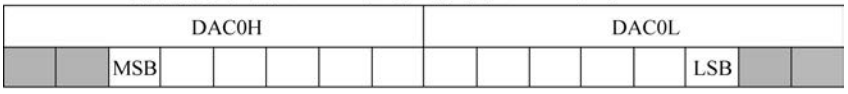
000：DAC0 数据字的高4 位在DAC0H[3:0]，低字节在DAC0L。



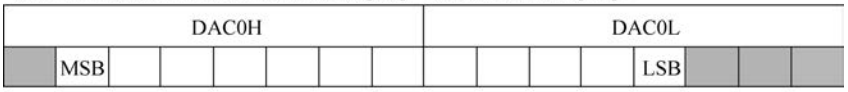
001：DAC0 数据字的高5 位在DAC0H[4:0]，低7 位在DAC0L[7:1]。



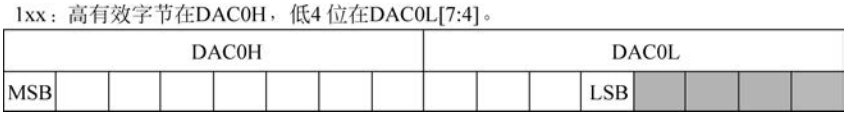
010：DAC0 数据字的高6 位在DAC0H[5:0]，低6 位在DAC0L[7:2]。



011：DAC0 数据字的高7 位在DAC0H[6:0]，低5 位在DAC0L[7:3]。



1xx：高有效字节在DAC0H，低4 位在DAC0L[7:4]。



5.2.3 DAC 输出更新

每个 DAC 都具有灵活的输出更新机制，允许全量程内平滑变化并支持无抖动输出更

新,适合于波形发生器应用。下面的描述都是以 DAC0 为例,DAC1 的操作与 DAC0 完全相同。注意,读 DAC0L 返回预锁存数据,所读值是最后写入该寄存器中的数据,而不是 DAC0L 锁存器中的值。但读 DAC0H 总是返回 DAC0H 锁存器中的值。

1. 根据软件命令更新输出

在默认方式下(DAC0CN.[4:3]=00),DAC0 的输出在写 DAC0 数据寄存器高字节(DAC0H)时更新。注意,写 DAC0L 时数据被保持,对 DAC0 输出没有影响,直到对 DAC0H 的写操作发生。如果向 DAC 数据寄存器写入一个 12 位字,则 12 位的数据字被写到低字节(DAC0L)和高字节(DAC0H)数据寄存器。在写 DAC0H 寄存器后数据被锁存到 DAC0。因此,如果需要 12 位分辨率,应在写入 DAC0L 之后写 DAC0H。DAC 可被用于 8 位方式,这种情况是将 DAC0L 初始化一个所希望的数值(通常为 0x00),将数据只写入 DAC0H。

2. 基于定时器溢出的输出更新

在前面介绍的 ADC 转换操作中,ADC 转换可以由定时器溢出启动,不用处理器干预。与之类似,DAC 的输出更新也可以用定时器溢出事件触发。这一特点在用 DAC 产生一个固定采样频率的波形时尤其有用,可以消除中断响应时间不同和指令执行时间不同对 DAC 输出时序的影响。当 DAC0MD 位(DAC0CN.[4:3])被设置为 01、10 或 11 时(分别为定时器 3、定时器 4 或定时器 2),对 DAC 数据寄存器的写操作被保持,直到相应的定时器溢出事件发生时 DAC0H:DAC0L 的内容才被复制到 DAC 输入锁存器,允许 DAC 数据改变为新值。

5.2.4 DAC 输出定标/调整

在某些情况下,对 DAC0 进行写入操作之前应对输入数据移位,以正确调整 DAC 输入寄存器中的数据。这种操作一般需要一个或多个装入和移位指令,因而增加软件开销和降低 DAC 的数据通过率。为了减少这方面的负担,数据格式化功能为用户提供了一种能对数据寄存器 DAC0H 和 DAC0L 中的数据格式编程的手段。3 个 DAC0DF 位(DAC0CN.[2:0])允许用户在 5 种数据字格式指定一种,具体见 DAC0CN 寄存器定义。

