华清远见 FARSIGHT 始于2004 | 嵌入式培训专家

10年口碑积累，成功培养50000多名研发工程师，铸就专业品牌形象

华清远见的企业理念是不仅要做良心教育、做专业教育，更要做受人尊敬的职业教育。

《DSP 嵌入式系统开发典型案例》

作者：华清远见

专业始于专注 卓识源于远见

第 3 章　中低速数据采集系统设计

专业始于专注 卓识源于远见

3.1 案例要求和应用对象

案例要求：实现对信号频率在 1MHz 以下的模拟信号的采样，并分析信号，对信号进行数字滤波后，滤波后信号从数模转换器输出。

本案例属于中速数据采集，使用 TI 公司的 TMS320C5409 型号 DSP 作为核心处理器，实现读取 AD 转换器的数据，对数字信号处理后，从 DA 转换器输出信号。由于是中速采样，DSP 可以直接对 AD、DA 控制，不需要增加中间单元。

3.2 系统软硬件设计和调试

系统的基本框图由 AD、DA、DSP 以及 Flash 接口组成，如图 3.1 所示。

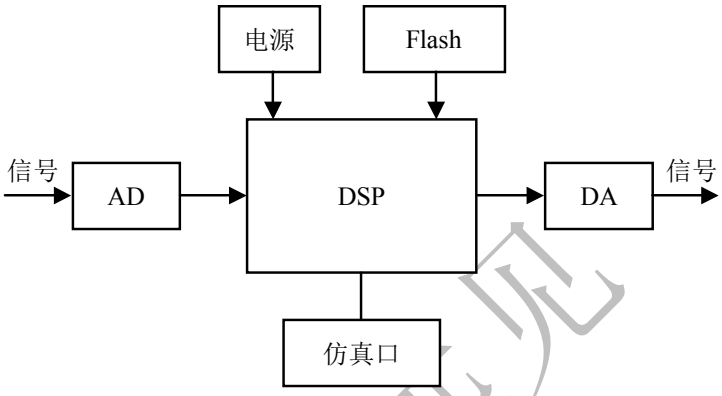

图 3.1 硬件结构框图

系统由 AD 转换器、DA 转换器、DSP 以及相应的电源转换电路、Flash 程序保存单元等组成。本案例 AD 转换器选择 TI 公司的 TLV1571、DA 转换器选择 TI 公司的 TLC7528、DSP 选择 TI 公司的 TMS320C5409、Flash 选择 SST29LE010、电源转换选择双电源方案。下面详细介绍软硬件设计方法。

3.2.1 TLV1571 的软硬件设计和调试

TLV1571 的内部结构如图 3.2 所示。TLV1571 的时钟源有内部时钟和外部时钟两种方式。TLV1571 的时钟信号可以由 CLK 从外部引入，也可以由 TLV1571 的内部时钟产生。和一般 AD 转换器不同，TLV1571 外部时钟信号必须经过 TLV1571 内部 MUX 时钟电路来提供给各个通道。由于 TLV1571 内部本身也带有时钟，因此 TLV1571 对各种时钟信号都兼容，这些时钟信号包括正弦波或者方波、TTL 电平或者 CMOS 电平。

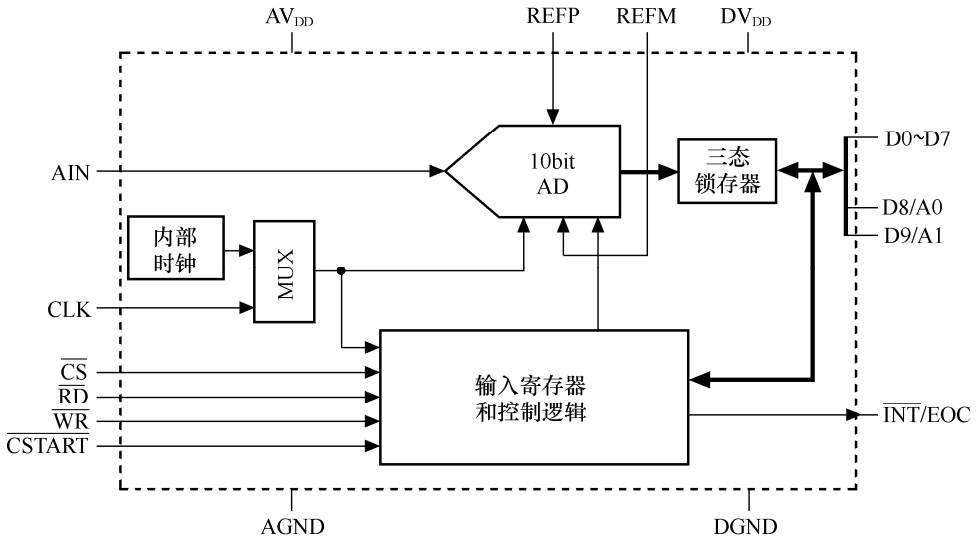

图 3.2　TLV1571 的内部结构原理框图

外部模拟信号从 TLV1571 的 AIN 引脚输入，信号到达 TLV1571 的中心单元（10bit 触发式 AD），将模拟信号转换为数字信号，同时 TLV1571 内部的输入寄存器和逻辑控制单元控制信号转变的方式，数字信号经过逻辑校验单元到达三态数据输出寄存器输出。此外，TLV1571 提供外部数据输出中断信号 INT 引脚，该引脚信号连接到 DSP 的中断信号，DSP 收到中断信号就可以读取数据总线，获得采样信号。

TLV1571 的引脚分布如图 3.3 所示。其中 $\overline{CS}$ 是片选信号，用于选通芯片；$\overline{RD}$ 是读信号，即 DSP 每读取一个数据通过该引脚通知 TLV1571，TLV1571 从而开始下一次采样；$\overline{WR}$ 是写信号，对 TLV1571 初始化寄存器，通过该引脚通知 TLV1571，TLV1571 从而将数据总线的数据写入到其内部寄存器；REFP 是高电平参考电压，一般直接连接到 VCC；REFM 是低电平参考电压，一般直接连接到地。

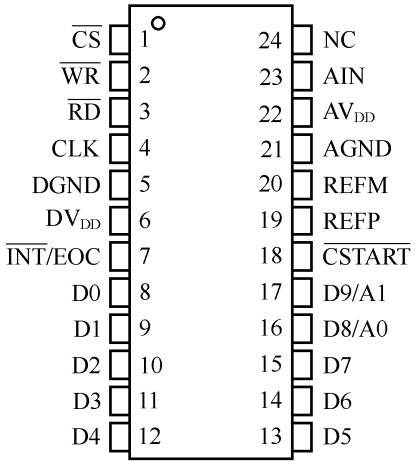

图 3.3　TLV1571 的引脚分布

TLV1571 的控制寄存器用于配置采样控制。TLV1571 有两个控制寄存器 CR0 和 CR1，它们都必须由用户配置。通过配置控制寄存器，TLV1571 可以选择不同的工作方式。数据总线的 D9 和 D8 引脚，也就是 A1 和 A0 引脚，用于区分当前配置哪一个寄存器，00 表示配置 CR0 寄存器，01 表示配置 CR1 寄存器，10 和 11 无效；数据总线其余的 8bit 用于配置控制寄存器。TLV1571 收到写信号脉冲信号后，就会将数据总线的值写入相应的控制寄存器。

TLV1571 内置有 10MHz 的振荡器，通过设置 CR1 寄存器的 D6 位，可使内部振荡器的速度提高 1 倍。如果 D6=0，内部振荡器的速度不变；如果 D6=1，内部振荡器的速度提高到 20MHz。通过设置 CR1 寄存器的 D3 位，可以设置 TLV1571 数字信号输出格式。如果 D3=0，输出数据格式是直接二进制格式；如果 D3=1，输出数据格式是二进制的补码格式。

TLV1571 的启动方式由 CR0 寄存器的 D7 位决定，表 3.1 给出了 TLV1571 转换启动方式的说明。

表 3.1 TLV1571 转换启动方式

方式	启动方式	说　明
单通道输入 CR0.D3=0 CR1.D7=0	硬件启动 CR0.D7=0	• $\overline{\text{CSTART}}$ 下降沿启动采样 • $\overline{\text{CSTART}}$ 上升沿启动转换 • INT 方式时，每次转换后产生一个 $\overline{\text{INT}}$ 脉冲 • EOC 方式时，转换开始时 EOC 将由高电平变至低电平，转换结束时返回高电平
	软件启动 CR0.D7=1	• 最初由 $\overline{\text{WR}}$ 的上升沿启动采样。在 $\overline{\text{RD}}$ 的上升沿发生采样 • 采样开始后的 6 个时钟后开始转换，INT 方式时，每次转换后产生一个 $\overline{\text{INT}}$ 脉冲 • EOC 方式时，转换开始时 EOC 由高电平变至低电平，转换结束时返回高电平

对于 TLV1571，单通道输入设置 CR0.D3=0，CR1.D7=0；采用软件启动设置 CR0.D7=1；采用内部时钟源方式设置 CR0.D5=0；时钟为 20MHz 设置 CR1.D6=1；采用二进制输出方式设置 CR1.D3=0。最终控制寄存器的设置为 CR0=0080H，CR1=0140H，将这两个数据写到控制寄存器，TLV1571 将按照以上设置开始工作。

TLV1571 提供 3 种自测试方式，并通过写 CR1 寄存器的 D1 和 D0 位来控制这 3 种测试方式。这些方式可用于不必提供外部信号就可检查 TLV1571 本身工作是否正常。具体方法如表 3.2 所示。