DAC1 的功能与上述 DAC0 的功能完全相同。

5.2.5 数模转换举例

D/A 转换器的编程相对 A/D 转换器要简单,按照要求设置好输出更新的条件,将要转换的数值量送到 DAC 数据寄存器就行。下面是产生锯齿波和阶梯波的示例。将 DAC0 设置成输出更新发生在写 DAC0H 时,即直接更新。DAC1 设置成输出更新发生在定时器 2 溢出时。

1. 产生阶梯波

DAC0 用程序更新输出,产生一个阶梯波形。

```
//-----------------------------------------------
//DAC 转换程序
//-----------------------------------------------
```

```
//-----------------------------------------------
//INCLUDES
//-----------------------------------------------
#include <C8051F020.h>                  //寄存器定义文件
//C8051F02X的16位SFR定义
//-----------------------------------------------
sfr16 DAC0 = 0xd2;                      //DAC0数据寄存器
//-----------------------------------------------
#define UP 0x010
#define T 1000
void d1ms(int count);                   //延时程序
void config(void);                      //配置程序
void main(void)
{
    int i;
    config();
    for(i=0;i<=4095;i+UP)               //形成阶梯波形
        {
        DAC0 = i;                       //送数字量到DAC0直接更新输出
        d1ms(T);
        }
}
void d1ms(int count)
{   int j;
    while(count--!=0)
    {
        for(j=0;j<100;j++);
    }
}
//-----------------------------------------------
//配置程序
//-----------------------------------------------
void config(void) {
    //Local Variable Definitions
    int n = 0;

    WDTCN = 0x07;                       //看门狗控制寄存器
    WDTCN = 0xDE;                       //禁止看门狗定时器
    WDTCN = 0xAD;
//-----------------------------------------------
//Oscillator Configuration
//-----------------------------------------------
    OSCXCN = 0x67;                      //外部振荡器寄存器,采用11.0952MHz
    for (n = 0; n < 255; n++);          //等待振荡器启动
    while ((OSCXCN & 0x80) == 0);       //等待晶振稳定
//-----------------------------------------------
//Reference Control Register Configuration
//-----------------------------------------------
    REF0CN = 0x02;                      //内部偏压发生器工作
//-----------------------------------------------
//-----------------------------------------------
```

```
//DAC Configuration
//-----------------------------------------------
    DAC0CN = 0x80;                    //允许 DAC0,程序直接更新输出,数据右对齐
    DAC0L = 0x00;                     //DAC1 数据寄存器初值
    DAC0H = 0x00;
//-----------------------------------------------
}
```

2. 产生锯齿波

用 DAC1 产生锯齿波,T2 定时中断更新输出。

```
//-----------------------------------------------
//DAC 转换程序
//-----------------------------------------------
//-----------------------------------------------
//INCLUDES
//-----------------------------------------------
#include <C8051F020.h>                //寄存器定义文件
//C8051F02X 的 16 位 SFR 定义
//-----------------------------------------------
sfr16 T2 = 0xcc;                      //定时器 2
sfr16 DAC1 = 0xd5;                    //DAC1 数据寄存器
//-----------------------------------------------
void T2_ISR();                        //T2 中断服务程序
void config(void);                    //配置系统
void main(void)
{
config();                             //配置
EA = 1;                               //开中断
while(1);
}
void T2_ISR() interrupt 5
{
TF2 = 0;                              //清中断标志
DAC1++;                               //因为是 T2 溢出更新 DAC1 输出
                                      //所以可以对 SFR16 操作,此时并不立即更新
if(DAC1 >= 0x1000)
    DAC1 = 0;                         //形成锯齿波
}
//-----------------------------------------------
//Config Routine
//-----------------------------------------------
void config(void) {

//Local Variable Definitions
    int n = 0;
    WDTCN = 0x07;                     //看门狗控制寄存器
    WDTCN = 0xDE;                     //禁止看门狗定时器
    WDTCN = 0xAD;
//-----------------------------------------------
//Oscillator Configuration
```

```
//-----------------------------------------------
OSCXCN = 0x67;                          //外部振荡器寄存器,采用 11.0952MHz
    for (n = 0; n < 255; n++);          //等待振荡器启动
    while ((OSCXCN & 0x80) == 0);       //等待晶振稳定
//-----------------------------------------------
//Reference Control Register Configuration
//-----------------------------------------------
REF0CN = 0x02;                          //内部偏压发生器工作
//-----------------------------------------------
//-----------------------------------------------
//DAC Configuration
//-----------------------------------------------
DAC1CN = 0x98;                          //允许 DAC1,T2 溢出中断更新输出,数据右对齐
DAC1L = 0x00;                           //DAC1 数据寄存器初值
DAC1H = 0x00;
//-----------------------------------------------
//-----------------------------------------------
//Timer Configuration
//-----------------------------------------------
    RCAP2H = 0x05;                      //重新装入的时间常数
    RCAP2L = 0x00;
    TH2 = 0x05;                         //初始值
    TL2 = 0x00;
    T2CON = 0x04;                       //启动 T2
//-----------------------------------------------
//-----------------------------------------------
//Interrupt Configuration
//-----------------------------------------------
    IE = 0x20;                          //T2 中断允许
}
```