表 3.2 TLV1571 自测方式

CR1（D1、D0）	自测试电压	数字输出
D1=0；D0=0	正常工作方式	N/A
D1=0；D0=1	将 V_{REFM} 作为基准电压加到 AD	000h
D1=1；D0=0	将（V_{REFP}-V_{REFM}）/2 作为基准电压加到 AD	200h
D1=1；D0=1	将 V_{IN}=V_{REFP} 作为基准电压加到 AD	3FFh

TLV1571 具有两个基准电压输入引脚：REFP 和 REFM。REFP 引脚的电压，是输入模拟信号的最大值；REFM 引脚的电压，是输入模拟信号的最小值；REFP、REFM 以及模拟输入一般不超出正电源电压或低于 GND，它们符合 TLV1571 规定的极限参数。当输入信号等于或高于 REFP 时，数字输出为最大值；当输入信号等于或小于 REFM 时，数字输出为零。外部基准电压值如表 3.3 所示。

表 3.3 TLV1571 基准电压值

外部基准电压		最小值=3V	最大值
V_{REFP}	AV_{DD}=3V	2	AV_{DD}
	AV_{DD}=5V	2.5	AV_{DD}
V_{REFM}	AV_{DD}=3V	AGND	1
	AV_{DD}=5V	AGND	2
V_{REFP} $-V_{REFM}$		2	AV_{DD-GND}

基准输入为 V_{REFP} =AV_{DD}，V_{REFM} =AGND。其中 AV_{DD}=5V。

为了限制反馈到电源和基准电压的高频瞬变和随机噪声，要注意印制电路板的设计。为此，要求充分考虑旁路电容和基准引脚之间的距离。在许多情况下，0.1μF 瓷片电容足以在宽频带范围内保持低阻抗，但由于它们的频率在很大程度上取决于对各电源引脚的靠近程度，所以电容要尽可能放在靠近电源引脚的地方。为了减少高频和噪声耦合，推荐把数字和模拟地在芯片引脚处立即短路（可在引脚 DGND 和 AGND 之间布一条低的阻抗线来实现），如图 3.4 所示。

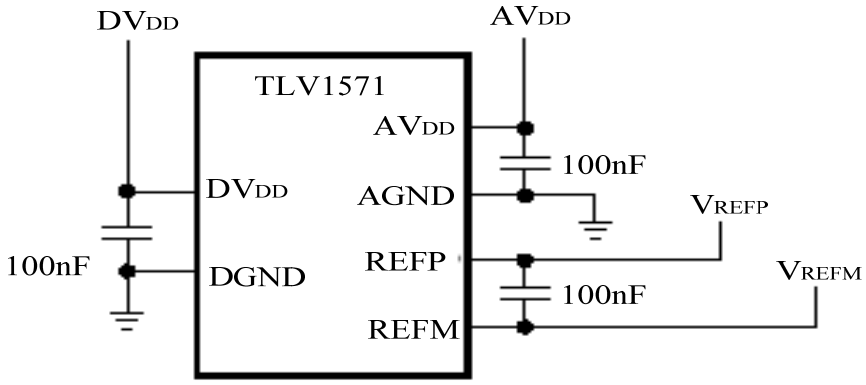

图 3.4　TLV1571 地线连接

TLV1571 与 TMS320C5409 的连接如图 3.5 所示。使用 DSP 的地址总线的 A0 引脚控制 TLV1571 的片选信号；使用 DSP 的 XF 引脚控制 TLV1571 的读信号；DSP 和 TLV1571 的数据总线和中断信号直接相连。

TLV1571 的调试，主要步骤如下。

（1）DSP 选通 TLV1571，根据图 3.5 的连接，将 DSP 的 A0 引脚置低，从而选通 TLV1571。

（2）DSP 初始化 TLV1571 的两个控制寄存器，通过 DSP 的 R/$\overline{W}$ 信号和数据总线初始化控制寄存器。

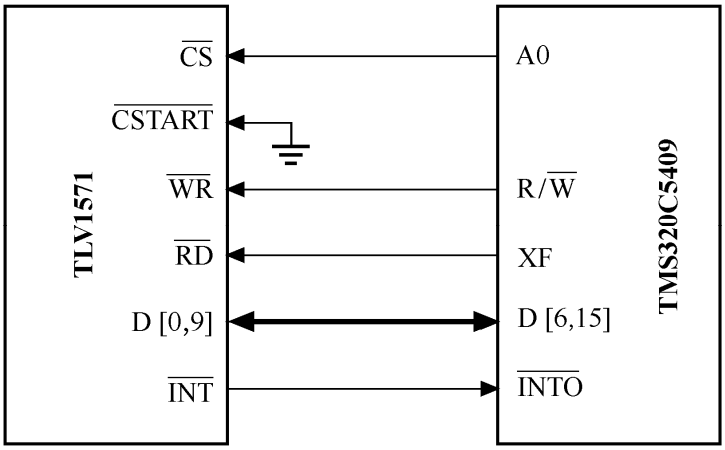

图 3.5　TLV1571 与 DSP5409 的连接

（3）DSP 接收 TLV1571 的中断信号，进入中断服务程序。

（4）DSP 在中断服务程序中，读取 TLV1571 的采样数据，并保存。

（5）重复步骤（3）和（4），读取下一个采样数据，并保存。

TLV1517 各种信号的时序如图 3.6 所示。

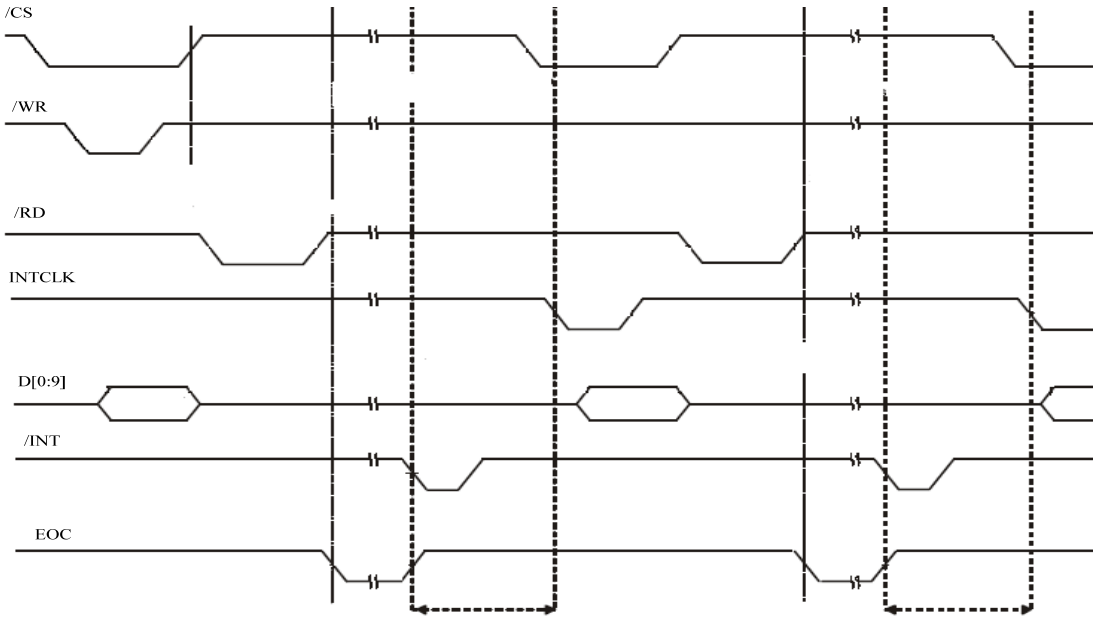

图 3.6　TLV1571 的读写时序

图 3.6 是 TLV1571 的读写时序图。从图中可以看出，对于 TLV1571 的具体操作如下。

（1）DSP 选通 TLV1571（置信号 $\overline{CS}$ 为低），同时写入两个寄存器的值到 TLV1571（置 $\overline{WR}$ 为低）。

（2）等待 TLV1571 产生中断信号（$\overline{INT}$ 信号产生下降沿），一般在初始化之后的 6 个时钟周期后才会有第一次中断信号产生。

（3）DSP 响应 TLV1571 的中断，读入数据到其内部存储器，数据读完后 DSP 通知 TLV1571，TLV1571 得到读入完成信号（$\overline{RD}$ 引脚电平为低）后，开始下一次采样。

在响应中断过程中，TLV1571 留出 6 个指令周期等待 DSP 读数据，直到 DSP 收到 $\overline{RD}$ 为低信号，TLV1571 才开始下一次采样。

在对 TLV1571 读或写的同时，必须保证 TLV1571 的 $\overline{CS}$ 为低，同时还必须保证其他外部 I/O 空间的片选信号为高（如果系统还有其他外部 I/O 空间），从而避免将数据写到其他外部设备中。

图 3.7 是实际中对 TLV1571 操作产生的信号波形。$\overline{WR}$ 信号开始有两个下降沿，其目的是为了设置 TLV1571 的两个控制寄存器。之后，$\overline{WR}$ 信号不再有下降沿，除非再次改变 TLV1571 的设置。紧接着 TLV1571 的中断信号产生，即图中的 $\overline{INT}$ 的下降沿。此时 TLV1571 已经完成第一次 AD 转换，数据已经送到数据线上，$\overline{INT}$ 的下降沿通知 DSP 可以读入数据。DSP 读入数据后，置 $\overline{RD}$ 为低，TLV1571 收到 $\overline{RD}$ 为低后开始下一次采样。