5.3 电压基准

电压基准电路为控制 ADC 和 DAC 模块工作提供了灵活性。有 3 个电压基准输入引脚,允许两个 ADC 和两个 DAC 使用外部电压基准或片内电压基准输出。通过配置 V_{REF} 模拟开关,ADC0 还可以使用 DAC0 的输出作为内部基准,ADC1 可以使用模拟电源电压作为基准,电压基准的功能框图如图 5-16 所示。内部电压基准电路由一个 1.2V、15ppm/℃(典型值)的带隙电压基准发生器和一个两倍增益的输出缓冲放大器组成。内部基准电压可以通过 V_{REF} 引脚连到应用系统中的外部器件或图 5-16 所示的电压基准输入引脚。建议在 V_{REF} 引脚与 AGND 之间接入 0.1μF 和 4.7μF 的旁路电容。电压基准的设置由基准电压控制寄存器 REF0CN 来完成,它可使能/禁止内部基准发生器和选择 ADC0、ADC1 的基准输入。REF0CN 中的 BIASE 位使能片内电压基准发生器,而 REFBE 位使能驱动 V_{REF} 引脚的缓冲放大器。当被禁止时,带隙基准和缓冲放大器消耗的电流小于 1μA(典型值),缓冲放大器的输出进入高阻状态。如果要使用内部带隙基准作为基准电压发生器,则 BIASE 和 REFBE 位必须被置 1。如果不使用内部基准,则 REFBE 位可以被清 0。