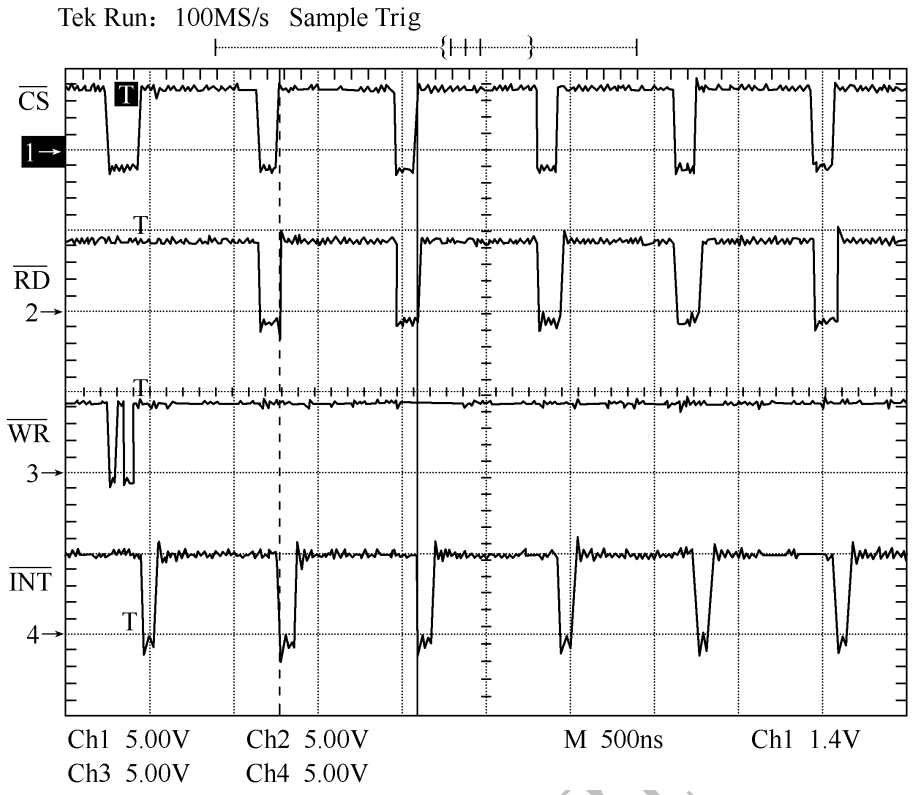

图 3.7 TLV1571 的实际运行时序

3.2.2 TLC7528 的软硬件设计和调试

TLC7528 是双通道、8bit、并行数模转换芯片，具有独立的片内数据锁存器。数据通过 8bit 输入口传送至两个 DA 数据锁存器中的任何一个。控制输入端 $\overline{DACA}$ 和 $\overline{DACB}$ 决定数据装载到哪一个 DA。器件的装载周期与随机存取存储器的写周期类似，因此可灵活地与通用微处理器数据总线相连接。TLC7528 是宽电压供电，供电电压在 5～15V 之间，功耗小于 15mW（典型值），2 或 4 象限乘法功能使得 TLC7528 十分适合应用于微处理器控制的增益设置和信号控制领域。如果 TLC7528 工作在电压方式下，其输出幅度是基于电压而不是基于电流。

TLC7528 包括两个相同的 8bit 乘法 DA 转换器 DACA 和 DACB。每一个转换器由反相 R-2R 梯形电阻网络、模拟开关以及数据锁存器组成。当 TLC7528 转换器工作时，二进制加权电流在转换器的输出端与 AGND 之间切换，以确保每一个梯形电阻网络分支电流恒定，且与开关状态无关。TLC7528 转换器的实际运用较为简单，仅需要在其外部加上外部运算放大器和电压基准就可正常工作。TLC7528 的 DAC 工作原理如图 3.8 所示。

从图 3.8 可以看出，TLC7528 通过数据总线、$\overline{CS}$ 、$\overline{WR}$ 以及 $\overline{DACA}$ 与 $\overline{DACB}$ 等控制信号与微处理器接口。当 $\overline{CS}$ 和 $\overline{WR}$ 均为低电平时，TLC7528 模拟输出将对 DB0～DB7 数据总线输入端的变化做出响应。在此方式下，TLC7528 的输入锁存器是透明的，从 DB0～DB7 数据总线输入的数据将直接影响 TLC7528 模拟输出。当 $\overline{CS}$ 或 $\overline{WR}$ 信号变为高电平时，DB0～DB7 数据总线输入端上的数据被锁存，直至 $\overline{CS}$ 和 $\overline{WR}$ 信号再次变低为止。必须注意的是，当 $\overline{CS}$ 为高电平时，不管 $\overline{WR}$ 信号的状态如何，DB0～DB7 数据总线将为高阻状态，禁止输入数据。

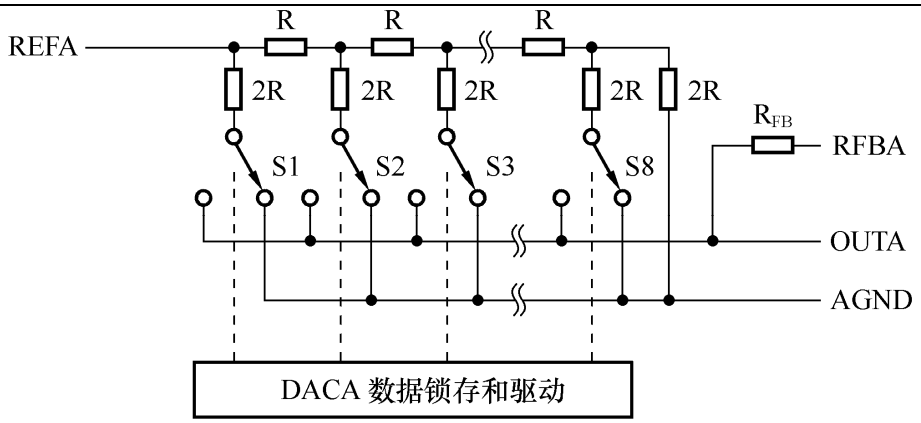

图 3.8　简化的 DACA 功能电路

当 TLC7528 工作电压为 5V 时，其数字输入与 TTL 电平和 CMOS 电平兼容。但工作电压超过 5V 时，其数字输入只与 CMOS 电平兼容。

表 3.4 是 TLC7528 的工作方式选择，根据引脚的电平可以选择不同的工作方式。

表 3.4　　TLC7528 工作方式选择

工作方式	$\overline{DACA}$ $\overline{DACB}$	$\overline{CS}$	$\overline{WR}$	DACA	DACB
1	L	L	L	写	保持
2	H	L	L	保持	写
3	X	H	X	保持	保持
4	X	X	H	保持	保持

注：表中 L 表示低电平；H 表示高电平；X 表示无关。

TLC7528 的引脚如图 3.9 所示。

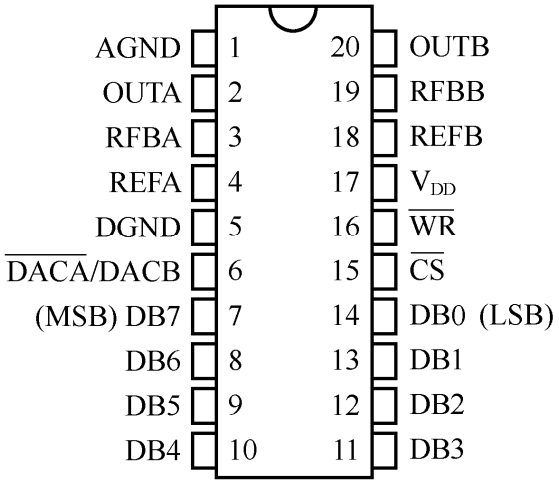

图 3.9　TLC7528 的引脚

其中，DB0～DB7 是数据脚，直接连接到 DSP 的数据总线。因为 DSP 的数据总线是 16bit，所以输出数据必须做相应的调整，以适合不同的接法，为了方便，一般接高 8bit 或者低 8bit 数据总线。$\overline{CS}$ 是片选信号，利用 DSP 的地址总线的 A1 引脚提供 DA 的片选信号。$\overline{DACA}$ 和 $\overline{DACB}$ 为输出通道选择信号，如果只使用一个输出 DACA，直接将此引脚和 $\overline{CS}$ 连接。REFA 和 REFB 是输出通道相应的参考输出电压。RFBA 和 OUTA 是选择输出极性引脚，不同的接法对应单极性输出或双极性输出，如图 3.10 和图 3.11 所示。

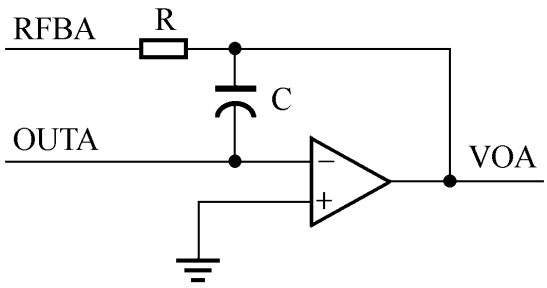

图 3.10　TLC7528 的单极性连接（*C*=10 pF）

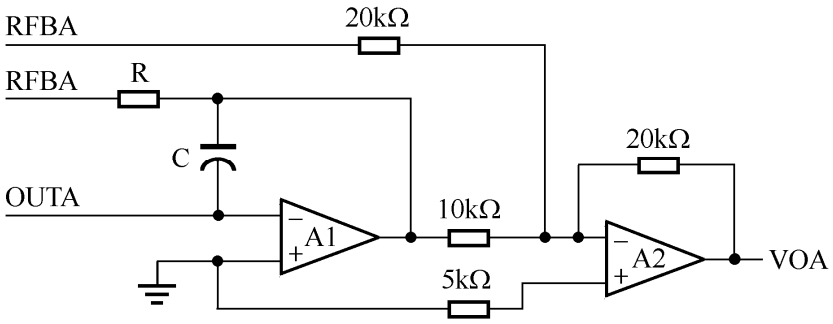

图 3.11　TLC7528 的双极性连接（C=10PF）

TLC7528 与 TMS320C5409 的连接如图 3.12 所示。使用 DSP 的地址总线的 A1 引脚控制 TLC7528 的片选信号；使用 DSP 的 R/W 引脚控制 TLC7528 的写信号；DSP 和 TLC7528 的数据总线直接相连。

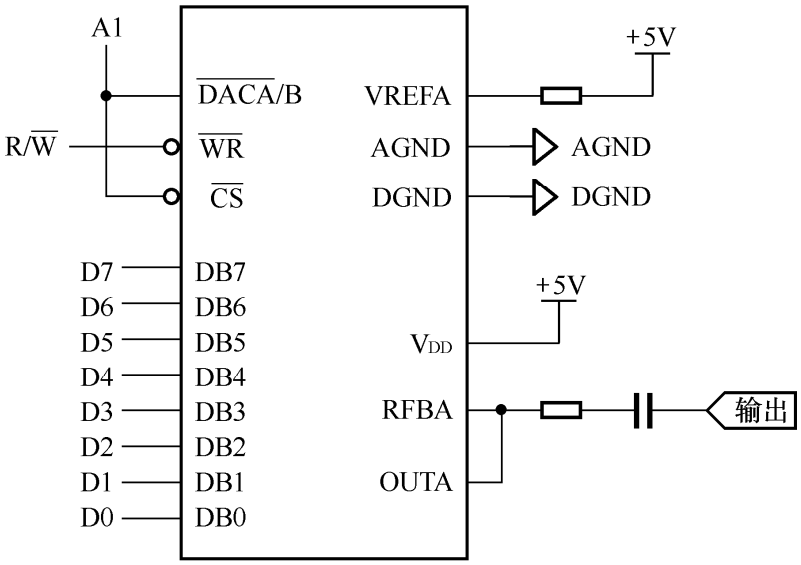

图 3.12　TLC7528 和 DSP 的连接

3.2.3　SST29LE010 的软硬件设计和调试

SST29LE010 的工作原理已经在高速数据采集中详细介绍。本节详细介绍 SST29LE010 与 TMS320C5409 的连接。

SST29LE010 连接到 TMS320C5409 作为其外部程序存储器，供 DSP 上电时在启动载入（BOOTLOAD）中使用，其作用是将 Flash 中保存的程序载入 DSP 中运行。TMS320C5409 与 Flash 的连接见图 3.13。其地址和数据总线连接到 TMS320C5409 外部总线，$\overline{CE}$ 直接接地，使 Flash 一直处于选通状态。TMS320C5409 的引脚 D8 用于控制 Flash 的写使能，当引脚 D8 为低电平时，可对 Flash 编程和擦除，这样可以通过编程控制引脚 D8 而使 $\overline{WE}$ 的时序满足要求；为防止误写，$\overline{WE}$ 一般被设置为高电平。而与 TMS320C5409 $\overline{DS}$ 相连的 $\overline{OE}$，只有设置为低电平时才有效，但一般被设置为高电平，以保护 Flash 的芯片。

图中所示电路采用了双刀双置开关，当开关置上方时（实线表示的地方），Flash 的 $\overline{OE}$ 与 DSP 的 DS＃相连，$\overline{WE}$ 置高，可上电读 Flash；当开关置下方时（虚线表示），$\overline{OE}$ 置高，$\overline{WE}$ 与 DSP 的 D8 相连，为写操作。

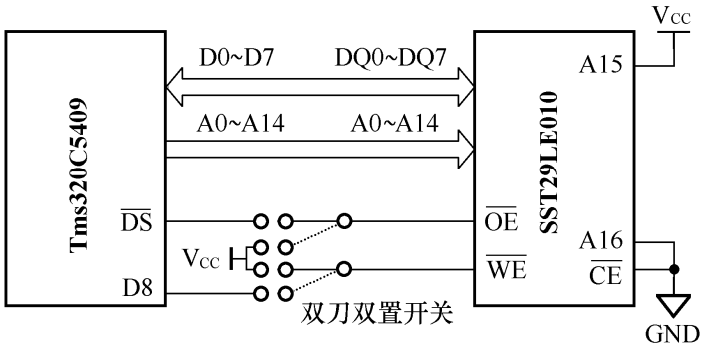

图 3.13　TMS320C5409 与 Flash 的连接

这样做的目的是将写操作和读操作完全分开，以和 DSP 的自举要求达成一致。在 DSP 运行过程中，当写操作和读操作完全分开后，Flash 的写信号将强制为高电平，以确保 DSP 不改写 Flash 中的数据。此外，将 Flash 的写信号强制为高电平后，还能确保系统在反复的上电复位过程中，不会改写 Flash 中的数据。

硬件设计时一般只要满足 BOOTLOAD 程序对各种时序的要求，也就是只要硬件连接正确就可以了。但 Flash 和 DSP 的软件设计只包括 DSP 对 Flash 的写操作过程，而 DSP 对 Flash 的读操作将由 DSP 上电后的自举程序控制。

DSP 写 Flash 的目的是将用户所编写的 DSP 程序写入 Flash，以供 DSP 上电调用。写入程序之前必须将程序的相关信息写入 Flash，这些相关信息包括程序的开始地址、程序大小、程序保存的地址、调用程序的方式等。按照自举程序的要求，写入程序的格式如表 3.5 所示，表中所述内容就是实际保存在 Flash 中的内容。

表 3.5　　程序写入的格式

数据区地址	内容	含义
8000～8001	08AA	8bitBOOTLOAD 标识
8002～8003	7FFF	SWWSR
8004～8005	F800	BSCR
8006～8007	0000	程序入口 XPC
8008～8009	2000	程序入口地址
800A～800B	0400	程序块长度
800C～800D	0000	存放目标 XPC
800E～800F	2000	存放目标地址
8010～8011	xxxx	程序代码
8012 以后	xxxx	程序代码

对表 3.5 的说明如下。

（1）使用 8bit 并行 BOOT 方式是根据 Flash 芯片的特征而决定的。

（2）程序在 Flash 中的保存地址必须从 8000H 开始，所以 Flash 的 A15 引脚始终置高。

（3）程序长度内容不要求和实际完全一致，只需要大于程序的实际长度就可以。

（4）程序存放目标地址必须和程序自身所要求的地址一致，程序自身要求的地址在配置文件中定义。

如何将以上内容写入 Flash 有两种方法，第一种方法通过编程器将程序写入 Flash 中。该方法简单，不需要编写程序，只需要编程器，但如今大部分型号的 Flash 都是表面贴装，因此无法通过编程器写入 Flash，所以现在已很少使用该方法。另外一种方法是通过 DSP 对 Flash 写入程序，即将程序通过仿真器从计算机调入到 DSP，运行程序，从而将程序写入 Flash。该方法方便可靠，适用于表面贴装的芯片，且只需要 DSP 仿真器。而在 DSP 开发过程中，一般都配备了仿真器这种工具，而无需额外购买。因此，通过 DSP 对 Flash 写入程序这种方法受到普遍应用。

3.2.4 电源和复位电路设计

TMS320C5409 型号 DSP 采用 3.3V 和 1.8V 电压供电，其中 I/O 采用 3.3V 电压，芯片内核采用 1.8V 电压，内核采用低电压供电可以降低整个芯片的工作功耗。本节介绍 TPS73xx 系列的电压转换芯片，它们是 TI 公司为了配合 C54xx 系列 DSP 而专门设计的电压转换芯片。

TPS73xx 系列的电压转换芯片包括 3 种固定输出电压的稳压器：TPS7333（3.3V）、TPS7348（4.85V）以及 TPS7350（5V）。同时，该系列还提供输出可调的低降落稳压器（LDO）TPS7301（1.2～9.75V）。此外，TPS73xx 系列的 LDO 和早期的 LDO 电压转换芯片相比有许多优点，例如改进节省功率的关断方式，增加电源电压监控功能等。

常规的 LDO 稳压器采用 PNP 通路元件。PNP 通路元件的基流正比于通过稳压器的负载电流，其实际工作电流比典型的静态电流与负载电流关系曲线中给出的电流大。因此，采用 PNP 通路元件的电压转换芯片，可能会导致常规的 LDO 稳压器进入降落状态，从而使电流趋于饱和，为了维持负载电流，此时 PNP 通路元件的基极电流就会增加。如果这种情况发生在芯片上电期间，会导致较大的启动电流，而限制的电源电流无法满足启动电路，将使启动失败。因此，当负载变化时，常规 LDO 稳压器可能无法正常工作。