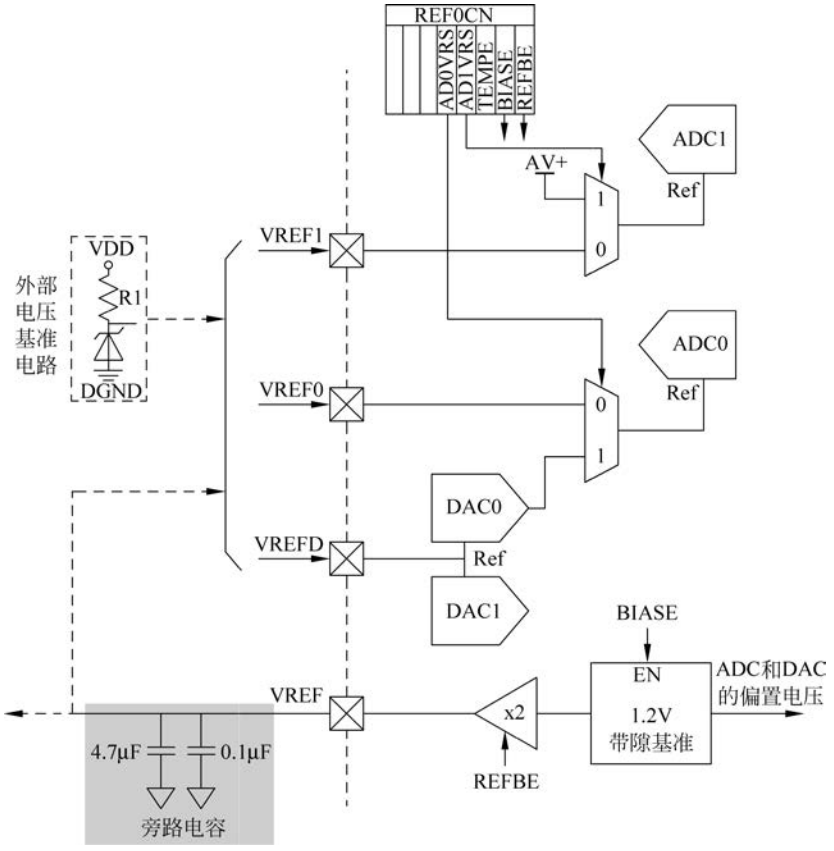


图 5-16　电压基准的功能框图

注意：如果使用 ADC 或 DAC，则不管电压基准取自片内还是片外，BIASE 位必须被置为逻辑 1。如果既不使用 ADC 也不使用 DAC，则这两位都应被清 0 以节省功耗。AD0VRS 和 AD1VRS 位分别用于选择 ADC0 和 ADC1 的电压基准源。基准电压控制寄存器 REF0CN 的格式如下：

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	复位值
—	—	—	AD0VRS	AD1VRS	TEMPE	BIASE	REFBE	00000000
位 7	位 6	位 5	位 4	位 3	位 2	位 1	位 0	SFR地址：0xD1

其中，各位的含义如下：

位 7～5——未用。读＝000b，写＝忽略。

位 4(AD0VRS)——ADC0 电压基准选择位。

0：ADC0 电压基准取自 VREF0 引脚。

1：ADC0 电压基准取自 DAC0 输出。

位 3(AD1VRS)——ADC1 电压基准选择位。

0：ADC1 电压基准取自 VREF1 引脚。

1：ADC1 电压基准取自 AV＋。

位 2(TEMPE)——温度传感器使能位。

0：内部温度传感器关闭。

1：内部温度传感器工作。

位 1(BIASE)——ADC/DAC 偏压发生器使能位(使用 ADC 和 DAC 时该位必须为 1)。

0：内部偏压发生器关闭。

1：内部偏压发生器工作。

位 0(REFBE)——内部电压基准缓冲器使能位。

0：内部电压基准缓冲器关闭。

1：内部电压基准缓冲器工作。内部电压基准提供从 V_{REF} 引脚输出。

电压基准的电气特性可参见附录 C。温度传感器接在 ADC0 输入多路开关的最后一个输入端，REF0CN 中的 TEMPE 位用于使能和禁止温度传感器。当被禁止时，温度传感器为默认的高阻状态，此时对温度传感器的任何 A/D 测量结果都是无意义的。

5.4 比 较 器

C8051F020 单片机内部有两个比较器，原理图及功能框图分别如图 5-17 和图 5-18 所示。CIP-51 内核与每个比较器之间的数据和控制接口都通过特殊功能寄存器实现，它可以将任何一个比较器置于低功耗关断方式。

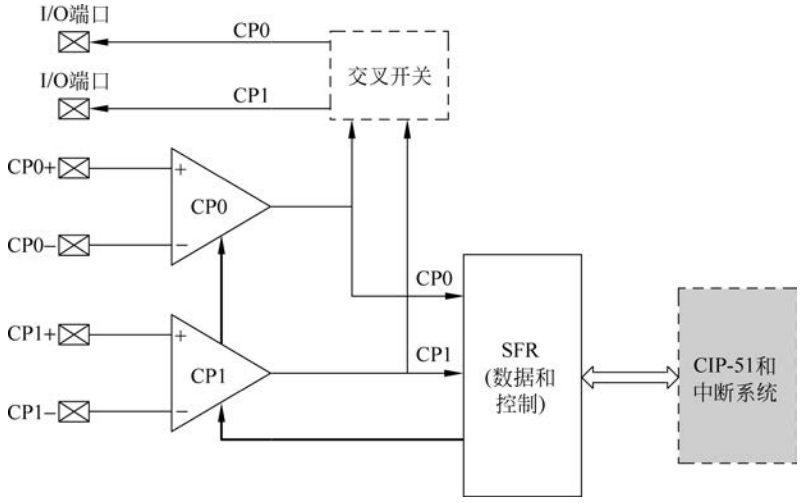


图 5-17　比较器原理框图

每个比较器都有两个输入引脚，可以承受－0.25V～(AV＋)＋0.25V 的外部驱动电压而不至损坏或发生工作错误。比较器的输出都可以经 I/O 交叉开关连到外部 I/O 引脚上。当被分配了外部引脚时，每个比较器的输出都可以被编程工作在漏极开路或者推挽方式，关于交叉开关和端口初始化的详细信息，请参见 2.4 节。

比较器的回差电压可以用软件通过比较器的控制寄存器进行编程，并且每个比较器都可以在上升沿、下降沿或在两个边沿产生中断。这些中断能将 CIP-51 内核从休眠方式唤醒，而比较器的输出状态可以用软件进行查询。两个比较器原理和使用方法类似，下面主要以比较器 0 为例介绍。

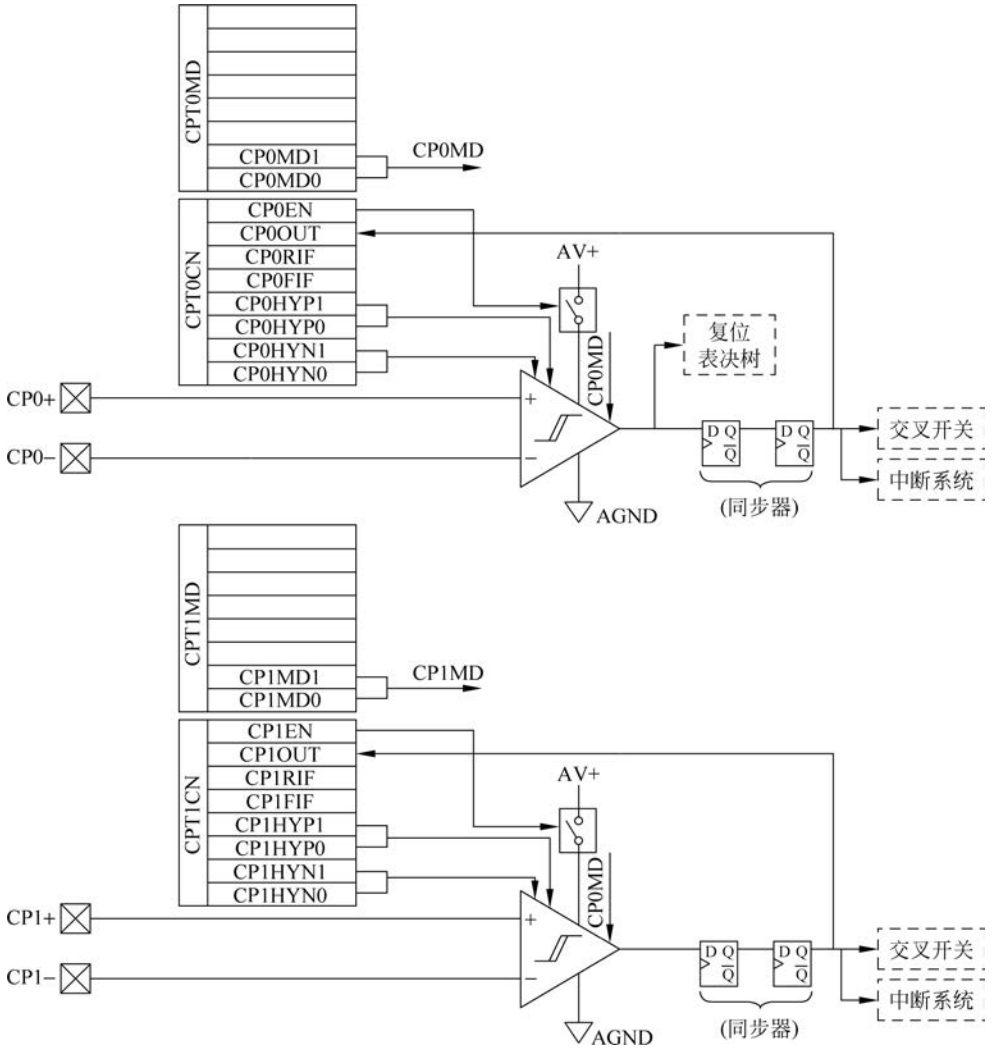


图 5-18　比较器功能框图

比较器 0 的回差电压编程的方法是通过程序修改对应的比较器 0 控制寄存器 CPT0CN 的位 3～0：负向回差电压值由 CP0HYN 位的设置决定，正向回差电压值由 CP0HYP 位决定。用户既可以选择对回差电压值(指输入电压)编程，也可以选择对门限电压两侧的正向和负向回差对称度编程。比较器回差电压曲线如图 5-19 所示，有关回差电压指标见附录 C 中的比较器电气特性。比较器 0 控制寄存器 CPT0CN 各位的定义如下：