TPS73xx 系列 LDO 克服了常规 LDO 稳压器的弊端，它具有非常低的静态电流，即使对于变化较大的负载，静态电流仍能保持稳定。TPS73xx 系列 LDO 采用晶石金属氧化物半导体（PMOS，Pachnolite Metal-Oxide-Semiconductor）晶体管来传送电流。PMOS 元件的栅极是电压驱动的，所要求的工作电流较低，且在全负载范围内其工作电流能保持不变。因而采用 PMOS 通路元件的电压转换芯片，即使稳压器处于降落状况，静态电流仍然保持较低值。所以当负载发生变化时，TPS73xx 系列 LDO 仍能正常工作。

TPS73xx 的另一个特点是具有关断特性。当关断时，可以使电源输出处于高阻状态（基本上等于反馈分压电阻），并使静态电流减至 0.5μA 以下。当不使用关断特性时，器件对使能端的跃变可以迅速做出反应，通常在 120μs 之后可重新建立起稳定的输出电压。

TPS73xx 上电时，输出电压跟踪输入电压。由于 $\overline{\text{RESET}}$ 输出是漏极开路的 NMOS，所以应当使用上拉电阻，以确保显示逻辑信号为高电平。因此，当输入电压接近有效 $\overline{\text{RESET}}$ 信号所需的最小值（25℃时规定为 1.5V，在整个推荐工作范围内为 1.9V）时，$\overline{\text{RESET}}$ 输出有效（低电平）；当输出电压达到合适的正向输入门限时，200μs（典型值）的超时周期开始（在此周期内，$\overline{\text{RESET}}$ 输出保持低电平）；一旦超时周期结束，$\overline{\text{RESET}}$ 输出便变为无效。

在欠压状态下，TPS73xx 的 $\overline{\text{RESET}}$ 输出能启动复位信号，该信号能实现对 DSP 的复位。TPS73xx 通过内部的比较器来监视稳压器的输出电容，从而检测输出电压是否处于欠压状态。当欠压状态发生时，$\overline{\text{RESET}}$ 输出晶体管导通，使 $\overline{\text{RESET}}$ 信号变为低电平。

TPS73xx 电源掉电时，电源电压监控功能将被激活。当输入电压下降且达到降落电压时，输出电压将随输入电压的下降而线性地下降。当输出电压降至低于规定的负向输入门限以下时，$\overline{\text{RESET}}$ 输出变为有效（低电平）。如果输入电压降至有效 $\overline{\text{RESET}}$ 所需的最小值以下，那么 $\overline{\text{RESET}}$ 是不确定的。因为电路具有监视稳压器输出电压的功能，所以 $\overline{\text{RESET}}$ 输出可以被禁止稳压器触发，或者被任何能导致输出降至规定的负向输入门限以下的故障状态（如输出短路和低输入电压等）触发。如果输出电压恢复正常（如故障排除后，稳压器恢复正常供电），内部定时器将被启动，它将在 200μs（典型值）的超时周期内使 $\overline{\text{RESET}}$ 信号保持有效。

TPS73xx 电路中输入和输出电容的选择。TPS73xx 不需要输入电容。但当它离电源的距离大于几英寸时，瓷片旁通电容（0.047pF 至 0.1μF）可以改进负载的瞬态响应。如果有快速上升时间的大负载瞬变（数百毫安），就必须使用大容量的电解电容。通常必须选择与 TPS73xx 相匹配的输入和输出电容。如果输入和输出电容选择不适当，那么瞬变负载或电源脉冲可能导致 TPS73xx 复位信号的产生。如果使用的 ESR 输出电容较高，那么快于 5μs 的负载瞬变可能产生 TPS73xx 复位信号。如果瞬变宽度很窄，那么瞬变期间内，输出电压的尖峰可以低于复位门限而不触发 TPS73x 的复位电路。在触发复位电路之前，1μs 的瞬变必须降至比门限低 500mV。2μs 的瞬变可以在刚好低于门限 400mV 处触发 $\overline{\text{RESET}}$。ESR 的输出电容低，可以通过减少瞬变期间输出电压的下降而有助于正常工作，当预期可能发生快瞬变时，应当使用低 ESR 的输出电容。

TPS73xx 与外部设备的连接。为了保证稳压器正常工作，外部设备的传感器输出端必须连接到 TPS73xx 稳压器的输入端，在 TPS73xx 系列电压转换芯片的内部，电源通过电阻分压网络连接至高阻宽带放大器，噪声拾取反馈通至稳压器输出，这两个端点之间的连线应尽可能短。但是远程检测时，外部设备的传感器输出端可以在关键处进行连接，以改进连接的性能。连接的布线方式应该尽量避免噪声拾取或使噪声拾取为最小。尽量不要在传感器与稳压器输出之间加 RC 网络来滤除噪声，因为这样可能会引起稳压器振荡。TPS7301 的硬件连接如图 3.14 所示。

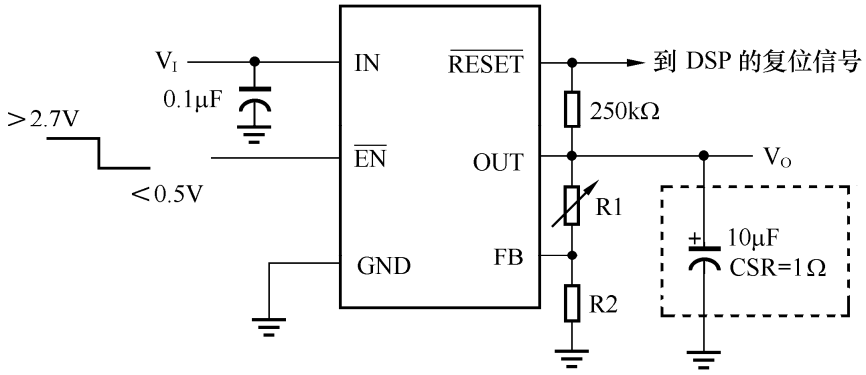

图 3.14　TPS7301 的连接

图 3.14 的外部电阻分压器可调整稳压器输出电压。控制稳压器输出电压的关系如下：

$$V_0 = V_{REF} \times \left(1 + \frac{R_1}{R_2}\right)$$

其中　V_{REF} 为基准电压，典型值为 1.182V。

电阻 R_1 和 R_2 选择的准则是使得分压器电流近似于 7μA。推荐的 R2 的阻值为 169kΩ，R_1 的阻值根据所需的输出电压来调整（一般为 82kΩ）。因为 FB 端的漏电流会引起误差，所以应当避免使用较大值的 R_1 与 R_2。根据上述关系式可得到 R_1 的表达式，$R_1=(V_0/V_{REF}-1)\times R_2$，得到不同输出电压对应的 R_1 阻值如表 3.6 所示，表中电阻单位为 kΩ。

表 3.6　　输出电压

输出电压（V）	R_1（kΩ）	R_2（kΩ）
2.5	191	169
3.3	309	169
3.6	348	169
4	402	169
5	549	169
6.4	750	169

注：符合表中阻值的电阻属于高精度电阻。实际中可以使用普通阻值的电阻，例如，对于 1.8V 电压输出，可以选择 R_1=82kΩ 与 R_2=180kΩ的值。

TPS7333 的应用和 TPS7301 的应用基本一致，请参照有关芯片手册。详细的 TPS7301 和 TPS7333 的电压转换连接如图 3.15 所示。

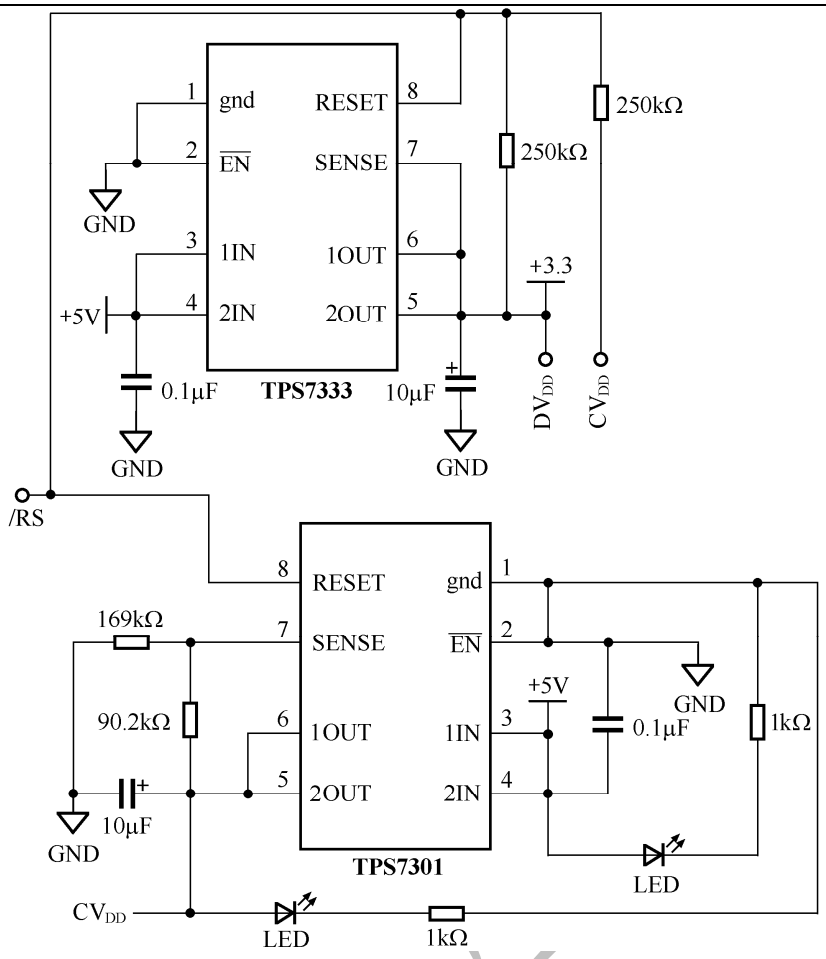

图 3.15　TPS7301 和 TPS7333 的连接

3.2.5　时钟电路设计

本案例的时钟信号的连接和高速数据采集系统一致，但 DSP 内部的频率设置电路和系数设置有所不同。DSP 的频率设置引脚为 CLKMD1～CKLMD3，这些引脚的状态来决定 DSP 内部倍频的大小。倍频是指在外部晶振的基础乘以设定的倍数，倍数与 CLKMD1～CLKMD3 的关系如表 3.7 所示。表中 PLL 禁止表示 DSP 内部的倍频电路禁止，此时 DSP 内部的分频电路工作，DSP 工作时钟为输入时钟的一半或者 1/4。