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	复位值
CP0EN	CP0OUT	CP0RIF	CP0FIF	CP0HYP1	CP0HYP0	CP0HYN1	CP0HYN0	00000000
位 7	位 6	位 5	位 4	位 3	位 2	位 1	位 0	SFR地址：0x9E

其中，各位的含义如下：

位 7(CP0EN)——比较器 0 使能位。

0：比较器 0 禁止。

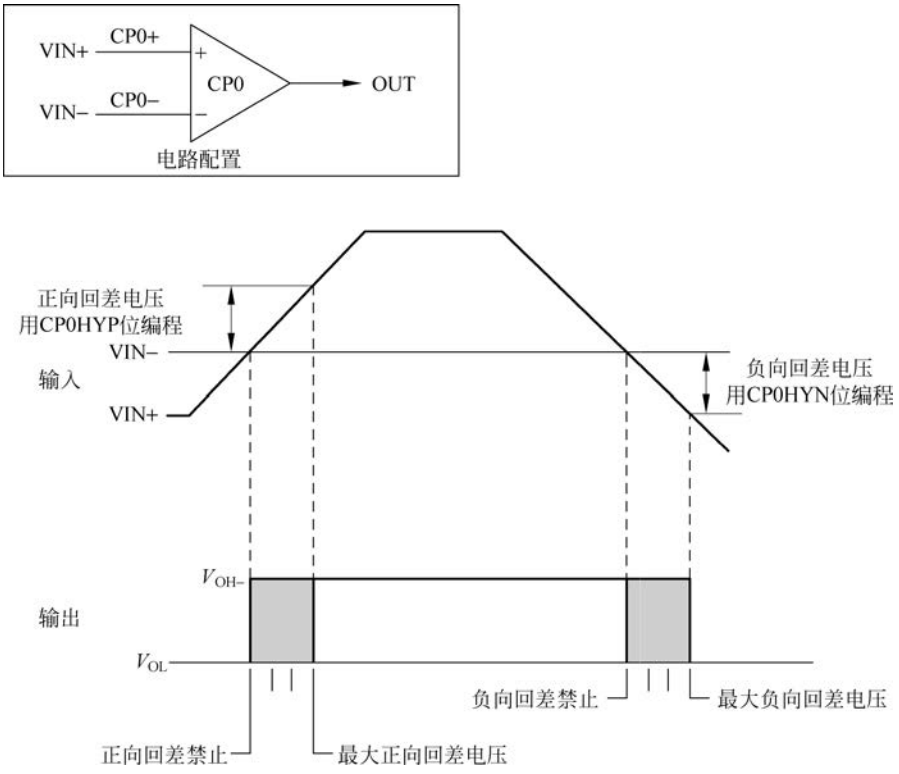


图 5-19　比较器回差电压曲线

1：比较器 0 使能。

位 6(CP0OUT)——比较器 0 输出状态标志。

0：电压值 CP0＋ < CP0－。

1：电压值 CP0＋ > CP0－。

位 5(CP0RIF)——比较器 0 上升沿中断标志。

0：自该标志位被清除后，没有发生过比较器 0 上升沿中断。

1：自该标志位被清除后，发生了比较器 0 上升沿中断。

位 4(CP0FIF)——比较器 0 下降沿中断标志。

0：自该标志位被清除后，没有发生过比较器 0 下降沿中断。

1：自该标志位被清除后，发生了比较器 0 下降沿中断。

位 3 和位 2(CP0HYP1－0)——比较器 0 正向回差电压控制位。

00：禁止正向回差电压。

01：正向回差电压＝2mV。

10：正向回差电压＝4mV。

11：正向回差电压＝10mV。

位 1 和位 0(CP0HYN1－0)——比较器 0 负向回差电压控制位。

00：禁止负向回差电压。

01：负向回差电压＝2mV。

10：负向回差电压＝4mV。

11：负向回差电压＝10mV。

比较器的输出可以采用软件查询，也可以作为中断源来触发中断。在比较器输出的上升沿和/或下将沿都可以产生中断(有关中断允许和优先级控制的内容见 2.3 节)。比较器 0 的下降沿中断置 1CP0FIF 标志，比较器 0 的上升沿中断置 1CP0RIF 标志。这些位一旦被置 1，将一直保持 1 状态直到被软件清除，可以在任意时刻通过读取 CP0OUT 位得到比较器 0 的输出状态。注意，在上电后直到比较器能稳定工作之前应忽略比较器的输出和中断。

每个比较器可以被单独使能或禁止(关断)，通过置 1，CP0EN 位使能比较器 0，通过清除该位禁止比较器 0。如果比较器被禁止，而比较器的输出已通过交叉开关分配到 I/O 端口引脚上，则对应的引脚默认值为逻辑低电平，它的中断能力被停止，电源电流降到小于 1μA。

另外，比较器 0 还可被配置为复位源，详见 2.6 节中的“3. 比较器 0 复位”的介绍。

比较器 1 的操作与比较器 0 完全相同，只是比较器 1 不能被配置为复位源，而比较器 1 受 CPT1CN 寄存器控制，寄存器 CPT1CN 各位的定义如下：

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	复位值
CP1EN	CP1OUT	CP1RIF	CP1FIF	CP1HYP1	CP1HYP0	CP1HYN1	CP1HYN0	00000000
位 7	位 6	位 5	位 4	位 3	位 2	位 1	位 0	SFR地址：0x9F

其中，各位的含义如下：

位 7(CP1EN)——比较器 1 使能位。

0：比较器 1 禁止。

1：比较器 1 使能。

位 6(CP1OUT)——比较器 1 输出状态位。

0：电压值 CP1＋ < CP1－。

1：电压值 CP1＋ > CP1－。

位 5(CP1RIF)——比较器 1 上升沿中断标志。

0：自该标志位被清除后，没有发生比较器 1 上升沿中断。

1：自该标志位被清除后，发生了比较器 1 上升沿中断。

位 4(CP1FIF)——比较器 1 下降沿中断标志。

0：自该标志位被清除后，没有发生过比较器 1 下降沿中断。

1：自该标志位被清除后，发生了比较器 1 下降沿中断。

位 3 和位 2(CP1HYP1－0)——比较器 1 正向回差电压控制位。

00：禁止正向回差电压。

01：正向回差电压＝2mV。

10：正向回差电压＝4mV。

11：正向回差电压＝10mV。

位 1 和位 0(CP1HYN1－0)——比较器 1 负向回差电压控制位。

00：禁止负向回差电压。

01：负向回差电压＝2mV。

10：负向回差电压＝4mV。

11：负向回差电压＝10mV。

习　题　5

1. A/D 转换器的作用是什么？D/A 转换器的作用是什么？各在什么场合下使用？

2. A/D 转换器在转换原理上有哪些类型？各有什么特点？C8051F 系列单片机采用什么类型的 A/D 转换器？

3. D/A 转换器一般为什么类型？为什么？

4. 衡量 D/A 转换器的技术指标有哪些？

5. 试用 C51 编程语言写出程序，分别用查询法和中断法（外部信号 CNVSTR 有效中断）对通道 AIN0.0～AIN0.3 进行采样，将采样数据通过 UART0 发送出去，通过 PC 的超级终端观察结果。

6. 如用 C8051F020 的 ADC0 进行 16 位数据的采集，用什么方法来实现？试用此方法对片内温度进行采集（查询法），将采集到的值转换为摄氏温度值通过 PC 的超级终端观察。如要得到器件周围的环境温度则如何计算？

7. 使用 C8051F020 的 DAC1 产生方波和锯齿波，使用定时器 T4 溢出更新输出。试用 C51 编程语言写出程序。

8. C8051F020 单片机有几个比较器？如何配置使用？思考在实际的项目开发中什么时候会用到 C8051F020 单片机的比较器。