表 3.7　　CLKMD1～CLKMD3 与分频关系

CLKMD1	CLKMD2	CLKMD3	CLKMD（复位值）	时钟模式
0	0	0	E007H	PLL×15
0	0	1	9007H	PLL×10
0	1	0	4007H	PLL×5
1	0	0	1007H	PLL×2
1	1	0	F007H	PLL×1
1	1	1	0000H	1/2（PLL 禁止）
1	0	1	F000H	1/4（PLL 禁止）
0	1	1	——	Reserved

本案例的 JTAG 仿真口的设计遵循 IEEE 标准设置。为了调试和扩展，系统将常用的地址总线、数据总线和缓冲串口总线连接到插件上。整个系统的最终布局如图 3.16 所示。

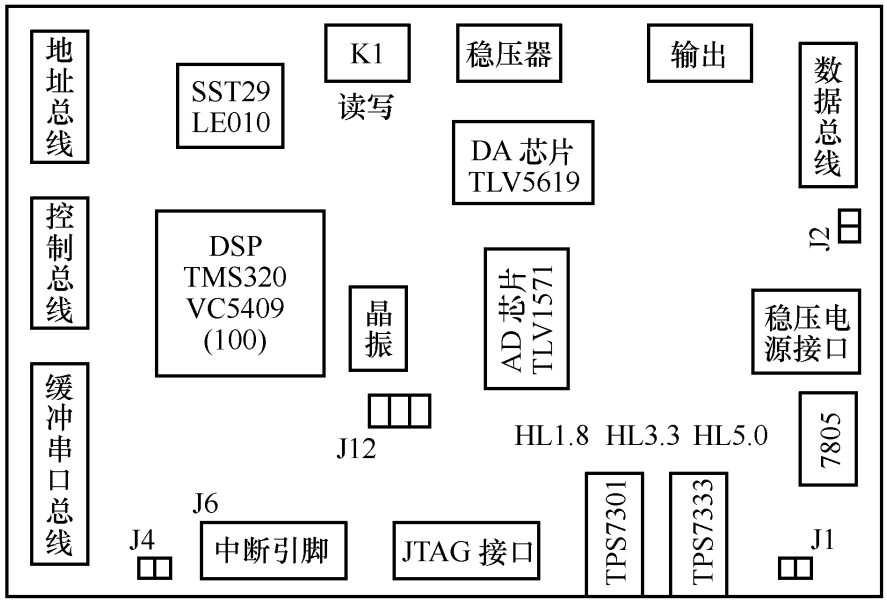

图 3.16　系统整体布局

图 3.16 中，HL3.3 为 DSP 芯片 I/O 电源（3.3V）指示灯、HL1.8 为 DSP 芯片核电源（1.8V）指示灯、HL5.0 为 5V 电源指示灯；J4 为脱机或仿真运行方式选择引脚，将 J4 短路时为脱机运行，反之则为仿真运行；J1 左边为模拟地接口，右边为 5V 电压接口；J2 上面为输入信号接口，下面为数字地接口；K1 为 Flash 读写开关，当 K1 开关置左端时，可进行上电读操作，当开关置右端时，可把自己编写的程序通过 DSP 仿真器写入到 Flash 中。

3.3　程序代码

3.3.1　AD 测试程序代码

TLV1571 调试过程中需要注意的问题如下。

（1）首先必须正确设置 TLV1571 的两个控制寄存器，可以采用写入后读控制寄存器的内容，来确认 TLV1571 的两个控制寄存器是否正确写入；也可以采用循环写入，利用示波器来确定写入的两个脉冲信号是否正确。

（2）设置一个控制寄存器后，必须等待一段时间（在程序中进入一些 nop 实现等待），在等待时间内，确保 TLV1571 将数据写到前一个控制寄存器中，才能开始设置后一个控制寄存器。

（3）因为 TMS320C5409 的读写信号只有一根地址线，所以只能利用 TMS320C5409 的 XF 引脚进行控制 TLV1571 的读信号。

（4）为了验证采样的信号是否正确，可以在仿真软件 CCS 中观察采样后数据的时域图或频域图，从而确定采样的正确性。

```
                .mmregs
                .def   jump
                .def   start

k_sample_num    .set 256                ；设置保存采样数据的点数
out_data        .usect      "out_vars", k_sample_num
                                        ；数据输出位置，大小为 k_sample_num
                                        ；用来保存一批数据来查看采样是否正确
                                        ；也可以供需要分批处理数据的程序调用程序中
```

```
                                        ；可以用 ar0～ar7 来指定数据的首地址
in_data         .usect      "in_vars",1
d_cr0_send      .usect      "cr_send",1
d_cr1_send      .usect      "cr_send",1
                                        ；定义 2 个状态字
                                        ；此状态字就是需要开始写入 TLV1571 的初始值
d_temp          .usect      "temp",1    ；存放临时数据
k_cr0_send      .set        0080h       ；初始化两个状态字的值
k_cr1_send      .set        0100h
                                        ；重新映射中断向量的变量
k_iptr          .set    000111000b<<07  ；point 1C00H  新的向量区
                                        ；iptr 是 DSP 内部 PMST 寄存器的高 9bit
                                        ；用来指定中断向量表的位置，复位时全为 1
                                        ；为了调试此程序，iptr 将重新映射到地址 1D00 处
k_temp          .set    1111111b
                                        ；定义一个常数，用于以后与 pmst 的与操作
                                        ；保持 pmst 的低 6bit 不变
d_pmst          .usect  "pmst",1
                .sect   "prog"
start:
                ldm     pmst,a          ；取出 PMST 的值，放入累加器 a
                                        ；改变 pmst 的高 9bit，再放入 pmst
                and     #k_temp,a
                or      #k_iptr,a
                stl     a,ar2
                mvdm    ar2,pmst

                stm     #0001h,imr      ；开中断 0
                nop
                nop
                stm     #0002h,ifr      ；通过写 IFR，取消所有已经挂起的中断
                nop
                rsbx    intm            ；状态寄存器 ST0 的 INTM 位，允许中断
                nop
                stm     #d_temp,ar2
                stm     #d_cr0_send,ar1
                st      #k_cr0_send,*ar1+
                st      #k_cr1_send,*ar1
                stm     #d_cr0_send,ar1
                ld      #k_sample_num,a ；k_sample_num 用来计数
                stm     #out_data,ar5
                portr   01h,*ar2        ；读其他端口，使 ADC 的 CS 为高
                nop
                nop
                nop
                portw   *ar1+,02h       ；填 AD 转换器的寄存器 cr1
                rpt     #8
```

```
                nop
                portw   *ar1,02h            ；填 AD 转换器的寄存器 cr1
                portr   01h, *ar2           ；读其他端口，使 ADC 的 CS 为高
                rpt     #10
                nop
                nop                         ；此时，写好状态字，AD 开始采样
                                            ；等待中断， 使用累加器 a 做计数器
                                            ；当写完 k_sample_num 个数据后
wait:           nop                         ；重置累加器 a，并使 ar5 指向 out_data 的开始地址
                nop
                bc      wait,aneq
                nop
                stm     #out_data,ar5
                ld      #k_sample_num,a
                b       wait
                nop
                                            ；中断服务程序，用以读转换信号，放入 out_data 区
jump:           sub     #1h,a               ；计数
                rsbx    xf                  ；发出读信号，AD 收到后开始下一次采样
                rpt     #5
                nop                         ；等待几个周期，便于示波器查看波形
                portr   00h,*ar5
                                            ；读 AD 转换数据,写在 RAM 的 out_data 数据段
                rpt     #5
                nop
                ssbx    xf                  ；使读信号无效
                nop
                portr   01h,*ar2            ；读其他端口，使 ADC 的 CS 为高
                nop
                nop
                portw   *ar5+,03h           ；从 DA 输出
                rete                        ；中断返回
                .end
```

3.3.2 AD 测试程序中断向量代码

```
        .mmregs
        .sect  "vectors"
        .ref   jump
        .ref   start
        .def   reset

k_stack_size    .set            200
k_stack     .usect    "stack_section",k_stack_size
system_stack    .set     k_stack+k_stack_size

reset:          bd          start
```

```
        stm   #system_stack,sp

nmi:    rete
        nop
        nop
        nop

sint17          .space 4*16
sint18          .space 4*16
sint19          .space 4*16
sint20          .space 4*16
sint21          .space 4*16
sint22          .space 4*16
sint23          .space 4*16
sint24          .space 4*16
sint25          .space 4*16
sint26          .space 4*16
sint27          .space 4*16
sint28          .space 4*16
sint29          .space 4*16
sint30          .space 4*16

int0:  b    jump
        nop
        nop
int1:   rete
        nop
        nop
        nop
int2:   rete
        nop
        nop
        nop
tint0: rete
        nop
        nop
        nop
brint0:     rete
        nop
        nop
        nop
bxint0:     rete
        nop
        nop
        nop
dmac0:      rete
        nop
```

```
            nop
            nop
dmac1:      rete
            nop
            nop
            nop
int3:       rete
            nop
            nop
            nop
hpint:      rete
            nop
            nop
            nop
dmac2:      rete
            nop
            nop
            nop
bxint1:     rete
            nop
            nop
            nop
dmac4:      rete
            nop
            nop
            nop
dmac5:      rete
            nop
            nop
            nop
            .end
```

3.3.3 AD 测试程序配置文件代码

```
-m ad.map
-o ad.out
-e reset

MEMORY
{
      PAGE 0:   VEC          : origin=0x1c00,      length=0x0080
                PROG         : origin=0x1d00,      length=0x1000
      PAGE 1:   STACKS       : origin=0x0200,      length=0x0300
}
SECTIONS
{
```

```
    vectors :       {} > VEC        PAGE 0
    prog :          {} > PROG       PAGE 0
    stack_section:  {} > STACKS     PAGE 1
}
```

3.3.4 DA 测试程序代码

DA 转换器的调试十分简单，编写一段发送数据的 DSP 程序，DSP 将不同的数据发送到 DA，如果 DA 工作正常，将输出不同幅度的模拟信号。如果采用两个数据交替发送，模拟信号实际上就是方波信号。运行下面程序，用示波器检测 DA 的输出引脚，可看到连续的方波信号。其输出方波的幅值可以通过改变 outdata1 和 outdata2 的值来改变，输出方波的占空比可以通过改变等待的时间来改变。

对于 I/O 空间的选通使用指令 portw，portw 可以将指定的数据输出到数据管脚上，同时选通相应的外设，屏蔽其他外设。这样可使 DA 收到转换信号后，就开始将数据转换输出。

```
                .mmregs
outdata1        .set    0000h           ；定义一个数据（最小值）
outdata2        .set    0ffffh          ；定义一个数据（最大值）
                .text
                st      #outdata1,*ar2  ；将数据送到*ar2
                st      #outdata2,*ar3  ；将数据送到*ar3
jump:
                  portw   ar2,0fdh      ；将*ar2 内容输出到数据总线
                  rpt     #7fffh        ；等待一段时间
                  nop
                  rpt     #7fffh
                  nop
                  rpt     #7fffh
                  nop
                  rpt     #7fffh
                  nop
                  portw   *ar3,0fdh     ；将*ar3 内容输出到数据总线
                  rpt     #7fffh
                  nop
                  rpt     #7fffh
                  nop
                  rpt     #7fffh
                  nop
                  rpt     #7fffh
                  nop
                  b       jump          ；循环输出两个数据
                .end                    ；代码段结束
```

3.3.5 DA 测试程序配置文件代码

```
MEMORY  {
        PAGE 0:         PARAM:   org = 0080h     len = 1780h
        PAGE 1:         DARAM:   org = 0080h     len = 1780h
```

```
        }
SECTIONS   {
        .text : >          PARAM     PAGE 0
        .bss : >           DARAM     PAGE 1
        .data : >          DARAM     PAGE 1
        }
```

3.3.6 写 Flash 程序代码

写 Flash 程序注意如下：

（1）工程文件中不要加上中断向量表文件，而应将中断向量表文件加到头文件中。

（2）Flash 的读信号用 DSP 的 $\overline{DS}$ 引脚来控制。

（3）Flash 的片选信号一直为低电平。

（4）用 DSP 的 D8 引脚作为 Flash 的写信号。

（5）可以通过修改写入程序块的大小来适应不同的程序要求。此时，配置文件应做相应的调整。

```
            .mmregs
SWCR        .set    002BH              ；定义 SWCR 寄存器
OUTDATA1    .set    0000H              ；定义 DA 输出数值 1
OUTDATA 2   .set    07FFH              ；定义 DA 输出数值 2
K_IPTR      .set    000111000B<<07     ；定义中断入口地址高 9bit
K_TEMP      .set    1111111B           ；定义中断入口地址低 7bit
            .data                      ；数据段开始
RES_SPACE：                            ；程序标号
        .space      07E0h              ；将前 127 个字节预留
        .word       8000h
        .word       0000H
        .sect       ".DISPLAY"
        .label      DISPLAY_SRC
        .word       08AAH              ；定义 BOOT 启动方式，并行 8bit
        .word       7fffH              ；SWWSR 的值
        .word       0F000H             ；BSCR 的值
        .word       0000h              ；xpc 的值
        .word       0200h              ；pc 的值
        .word       01e0h              ；程序长度
        .word       0000h              ；第一段的 xpc 值
        .word       0100h              ；第一段的 pc 值
        .copy       "vectors.asm"      ；拷贝中断向量表程序

start：                                ；以下斜体部分为写入 Flash 的程序，这里不做说明
        stm         #3000h,sp          ；如果写入的内容改变，可以将用户程序替代斜体
        STM         #80H,AR2           ；部分程序就可以了
        STM         #81H,AR3
        ST          #outdata1,*ar2
        ST          #outdata2,*ar3
begin：
        portw       *ar2,0bfffh
```

```
        rpt     #7fffh
        nop
        portw   *ar3,0bfffh
       rpt     #7fffh
       nop
       b   begin                    ；程序结束

    .space    2000h                 ；以下程序实现将斜体部分程序内容写入 Flash 中
    .label DISPLAY_END

  .text
MAIN_START：
    STM     #3000h,SP
    STM     #0FFA0H,PMST
     STM     #07FFFH,SWWSR
     STM     #0FFFFH,SWCR
     STM     #0H,34H
     STM     #0H,35H
     STM     #1H,34H
     STM     #0H,35H
     STM     #0EH,34H
     STM     #3F4FH,35H
     SSBX    INTM                  ；关闭所有中断
     STM      #8000h,AR6
     STM     #8000H,AR3            ；写入的地址
     STM     DISPLAY_SRC,AR5       ；写入的程序头
    STM     #0BH,AR4               ；一共写入 12 页
WRI_RPT：
     STM     #63,AR1               ；一页写入 64Byte，在 Flash 中为 128Byte

WRI_LOP：
    LD      *AR5,-8,A              ；将 128 个半字节依次写入 Flash 中
    NOP
    NOP
    AND     #0FEFFH,A          ；将 D8 数据总线置 0，用于 Flash 的写时钟
    NOP
    NOP
     STL     A,*AR6
    NOP
    NOP
    LD      *AR5,-8,A
    NOP
    NOP
    OR      #0100H,A               ；将 D8 数据总线置 1
    NOP
    NOP
    STL     A,*AR6+                ；地址加 1，准备写入下一个数据
```

```
    NOP
    NOP
    LD      *AR5,A
    NOP
    NOP
    AND     #0FEFFH,A
    NOP
    NOP
    STL     A,*AR6
    NOP
    NOP
    LD      *AR5+,A
    NOP
    NOP
    OR      #0100H,A
    NOP
    NOP
    STL     A,*AR6+
    NOP
    NOP
    BANZ    WRI_LOP,*AR1-
    CALL    DELAYY
    BANZ    WRI_RPT,*AR4-
    STM     #RES_SPACE,AR1
    STM     #0FF80H,AR5                 ；将原中断向量清零
    STM     #127,AR3
WRI_LOOP：
    LD    *AR1,-8,A
    NOP
    NOP
    AND     #0FEFFH,A
    NOP
    NOP
    STL     A,*AR5
    NOP
    NOP
    LD      *AR1+,-8,A
    NOP
    NOP
    OR      #0100H,A
    NOP
    NOP
    STL   A,*AR5+
    NOP
    BANZ    WRI_LOOP,*AR3-
LOAD_LOOP：
    B       LOAD_LOOP
```

```
DELAYY:                                    ; 延时子程序
      PSHM     AR2                         ; 等待 Flash 将一页数据写入的时间
      STM      #0020H,AR2
DELAYY_LOOP:
      RPT      #0fff0h
      NOP
      BANZ     DELAYY_LOOP,*AR2-
      POPM     AR2
      RET
      .END                                 ; 程序结束
```

3.3.7 写 Flash 配置文件代码

```
MEMORY   {
       PAGE 0:       PARAM0:      org = 0F8h      len = 708h
       PAGE 0:       PARAM1:      org = 800h      len = 800h
       PAGE 1:       DARAM1:      org = 1000H     len = 1000H
      PAGE 1:        DARAM:       org = 2000h     len = 1000h
          }
SECTIONS{
          .text       :   >        PARAM1   PAGE 0
          .DISPLAY    :   >        PARAM0   PAGE 0
          .bss        :   >        DARAM1   PAGE 1
          .data       :   >        DARAM    PAGE 1
          }
```

3.3.8 写 Flash 中断向量代码

```
          .sect      ".DISPLAY1"
          .ref       start
          .def       reset

k_stack_size    .set       200
k_stack         .usect     "stack_section",k_stack_size
system_stack    .set       k_stack+k_stack_size

          .word      08aah          ; INDICATE 8BITS PARALLEL BOOT MODE
          .word      7fffh          ; VALUE FOR SWWSR
          .word      0f000h         ; VALUE FOR BSCR
          .word      0000h          ; xpc for entry point
          .word      0200h          ; pc for entry point
          .word      01e0h          ; size of first section
          .word      0000h          ; xpc for first section
          .word      0100h          ; pc for first section
```

```
reset:      b       start
            stm     #3000h,sp

nmi:        rete
            nop
            nop
            nop

sint17          .space 4*16
sint18          .space 4*16
sint19          .space 4*16
sint20          .space 4*16
sint21          .space 4*16
sint22          .space 4*16
sint23          .space 4*16
sint24          .space 4*16
sint25          .space 4*16
sint26          .space 4*16
sint27          .space 4*16
sint28          .space 4*16
sint29          .space 4*16
sint30          .space 4*16

int0:       rete
            nop
            nop
            nop
int1:       rete
            nop
            nop
            nop
int2:       rete
            nop
            nop
            nop
tint0:      rete
            nop
            nop
            nop
brint0:     rete
            nop
            nop
            nop
bxint0:     rete
            nop
            nop
            nop
```

```
dmac0：    rete
           nop
           nop
           nop
dmac1：    rete
           nop
           nop
           nop
int3：     rete
           nop
           nop
           nop
hpint：    rete
           nop
           nop
           nop
dmac2：    rete
           nop
           nop
           nop
bxint1：   rete
           nop
           nop
           nop
dmac4：    rete
      nop
           nop
           nop
dmac5：    rete
           nop
           nop
           nop
RESERVED：  .space     880H
```

3.3.9 AD/DA 联合调试程序代码

AD/DA 联合调试程序实现从 AD 读取数据直接将数据从 DA 输出，使用示波器可以从 DA 输出引脚看到和 AD 相近的模拟信号，由于 DA 放大系数的原因，可能在信号幅度上有所差别。本程序使用的配置文件和中断向量文件和测试 AD 程序使用的程序一样。

```
                .mmregs
                .def        jump
                .def        start

k_sample_num    .set        256
out_data        .usect      "out_vars",k_sample_num
in_data         .usect      "in_vars",1
d_cr0_send      .usect      "cr_send",1
```

```
d_cr1_send      .usect      "cr_send",1
d_temp          .usect      "temp",1
k_cr0_send      .set        0080h
k_cr1_send      .set        0100h

k_iptr          .set        000111000b<<07
k_temp          .set        1111111b
d_pmst          .usect      "pmst",1

                .sect       "prog"
start:
                ldm         pmst,a
                and         #k_temp,a
                or          #k_iptr,a
                stl         a,ar2
                mvdm        ar2,pmst

                stm         #0001h,imr
                nop
                nop
                stm         #0002h,ifr
                nop
                rsbx        intm
                nop

                stm         #d_temp,ar2
                stm         #d_cr0_send,ar1
                st          #k_cr0_send,*ar1+
                st          #k_cr1_send,*ar1
                stm         #d_cr0_send,ar1
                ld          #out_data,ar5
                portr       0fffeh,*ar2
                portw       *ar1+,07fffh
                nop
                nop
                nop
                nop
                nop
                portw       *ar1,07fffh
                nop
                nop
                nop
                nop
                portr       0ffffh,*ar2
                nop
                nop
wait:           nop
```

```
                nop
                nop
                b       wait

jump:           rsbx    xf
                nop
                nop
                nop
                portr   07fffh,*ar5
                nop
                nop
                nop
                ssbx    xf
                nop
                nop
                nop
                portr   0fffeh,*ar2
                nop
                nop
                nop
                portw   *ar5,0bfffh
                nop
                nop
                nop
                rete
```

3.3.10 数据滤波程序代码

```
                .mmregs
                .ref    filter_start
K_DATA_SIZE     .set    256             ；输入数据个数
K_BUFFER_SIZE   .set    8               ；缓冲大小，必须是大于 a 和 b 并且是 2 的整数次幂
K_STACK_SIZE    .set    256             ；堆栈大小
K_A             .set    3               ；滤波器 a 参数的个数
K_B             .set    4               ；滤波器 b 参数的个数
K_CIR           .set    K_BUFFER_SIZE

STACK           .usect"stack",K_STACK_SIZE
SYSTEM_STACK    .set    K_STACK_SIZE+STACK

DATA_DP         .usect "filter_vars",0
filterdata      .usect "filter_vars",K_DATA_SIZE
bufferdatay     .usect "filter_vars",K_BUFFER_SIZE*2
bufferdatax     .usect "filter_vars",K_BUFFER_SIZE*2

        .data
```

```
            .global     inputdata

            .text
            .asg AR2,ORIGIN
            .asg AR3,INPUT
            .asg AR4,FILTER
            .asg AR5,OUTPUT
START:
            SSBX     FRCT
            SSBX     INTM
            LD       #DATA_DP,DP
            STM      #STACK,SP
            CALL     filter_start
            NOP
            NOP
            NOP
LOOP
            B        LOOP

           .def      b1,b2,b3,a1,a2
           .def      filter_start
b1         .set      1456H              ;   b1=0.1589
b2         .set      3D07H              ;   b2=0.4768
b3         .set      3D07H              ;   b3=0.4768
b4         .set      1456H              ;   b4=0.1589
a1         .set      －103AH            ;   a1=－0.1268
a2         .set      430FH              ;   a2=0.5239
a3         .set      －1016H            ;   a3=－0.1257

           .text
filter_start:
           STM       #K_CIR,BK
           STM       #1,AR0                    ；设置循环缓冲区大小和步长
           STM       #inputdata,ORIGIN
           STM       #bufferdatax,INPUT
           STM       #bufferdatay,FILTER
           STM       #filterdata,OUTPUT
                                               ；初始化
           RPT       #K_A－1                   ；系数 a1,a2,a3 的个数
           MVDD      *ORIGIN+,*INPUT+0%        ；初始化头 3 个 Ys
           STM       #bufferdatax,INPUT
                                               ；设置初始化滤波数据
           RPT       #K_A－1                   ；系数 a1,a2,a3 的个数
           MVDD      *INPUT+0%,*FILTER+0%      ；初始化 3 个 Ys
           STM       #bufferdatay,FILTER
           STM       #bufferdatax,INPUT
                                               ；滤波
```

```
            STM     #K_DATA_SIZE－3－1,BRC          ；设置块循环计数器
            RPTB    filter_end－1                   ；头 3 个值直接通过
            MVDD    *ORIGIN+,*INPUT
            RPT     #K_B－1－1                      ；系数 b1,b2,b3, b4 的个数
            MAR     *INPUT－0%                      ；调整输入到相应位置
            MPY     *INPUT+0%,#b4,B                 ；计算 B=b4*x(i)
            LD      B,A
            MPY     *INPUT+0%,#b3,B                 ；计算 B=b3*x(i+1)
            ADD     B,A
            MPY     *INPUT+0%,#b2,B                 ；计算 B=b2*x(i+2)
            ADD     B,A
            MPY     *INPUT+0%,#b1,B                 ；计算 B=b1*x(i+3)
            ADD     B,A                             ；计算 y(x+3)=A
            MPY     *FILTER+0%,#a3,B                ；计算 B=y(i)*a3
            ADD     B,A                             ；计算 A=A+B
            MPY     *FILTER+0%,#a2,B                ；计算 B=y(i+1)*a2
            ADD     B,A                             ；计算 A=A+B
            MPY     *FILTER+0%,#a1,B                ；计算 B=y(i+2)*a1
            ADD     B,A                             ；计算 y(x+3)=A+B
            STH     A,*FILTER－0%                   ；为下一次滤波保存数据
            STH     A,*OUTPUT+                      ；输出数据到 OUTPUT 的下一个单元
            MAR     *FILTER－0%                     ；调整滤波器到相应位置
filter_end: NOP
            RET
            .end                                    ；程序代码结束
```

3.4 案例总结

本案例详细介绍了一种中速数据采集的系统设计。该系统以 TMS320C5409 为核心处理器，实现 1MHz 信号的采样、滤波并从 DA 输出。TMS320C5409 的时钟周期为 100MHz，实现对 AD 的采样，包括控制 AD、响应中断、读取数据一共大概需要花销 20 条指令周期，这样留给处理器做数字信号的算法处理时间就不是很多。采用降低 AD 的采样频率和减小算法的复杂性是解决这一问题常用的方法。可以根据具体需要选择适合采样系统的方法。

联系方式

集团官网：www.hqyj.com　　嵌入式学院：www.embedu.org　　移动互联网学院：www.3g-edu.org

企业学院：www.farsight.com.cn　　物联网学院：www.topsight.cn　　研发中心：dev.hqyj.com

集团总部地址：北京市海淀区西三旗悦秀路北京明园大学校内 华清远见教育集团

北京地址：北京市海淀区西三旗悦秀路北京明园大学校区，电话：010-82600386/5

上海地址：上海市徐汇区漕溪路银海大厦 A 座 8 层，电话：021-54485127

深圳地址：深圳市龙华新区人民北路美丽 AAA 大厦 15 层，电话：0755-22193762

成都地址：成都市武侯区科华北路 99 号科华大厦 6 层，电话：028-85405115

南京地址：南京市白下区汉中路 185 号鸿运大厦 10 层，电话：025-86551900

武汉地址：武汉市工程大学卓刀泉校区科技孵化器大楼 8 层，电话：027-87804688

西安地址：西安市高新区高新一路 12 号创业大厦 D3 楼 5 层，电话：029-68785218